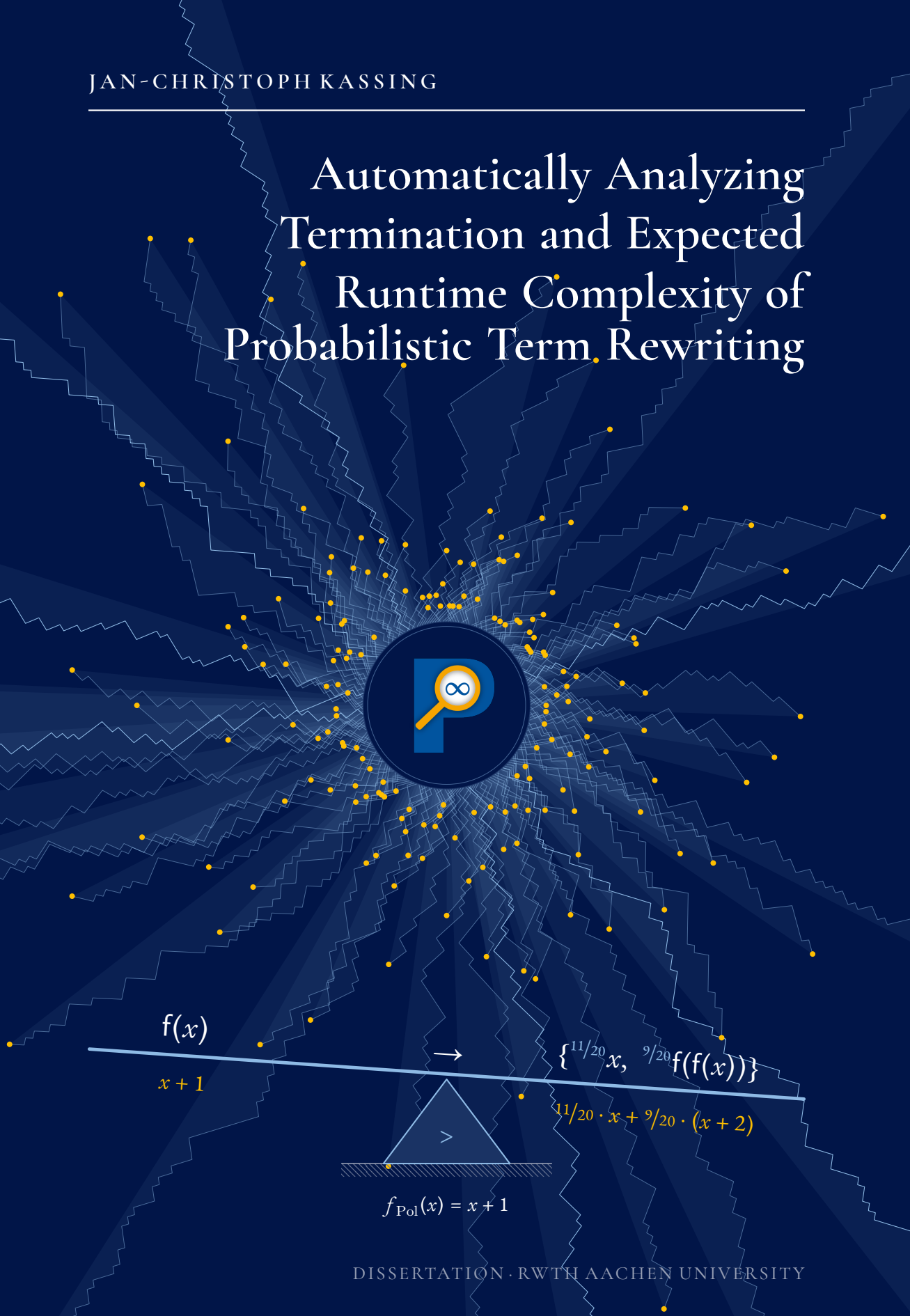
JAN-CHRISTOPH KASSING

# Automatically Analyzing Termination and Expected Runtime Complexity of Probabilistic Term Rewriting




DISSERTATION · RWTH AACHEN UNIVERSITY

# About the Cover

Each line on the cover is an individual run of a probabilistic program (a *random walk*). The program keeps a counter that goes down by one in 11 out of 20 cases and up by one otherwise. Each run starts with the counter being 3 at the logo in the middle and moves outward with each computation step. A gold dot marks where the counter reaches zero and the run stops. Because the coin slightly favors "down", every run stops with probability 1. Infinite runs exist, but the probability of all infinite runs is 0. The bottom of the cover shows why: every flip brings the program a little closer to its end in expectation. This thesis develops methods that prove such mathematical guarantees fully automatically for programs that make random choices, implemented in the tool AProVE, whose logo sits in the center.

# Automatically Analyzing Termination and Expected Runtime Complexity of Probabilistic Term Rewriting

Von der Fakultät für Informatik der RWTH Aachen University
zur Erlangung des akademischen Grades

eines Doktors der Naturwissenschaften

genehmigte Dissertation

vorgelegt von

Jan-Christoph Kassing, Master of Science

Berichter:

Prof. Jürgen Giesl

Prof. Ugo Dal Lago

Tag der mündlichen Prüfung: 01.10.2026

Diese Dissertation ist auf den Internetseiten der Universitätsbibliothek verfügbar.

# Abstract

Probabilistic programs (i.e., programs that can make random choices during execution) are central to modern computing. They are used for randomized algorithms, Bayesian inference, cryptographic protocols, and many other applications. While randomness brings many benefits, it also introduces new challenges for formal verification. In particular, a probabilistic program may terminate with probability one (almost-sure termination, AST) while still admitting infinite executions. Moreover, even when termination is guaranteed with probability one, the expected runtime may be infinite, motivating the stronger notion of strong almost-sure termination (SAST) where for every initial state, the expected runtime is bounded by a finite constant, uniformly for all resolutions of the non-determinism.

In this thesis, we present a framework for the automatic analysis of AST and SAST for probabilistic term rewrite systems (PTRSs), a rule-based functional programming language that combines non-determinism with probabilistic choice. Since programs in languages such as Java, C, Haskell, and Prolog can be automatically transformed into term rewrite systems, approaches based on term rewriting have broad applications.

We present several contributions towards this goal. Most notable is the annotated dependency pair framework, a modular, divide-and-conquer method for proving AST and SAST. We extend this framework to derive upper bounds on the expected runtime and complement it with automated techniques for disproving AST and SAST via reductions to stochastic processes such as random walks. Moreover, we investigate several other important properties for termination analysis. First, we consider relations between different evaluation strategies: when does AST for an innermost (or call-by-value) evaluation strategy imply AST for an arbitrary evaluation strategy? This is an important question for automatic termination analysis as it is easier to prove termination for innermost evaluation than for an arbitrary evaluation strategy. Second, we show how the combination of two disjoint systems behaves if both original systems are AST or SAST. Finally, we present restrictions on PTRSs under which AST becomes decidable, yielding the first decision procedure for AST of a non-trivial subclass of PTRSs.

All techniques to analyze PTRSs have been implemented in the tool AProVE and demonstrate strong performance in the annual termination competition.

# Zusammenfassung

Probabilistische Programme (d.h. Programme, die während ihrer Ausführung zufällige Entscheidungen treffen können) sind ein zentraler Bestandteil der modernen Informatik. Sie werden für randomisierte Algorithmen, Bayes'sche Inferenz, kryptographische Protokolle und vieles mehr eingesetzt. Während die Integration von Zufall in Programme viele Vorteile mit sich bringt, führt sie auch zu neuen Herausforderungen für die formale Verifikation. Insbesondere kann ein probabilistisches Programm mit Wahrscheinlichkeit eins terminieren (engl. almost-sure termination, AST) und dennoch unendliche Ausführungen zulassen. Darüber hinaus kann die erwartete Laufzeit selbst dann unendlich sein, wenn AST garantiert ist. Dies motiviert den Begriff der starken fast-sicheren Terminierung (engl. strong almost-sure termination, SAST), bei der die erwartete Laufzeit jedes Anfangszustands für alle Auflösungen des Nichtdeterminismus durch eine gemeinsame endliche Konstante beschränkt ist.

In dieser Dissertation präsentieren wir ein Framework zur automatischen Analyse von AST und SAST für probabilistische Termersetzungssysteme (PTESe), eine regelbasierte funktionale Programmiersprache, die Nichtdeterminismus mit probabilistischen Entscheidungen kombiniert. Da Programme in Sprachen wie Java, C, Haskell und Prolog automatisch in Termersetzungssysteme transformiert werden können, haben Ansätze, die auf Termersetzung basieren, ein breites Anwendungsspektrum.

Hierzu präsentieren wir zahlreiche Beiträge. Der bedeutendste ist das Annotated-Dependency-Pair-Framework, eine modulare Divide-and-Conquer-Methode zum Beweis von AST und SAST. Wir erweitern dieses Framework, um obere Schranken für die erwartete Laufzeit herzuleiten, und ergänzen es durch automatisierte Techniken zum Widerlegen von AST und SAST mittels Reduktionen auf stochastische Prozesse wie stochastische Irrfahrten (engl. random walks). Darüber hinaus untersuchen wir mehrere weitere wichtige Eigenschaften für die Terminierungsanalyse. Zum einen betrachten wir Beziehungen zwischen verschiedenen Auswertungsstrategien: Wann impliziert AST für eine Innermost-Auswertungsstrategie (bzw. Call-by-Value-Auswertung) auch AST für eine beliebige Auswertungsstrategie? Dies ist eine wichtige Frage für die automatische Terminierungsanalyse, da es einfacher ist, Terminierung für Innermost-Auswertungen zu beweisen als für eine beliebige Auswertungsstrategie. Zum anderen zeigen wir, wie sich die Kombination zweier disjunkter Systeme verhält, wenn beide ursprünglichen Systeme AST bzw. SAST sind. Schließlich präsentieren wir Einschränkungen von PTESen, unter denen AST entscheidbar wird, was zur ersten Entscheidungsprozedur für AST auf einer nichttrivialen Teilklasse von PTESen führt.

Alle Techniken zur Analyse von PTESen wurden im Tool AProVE implementiert und zeigen ihre Leistungsstärke in der jährlichen Termination Competition.

# Acknowledgments

First and foremost, I would like to express my profound appreciation to my supervisor, Jürgen Giesl. Thank you for introducing me to the realm of term rewriting, for giving me the freedom to follow my own ideas, and for the countless hours you spent reading my drafts and improving them with your comments and suggestions. I could not have wished for a better supervisor during these years!

Second, I am grateful to Ugo Dal Lago for agreeing to act as my second examiner, and for the interesting discussions we had at the summer school on rewriting and at PPDP in southern Italy. Some ideas from the discussions are reflected in this thesis.

Doing a PhD can sometimes feel very lonely, but I was fortunate to spend these years surrounded by wonderful colleagues and friends. Many thanks to all current and former members of our research group "Programming Languages and Verification" for all the interesting discussions and all the fun we had during the hard times. Thanks to Marcel Hark, Jera Hensel, and David Korzeniewski for a nice programming lecture in my first semester, and the opportunity to become a student assistant for the course in the years after. Thanks to Stefan Dollase for the opportunity to become a research assistant after my seminar. Thanks to Daniel Cloerkes for the work together in all the courses, Florian Frohn for all the time we spent trying to understand AProVE, Moritz Leven Rosarius for being a funny office mate to talk to, Nils Lommen for nearly winning the soccer tournament of the computer science departments together, Éléanore Meyer for explaining basic concepts of probability theory whenever I asked, and most recently Arian Kulmer, who has just joined the group, and hopefully finds the same supportive atmosphere here that I was fortunate to experience.

During my time as a (PhD) student, I met many interesting and wonderful people, and I could write a separate paragraph about each of them. However, to keep these acknowledgments brief, I will only mention the most important names.

My thanks go to everyone at the MOVES group and Theory of Hybrid Systems group and all the guests I have met during my time in Aachen for the help, the shared coffee breaks, and a very welcoming atmosphere. Thanks to László Antal, Lina Gerlach, József Kovács, Jasper Nalbach, Valentin Promies, Nicolai Radke, and Agustin Viglianco for the fiercely contested table soccer matches and the support in the university soccer tournament. Thanks to Alexander Bork, Hannah Mertens, and Tobias Winkler for the great time in Luxembourg, thanks to Emma Ahrens for the time in Birmingham, thanks to Darion Haase for the time in Lisbon and the grading of my first-semester exercise sheets, and also thanks to the rest of the groups and all the guests: Erika Ábrahám, Thom Badings, Kevin Batz, Jana Berger, Diane Cauquil, Mingshuai Chen,

Ira Fesefeldt, Xaver Fink, Christina Gehnen, Roy Hermanns, Benjamin Kaminski, Joost-Pieter Katoen, Lutz Klinkenberg, Maxime Magnin, Ivo Melse, Jan Michnia, Thomas Noll, Kittiphon Phalakarn, Lisa Pühl, Tim Quatmann, Philipp Schroer, Jip Spel, Lena Verscht, and Daniel Zilken. Moreover, thanks to Elke Ohlenforst for keeping everything running smoothly behind the scenes.

Apart from the research groups on our floor, I am also grateful to all the people from the UnRAVeL Research Training Group. Thanks to Helen Bolke-Hermanns and Birgit Willms for organizing all the workshops and retreats, and their help with everything! Thanks to my fellow PhD students who suffered and celebrated together with me: Mariia Anapolska, Tabea Brandt, Paul Brunzema, Katharina Eickhoff, Tamme Emunds, Felix Engelhardt, Julia Feith, Qihui Feng, Nadine Friesen, Vincent Grande, Christoph Grüne, Henrik Hose, Andreas Klinger, A. Marcus, Maren Maus, Komal Muluk, Philipp Pabst, Eran Rosenbluth, Michael Scholkemper, Tim Seppelt, Christoph Standke, Antonios Vasileiou, Tarik Viehmann, and Solveig Wittig.

I also want to mention and thank the students who worked with me on their theses: Fabian Berkner, Michelle Dahmen, Moritz Heim, Moritz van Hettinga, Nils Maasch, Kiran Monteiro Pai, Henri Nagel, Thomas Neudörfl, Luca Reinartz, Jonas Säuberlich, Arion Scheid, Alexander Schlecht, Alwin Schüßler, Ole Siemer, Tobias Sokolowski, Leon Spitzer, Grigory Vartanyan, David Weber, and Ivan Zlatin.

Beyond Aachen, I would like to thank Carsten Fuhs, Cynthia Kop, Aart Middeldorp, Johannes Niederhauser, Vincent van Oostrom, Teppei Saito, René Thiemann, Deivid Vale, and everyone else from the rewriting community for making conferences something I looked forward to, and where I always learned something new. Also thanks to Fabian Mitterwalner and Jonas Schöpf for the nice walks around Obergurgl. The same holds for Antonia Swiridoff, Balasz Toth and everyone I met at the rewriting summer school, as well as Anna, Moritz, Théo, and everyone from the SAT and SMT summer school. I haven't forgotten the people from the various Dagstuhl seminars either, e.g., the HPI crowd including Tom Beckmann, Lukas Boehme, and Mustafa Ghani, whom I visited later during my PhD. Thanks to Thomas Hader for all the fun and both of the stars we saw in the sky as well!

Another important part of my studies were the early days, when I met a lovely group of fellow first-semester bachelor students. Thanks for all the fun we had in our poker group: Christopher, Felix, Freddy, Jacob, Jenny, Jonas, Jörn, Lea, the Lennager, Lucky, Robert, Tom, Witti. Also thanks to Paul Wirkus, Johannes Lehmann, Adrian Gallus, and Thomas Becker.

Outside of academia, I want to thank Anton, Chrissi, Hendryk, Lea, Malte, and all the people I annoyed with my dissertation while on vacation. Additionally, thanks to my friends from Soest: Anka, Bent, Florian, Florian, Jan, Johannes, Marten, Marvin, Morris, Robin, Sarah, Schnelge, Thomas, Till, Till, Till, Tim, Timon, Chrissi, Chrissi, Luca, Max, Niklas, Yannik, etc. Thanks to all the people from the several handball teams I played in. The time completely apart from research to clear my head was one of the reasons I could continue for so long!

Finally, and most importantly, I want to thank my family for their unconditional support throughout all these years and all of my crazy ideas. I thank my brother, who showed me that an academic path is possible, and my sister for all the love and support over the years. And of course, my girlfriend Eva: You are one of the few people so far who are capable of stopping me from doing research just for more time together!

# Preface

*"Coding is largely solved."*

— Boris Cherny
Head of Claude Code at Anthropic

The aim of this thesis is to develop *automatic techniques* for analyzing *termination* and *runtime behavior* of programs that contain *probabilistic choices*. In a world where code is increasingly generated automatically, e.g., through program synthesis, AI-assisted development, or template-based code generators, and where the software engineer may not always comprehend what the code in front of them is doing, *automatic formal verification* is more pressing than ever. While there already exists plenty of work on the automatic analysis of ordinary programs, the automatic analysis of probabilistic programs is comparatively recent. Incorporating probabilities into algorithms is motivated by its successful usage within machine learning, Bayesian inference, cryptographic protocols, and to deal with uncertainty in data. Additionally, probabilistic programs can have strictly better runtime behavior compared to their deterministic variants. An example is the well-known quicksort algorithm. Here, on lists with pairwise distinct elements, randomized quicksort achieves a lower expected number of comparisons in the worst case than any deterministic variant that chooses its pivots at fixed positions. While the analysis of *imperative* probabilistic programs already has many well-established results, the fully automatic analysis of *functional* probabilistic programs is comparatively sparse. Instead of control structures like while-loops, a functional program is purely based on recursion and pattern matching.

Even if an algorithm, like *quicksort*, is theoretically correct and always terminates, a concrete implementation—a program—whether written by humans or machines, can still be incorrect and run infinitely long. So in practice one needs to verify the correctness of programs, and the analysis of such programs differs drastically depending on the programming language used. But what is the "best" programming language to analyze? The one with the most active users? The one that is the easiest to analyze? In this thesis, we focus on *term rewrite systems* (TRSs), a formalism for functional programs based on pattern matching that is well studied in the non-probabilistic setting. TRSs are easier to analyze than Java or C programs, since their operational semantics is given by a simple rewrite relation that abstracts away low-level details like memory layout and control-flow constructs. Consequently, one can concentrate the effort on developing more powerful analysis techniques for TRSs, and only needs to develop a single transformation for each of the complex languages to this core model.

This thesis develops the fundamental theory behind *probabilistic* TRSs (or PTRSs) where the inherent non-deterministic selection of a redex and a rule in TRSs is extended by an additional probabilistic choice afterwards. So the selection of the redex and the rule remains non-deterministic, but after this selection one has multiple probabilistic choices for the reduct. While many of the techniques used for probabilistic imperative programs can be adapted for PTRSs, it requires a lot of effort to lift the ideas, the results, and their corresponding proofs from the imperative to the functional setting, and it does not always work out as expected. Similarly, many of the results for non-probabilistic TRSs can be adapted to PTRSs, but these adaptations suffer from the same problems.

## Detailed Structure and Contributions

In the following, we briefly describe the contents of each chapter of this thesis and the rationale behind it. Since most of this work has already been published and presented at several conferences, each of Chapters 3–9 begins with a list of its contributions, highlighting both the improvements over existing work from the literature and the results that have not been published before. Moreover, each of these chapters ends with an evaluation (where an implementation is available) illustrating the power of the presented techniques, followed by a discussion of related work.

**Chapter 1** In the first chapter we set the stage and introduce the general concept of fully automatic program verification. We explain different approaches for verifying the correctness of software and we present an overview of established techniques for analyzing complex Java or C codebases based on a transformation to TRSs. Moreover, we mention the relation of PTRSs to several other probabilistic programming formalisms like the probabilistic guarded command language (pGCL) or the probabilistic $\lambda$-calculus.

**Chapter 2** In the second chapter we recapitulate all necessary notions of probability theory needed to understand this thesis. Overall, our goal is to develop a mathematically sound and formal theory for probabilistic rewrite systems. Additionally, we introduce several well-known stochastic systems that have been investigated in the literature for decades, namely Markov chains, Markov decision processes, and random walks. We use these systems to demonstrate the difficulties one has to take into account when defining probability spaces, termination probabilities, and expected runtimes for stochastic systems. Moreover, we illustrate how to analyze stochastic systems and even rely on these analyses in later parts of this thesis. Chapter 2 only recapitulates known results and formalisms.

**Chapter 3** In the third chapter we introduce the programming language we are interested in: *term rewrite systems* (TRSs). This chapter is the backbone of this entire thesis and introduces all notions and notations used in the remaining chapters. We recapitulate all important notions from the non-probabilistic setting and lift them to *probabilistic term rewrite systems* (PTRSs). While the notion of a PTRS is not a novel contribution—this has already been defined, e.g., in [BK02, BG05, ADLY20]—it has often been introduced from a specific perspective only. We compile the relevant literature into this chapter, which includes all the key information already known, and, in particular, also presents novel findings regarding these systems. To be precise,

we explain what a PTRS is, how we can measure the *probability of termination* for such a system, and how to measure the *expected runtime complexity*. Finally, we give a first glimpse of how to automatically prove that the termination probability is 1, called almost-sure termination (AST), and how to prove that for every initial term, the expected runtime is bounded by a finite constant, uniformly for all resolutions of the non-determinism, called strong almost-sure termination (SAST). Chapter 3 recapitulates many important parts from the literature, but also contains both novel contributions and contributions published in

[KG23] Jan-Christoph Kassing and Jürgen Giesl. Proving Almost-Sure Innermost Termination of Probabilistic Term Rewriting Using Dependency Pairs. In *Proc. CADE '23*, LNCS 14132, pages 344–364, 2023. doi:10.1007/978-3-031-38499-8_20.

[KG25] Jan-Christoph Kassing and Jürgen Giesl. From Innermost to Full Probabilistic Term Rewriting: Almost-Sure Termination, Complexity, and Modularity. *Log. Methods Comput. Sci.*, 21(4), 2025. doi:10.46298/lmcs-21(4:28)2025.

[KSG25] Jan-Christoph Kassing, Leon Spitzer, and Jürgen Giesl. Dependency Pairs for Expected Innermost Runtime Complexity and Strong Almost-Sure Termination of Probabilistic Term Rewriting. In *Proc. PPDP '25*, pages 10:1–10:14, 2025. doi:10.1145/3756907.3756917.

**Chapter 4** In the fourth chapter we relate different evaluation strategies in terms of termination and runtime behavior. An evaluation strategy defines how to evaluate a given term. One of the most widely used evaluation strategies is the innermost strategy, where we have to rewrite innermost reducible expressions. Additionally, innermost evaluation is the easiest to analyze in practice for proving termination. We present sufficient criteria that ensure equivalence between almost-sure termination with respect to innermost evaluation and with respect to an arbitrary evaluation strategy. We focus on criteria that can be checked automatically to use them within our overall verification procedure. Chapter 4 contains both novel contributions and contributions published in

[KFG24] Jan-Christoph Kassing, Florian Frohn, and Jürgen Giesl. From Innermost to Full Almost-Sure Termination of Probabilistic Term Rewriting. In *Proc. FoSSaCS '24*, LNCS 14575, pages 206–228, 2024. doi:10.1007/978-3-031-57231-9_10.

[KG25] Jan-Christoph Kassing and Jürgen Giesl. From Innermost to Full Probabilistic Term Rewriting: Almost-Sure Termination, Complexity, and Modularity. *Log. Methods Comput. Sci.*, 21(4), 2025. doi:10.46298/lmcs-21(4:28)2025.

**Chapter 5** In the fifth chapter we consider the problem of combining different systems and how it impacts the termination probability and the expected runtime. For example, we show that the sequential combination of two almost surely terminating programs $A$ and $B$ (first executing $A$, then $B$) remains almost surely terminating. Moreover, since we investigate PTRSs, which are simply sets of rewrite rules, we consider how the termination probability behaves if we take the union of such sets of rewrite rules.

For example, we show that for innermost evaluation, the union of two almost surely terminating systems remains almost surely terminating, provided that the two systems are completely independent and do not share any functions or data. In addition to the termination probability and almost-sure termination, we also consider expected runtime and strong almost-sure termination. Chapter 5 contains both novel contributions and contributions published in

[KG25] Jan-Christoph Kassing and Jürgen Giesl. From Innermost to Full Probabilistic Term Rewriting: Almost-Sure Termination, Complexity, and Modularity. *Log. Methods Comput. Sci.*, 21(4), 2025. doi:10.46298/lmcs-21(4:28)2025.

**Chapter** 6 In the sixth chapter we present the first decision procedure for almost-sure termination of a non-trivial subclass of PTRSs. To this end, we present another stochastic system from the literature, namely stochastic context-free grammars, where almost-sure termination is decidable. We present criteria on the PTRS ensuring that it can be automatically transformed into a stochastic context-free grammar such that the termination probability remains the same. In this way, we obtain a decision procedure for PTRSs that satisfy these criteria. Chapter 6 is completely novel, and a paper about its content has been submitted.

**Chapter** 7 In the seventh chapter we present the heart of this thesis and of the automatic verification of PTRSs: *dependency pairs*. Dependency pairs are one of the most powerful techniques to prove termination of non-probabilistic TRSs. The core principle behind dependency pairs is that a recursive function terminates if and only if its arguments strictly decrease (with respect to some well-founded order) with each recursive call. In the probabilistic setting, all recursive calls of a rule have to be considered simultaneously, and the arguments only need to decrease in expectation. The *annotated dependency pair framework* is a divide-and-conquer framework that proves almost-sure termination and can split a large problem into simpler sub-problems whose termination can then be analyzed independently of the other sub-problems. Chapter 7 contains both novel contributions and contributions published in

[KG23] Jan-Christoph Kassing and Jürgen Giesl. Proving Almost-Sure Innermost Termination of Probabilistic Term Rewriting Using Dependency Pairs. In *Proc. CADE '23*, LNCS 14132, pages 344–364, 2023. doi:10.1007/978-3-031-38499-8_20.

[KDG24] Jan-Christoph Kassing, Stefan Dollase, and Jürgen Giesl. A Complete Dependency Pair Framework for Almost-Sure Innermost Termination of Probabilistic Term Rewriting. In *Proc. FLOPS '24*, LNCS 14659, pages 62–80, 2024. doi:10.1007/978-981-97-2300-3_4.

[KG24] Jan-Christoph Kassing and Jürgen Giesl. Annotated Dependency Pairs for Full Almost-Sure Termination of Probabilistic Term Rewriting. In *Principles of Verification: Cycling the Probabilistic Landscape: Essays Dedicated to Joost-Pieter Katoen on the Occasion of His 60th Birthday*, LNCS 15260, pages 339–366, 2024. doi:10.1007/978-3-031-75783-9_14.

[KG26] Jan-Christoph Kassing and Jürgen Giesl. The Annotated Dependency Pair Framework for Almost-Sure Termination of Probabilistic Term Rewriting. *Sci. Comput. Program.*, 251:103417, 2026. doi:10.1016/j.scico.2025.103417.

**Chapter 8** In the eighth chapter we adapt the previously introduced dependency pair framework to infer upper bounds on the expected number of innermost rewrite steps of a PTRS. This adaptation not only requires a completely reworked framework itself, but all of the techniques within the dependency pair framework have to be adjusted as well. While the idea of the techniques remains the same, the concrete result and the proof of soundness differ drastically. Chapter 8 contains both novel contributions and contributions published in

[KSG25] Jan-Christoph Kassing, Leon Spitzer, and Jürgen Giesl. Dependency Pairs for Expected Innermost Runtime Complexity and Strong Almost-Sure Termination of Probabilistic Term Rewriting. In *Proc. PPDP '25*, pages 10:1–10:14, 2025. doi:10.1145/3756907.3756917.

**Chapter 9** In the ninth chapter we develop the first automatic techniques to disprove almost-sure termination of PTRSs. While proving termination ensures the absence of bugs (i.e., infinite loops), disproving termination can be used to find bugs in programs. Disproving almost-sure termination can be done by reducing the PTRS to stochastic systems that are known not to be almost surely terminating, like certain types of random walks. The same idea can be used to show that the expected runtime of a PTRS is infinite, i.e., to disprove positive and strong almost-sure termination. Chapter 9 contains both novel contributions and contributions published in

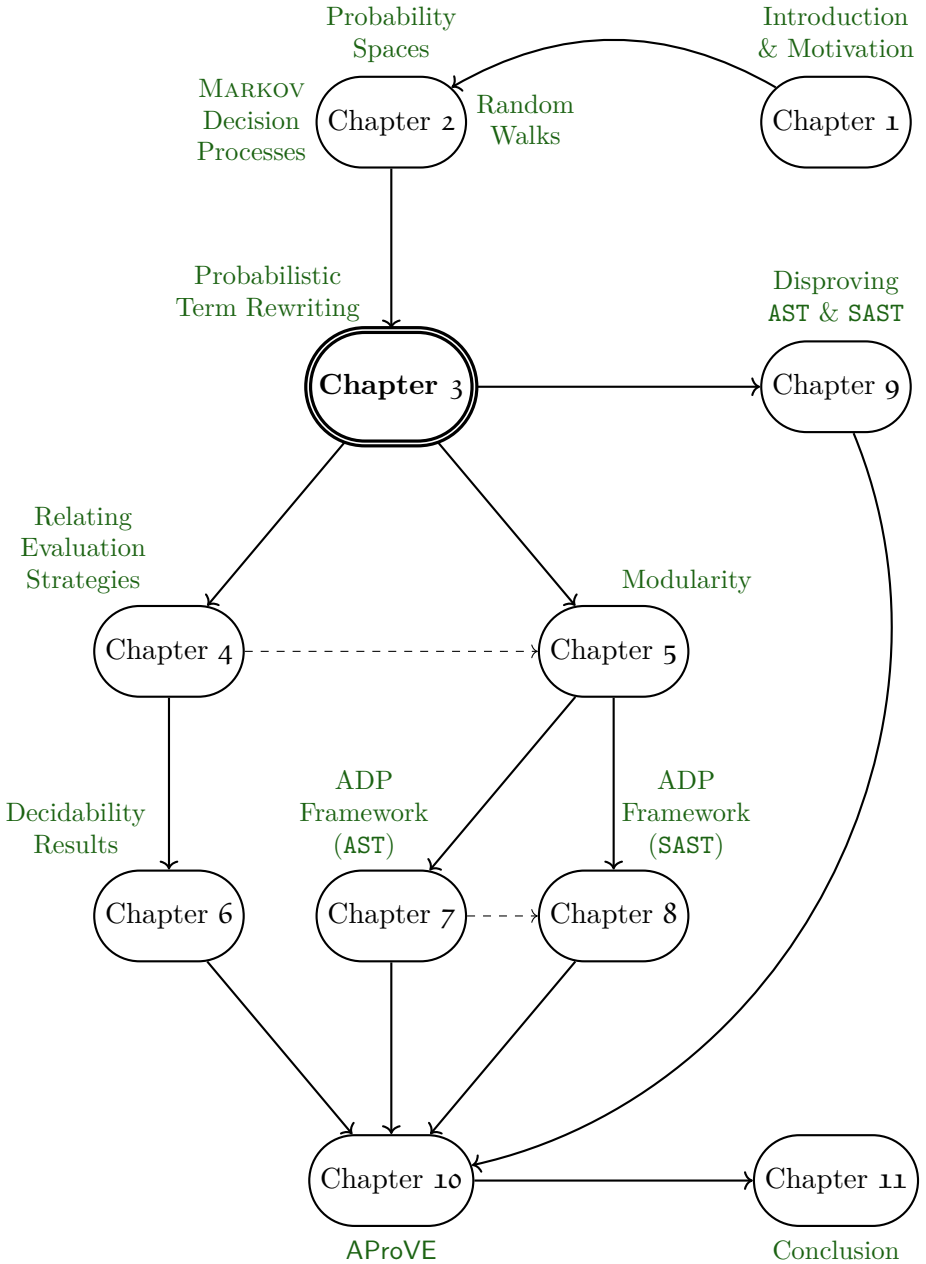
[KNSG26] Jan-Christoph Kassing, Henri Nagel, Alexander Schlecht, and Jürgen Giesl. Disproving (Positive) Almost-Sure Termination of Probabilistic Term Rewriting via Random Walks. In *Proc. IJCAR '26*, LNCS 16689, pages 304–323, 2026. doi:10.1007/978-3-032-32592-1_18.

**Chapter 10** In the tenth chapter we introduce the tool AProVE where all of our contributions have been implemented. AProVE is a powerful system for automated termination, complexity, and safety proofs of several variations of term rewrite systems, as well as other formalisms including imperative, functional, and logic programs. We extended AProVE with the probabilistic termination and runtime analysis we present in Chapters 6–9. First, we explain the overall strategy of how to analyze PTRSs automatically, and then, we evaluate all of the techniques on the benchmarks of the Termination Problem Database (TPDB) [TPDB], which is the benchmark set used in the annual termination competition. AProVE participates in the termination competition [GRS+19], competing with other state-of-the-art termination tools in several termination categories.[1] We show that the performance of AProVE on this benchmark set is quite strong. Chapter 10 is completely novel and there are plans to submit a tool paper covering the new strengths of AProVE at the end of 2026.

**Chapter 11** Finally, we conclude the thesis by mentioning some additional future work and recapitulating the most important parts.

[1] The category concerning probabilistic rewriting is currently only a demonstration category, since AProVE is its only participant.

**Relational Structure** The structure of this thesis is described in the following diagram. Normal arrows ($A \to B$) describe the order in which the chapters should be read (first $A$, then $B$). Dashed arrows ($A \dashrightarrow B$) indicate optional dependencies: $A$ is not required to understand the core ideas of $B$, but $A$ provides useful background or is used within some result of $B$. Chapter 3 is highlighted as it is the backbone behind the whole thesis and contains the foundational definitions used everywhere else.



## Note on the Contributions by the Author

In this section, I present a complete list of all my peer-reviewed publications. Besides the publications that have already been mentioned before, there are further publications that are not covered by this thesis. Nevertheless, all my publications emerged from the initial task I was given in my PhD, namely investigating the automatic formal verification of probabilistic rewrite systems. Before presenting the list of my publications, I would like to emphasize that research in this area is far from complete, and several aspects of probabilistic rewrite systems remain unresolved. Moreover, while I developed the main results including their proofs and implementations in all these publications, these publications are the result of collaborations with many co-authors, to whom I am very grateful for their valuable contributions.

## List of Publications Covered in This Thesis

[KG23] Jan-Christoph Kassing and Jürgen Giesl. Proving Almost-Sure Innermost Termination of Probabilistic Term Rewriting Using Dependency Pairs. In *Proc. CADE '23*, LNCS 14132, pages 344–364, 2023. doi:10.1007/978-3-031-38499-8_20.

[KDG24] Jan-Christoph Kassing, Stefan Dollase, and Jürgen Giesl. A Complete Dependency Pair Framework for Almost-Sure Innermost Termination of Probabilistic Term Rewriting. In *Proc. FLOPS '24*, LNCS 14659, pages 62–80, 2024. doi:10.1007/978-981-97-2300-3_4.

[KFG24] Jan-Christoph Kassing, Florian Frohn, and Jürgen Giesl. From Innermost to Full Almost-Sure Termination of Probabilistic Term Rewriting. In *Proc. FoSSaCS '24*, LNCS 14575, pages 206–228, 2024. doi:10.1007/978-3-031-57231-9_10.

[KG24] Jan-Christoph Kassing and Jürgen Giesl. Annotated Dependency Pairs for Full Almost-Sure Termination of Probabilistic Term Rewriting. In *Principles of Verification: Cycling the Probabilistic Landscape: Essays Dedicated to Joost-Pieter Katoen on the Occasion of His 60th Birthday*, LNCS 15260, pages 339–366, 2024. doi:10.1007/978-3-031-75783-9_14.

[KSG25] Jan-Christoph Kassing, Leon Spitzer, and Jürgen Giesl. Dependency Pairs for Expected Innermost Runtime Complexity and Strong Almost-Sure Termination of Probabilistic Term Rewriting. In *Proc. PPDP '25*, pages 10:1–10:14, 2025. doi:10.1145/3756907.3756917.

[KG25] Jan-Christoph Kassing and Jürgen Giesl. From Innermost to Full Probabilistic Term Rewriting: Almost-Sure Termination, Complexity, and Modularity. *Log. Methods Comput. Sci.*, 21(4), 2025. doi:10.46298/lmcs-21(4:28)2025.

[KG26] Jan-Christoph Kassing and Jürgen Giesl. The Annotated Dependency Pair Framework for Almost-Sure Termination of Probabilistic Term Rewriting. *Sci. Comput. Program.*, 251:103417, 2026. doi:10.1016/j.scico.2025.103417.

[KNSG26] Jan-Christoph Kassing, Henri Nagel, Alexander Schlecht, and Jürgen Giesl. Disproving (Positive) Almost-Sure Termination of Probabilistic Term Rewriting via Random Walks. In *Proc. IJCAR '26*, LNCS 16689, pages 304–323, 2026. doi:10.1007/978-3-032-32592-1_18.

## List of Additional Publications

**Dependency Pairs for Relative Termination** A combination of two TRSs (a *main* TRS $\mathcal{R}$ and a *base* TRS $\mathcal{B}$) is *relatively terminating* if there exists no rewrite sequence that uses infinitely many steps with rules from $\mathcal{R}$ (whereas rules from $\mathcal{B}$ may be used infinitely often). Relative termination of TRSs has been studied for decades [Ges90], and approaches based on relative rewriting are used for many applications, e.g., in complexity analysis [Fuh19, ZK14, BFG22, NFB+17, FGH+17], for proving confluence [HM11, KH12], for certifying confluence proofs [NFZ17], and for proving liveness properties [KZ05].

However, while techniques and tools for analyzing ordinary termination of TRSs are very powerful due to the use of the dependency pair framework, an adaptation of this framework to analyze relative termination was not known. A long-standing open problem regarding dependency pairs was Problem #106 of the RTA List of Open Problems [DT]: *Can we use the dependency pair method to prove relative termination?* A first major step towards an answer to this question was presented in [INVY17] by giving criteria for $\mathcal{R}$ and $\mathcal{B}$ that allow the use of ordinary dependency pairs (as defined for proving termination of TRSs) for relative termination. We managed to use our adaptation of dependency pairs for the probabilistic setting to solve Problem #106 and developed the first dependency pair framework specifically targeting relative termination.

[KVG24] Jan-Christoph Kassing, Grigory Vartanyan, and Jürgen Giesl. A Dependency Pair Framework for Relative Termination of Term Rewriting. In *Proc. IJCAR '24*, LNCS 14740, pages 360–380, 2024. doi:10.1007/978-3-031-63501-4_19.

**Weighted Rewriting** While adapting techniques from the non-probabilistic to the probabilistic setting, we came across the question of whether numerous different analyses in the area of rewriting, computation, logic, and deduction are "inherently similar", i.e., whether they can all be seen as special instances of a uniform framework. Analogous research has been done, e.g., in the database and logic community, where there have been numerous approaches to *provenance analysis*, i.e., to analyze not only whether certain results hold but also their explainability (see, e.g., [CCT09, DG19, GKT07]). In these works, *semirings*—an algebraic structure with an addition and a multiplication—are often used to obtain information beyond just satisfiability of a query. For instance, they may be used to compute the confidence in an answer or to calculate the cost of proving satisfiability.

In this work we generalize the idea of evaluating a database query within a given semiring to abstract reduction systems, i.e., a set $A$ together with a binary relation on $A$. Moreover, we present first results on how to automatically prove that the weight of a computation is bounded by some finite constant, and how to disprove it. Finally, we show that PTRSs can also be expressed using this novel formalism.

[AKGK25] Emma Ahrens, Jan-Christoph Kassing, Jürgen Giesl, and Joost-Pieter Katoen. Weighted Rewriting: Semiring Semantics for Abstract Reduction Systems. In *Proc. FSCD '25*, LIPIcs 337, pages 6:1–6:21, 2025. doi: 10.4230/LIPIcs.FSCD.2025.6.

# Contents

**Abstract** **iii**

**Zusammenfassung** **v**

**Acknowledgments** **vii**

**Preface** **ix**
Detailed Structure and Contributions . . . . . x
Note on the Contributions by the Author . . . . . xv

**Contents** **xviii**

**1 Introduction** **1**
1.1 Automatic Formal Termination and Runtime Analysis . . . . . 2
1.2 Term Rewriting . . . . . 4
1.3 Probabilistic Programs and Formalisms . . . . . 5

**2 Probability Theory** **9**
2.1 Probability Spaces . . . . . 10
2.2 Random Variables . . . . . 14
2.3 Random Walks and Random Walk Programs . . . . . 18
2.3.1 Random Walks . . . . . 19
2.3.2 Random Walk Programs . . . . . 22
2.4 Markov Chains and Markov Decision Processes . . . . . 27
2.4.1 Markov Chains . . . . . 28
2.4.2 Markov Decision Processes . . . . . 34

**3 Probabilistic Term Rewriting** **39**
3.1 Abstract Reduction Systems . . . . . 41
3.2 Term Rewriting . . . . . 44
3.2.1 Syntax and Semantics . . . . . 45
3.2.2 Complexity . . . . . 53
3.2.3 Evaluation Strategies . . . . . 54
3.2.4 Non-Termination Witnesses . . . . . 56
3.2.5 Proving Termination and Complexity via Interpretations . . . 59
3.3 Probabilistic Abstract Reduction Systems . . . . . 64
3.3.1 Notions of Probabilistic Termination . . . . . 69
3.4 Probabilistic Term Rewriting . . . . . 79
3.4.1 Syntax and Semantics . . . . . 80

3.4.2 Expected Complexity . . . 83
3.4.3 Evaluation Strategies . . . 83
3.4.4 PAST vs. SAST . . . 85
3.4.5 Proving Termination and Complexity via Interpretations . . . 88
3.5 Related Work & Open Problems . . . 101

**4 Relating Evaluation Strategies 105**
4.1 Relating Evaluation Strategies for TRSs . . . 107
4.1.1 Termination . . . 107
4.1.2 Complexity . . . 109
4.2 Relating Evaluation Strategies for PTRSs . . . 110
4.2.1 From Innermost to Full Rewriting . . . 111
4.2.2 From Weak to Full Normalization . . . 122
4.2.3 From Leftmost-Innermost to Innermost Rewriting . . . 125
4.3 Improving Applicability . . . 127
4.4 Transformational Approaches . . . 130
4.4.1 From Derivational to Runtime Complexity . . . 130
4.4.2 From Outermost to Innermost Evaluation . . . 134
4.5 Evaluation . . . 137
4.6 Related Work & Open Problems . . . 140

**5 Structural Properties and Composability of PTRSs 143**
5.1 Sequential Execution . . . 146
5.2 Parallel Execution . . . 152
5.3 Non-Termination Witnesses . . . 163
5.4 Modularity . . . 166
5.4.1 Disjoint Unions . . . 167
5.4.2 Shared Constructor Unions . . . 176
5.4.3 Hierarchical Unions . . . 178
5.5 Signature Extensions . . . 181
5.6 Related Work & Open Problems . . . 184

**6 Decidable Subclasses 187**
6.1 Stochastic Grammars . . . 190
6.1.1 Context-Free Grammars . . . 191
6.1.2 Stochastic Context-Free Grammars . . . 192
6.2 Transformation to Stochastic Grammars . . . 197
6.3 Extending Decidability . . . 203
6.4 Evaluation . . . 207
6.5 Related Work & Open Problems . . . 208

**7 Proving Almost-Sure Termination 211**
7.1 The Dependency Pair Framework . . . 216
7.1.1 Dependency Graph Processor . . . 218
7.1.2 Usable Rules Processor . . . 219
7.1.3 Reduction Pair Processor . . . 220
7.2 Annotated Dependency Pairs . . . 220
7.2.1 Innermost Rewriting with ADPs . . . 223
7.2.2 Full Rewriting with ADPs . . . 225
7.2.3 Chains . . . 227

7.3 The ADP Framework for AST . . . 232
7.4 Structural Processors . . . 240
7.4.1 Dependency Graph Processor . . . 241
7.4.2 Usable Terms Processor . . . 248
7.4.3 Usable Rules Processor . . . 250
7.4.4 Probability Removal Processor . . . 253
7.4.5 Switching From Full to Innermost AST . . . 256
7.4.6 Switching Back to PTRSs . . . 257
7.5 Processors Based on Algebras . . . 259
7.5.1 Reduction Pair Processor . . . 259
7.5.2 Subterm Criterion . . . 281
7.6 Transformational Processors . . . 284
7.6.1 Rewriting Processor . . . 285
7.6.2 Instantiation Processor . . . 298
7.6.3 Forward Instantiation Processor . . . 304
7.6.4 Rule Overlap Instantiation Processor . . . 309
7.7 All Processors at One Glance . . . 314
7.8 Evaluation . . . 314
7.9 Related Work & Open Problems . . . 316

**8 Proving Strong Almost-Sure Termination 319**
8.1 The ADP Framework for SAST . . . 321
8.2 Structural Processors . . . 329
8.2.1 Usable Rules Processor . . . 330
8.2.2 Dependency Graph Processor . . . 332
8.2.3 Knowledge Propagation Processor . . . 336
8.3 Processors Based on Algebras . . . 338
8.3.1 Reduction Pair Processor . . . 338
8.4 Transformational Processors . . . 343
8.5 All Processors at One Glance . . . 347
8.6 Evaluation . . . 348
8.7 Related Work & Open Problems . . . 349

**9 Disproving Termination 351**
9.1 Disproving AST and PAST via Loops and Embeddings . . . 352
9.2 Embedding Random Walks Based on Term Occurrences . . . 357
9.2.1 Counting Linear Term Occurrences . . . 359
9.2.2 Counting in Non-Variable-Decreasing RTs . . . 362
9.2.3 Counting Orthogonal Term Occurrences in RTs . . . 364
9.3 Embedding Random Walks Based on Pattern Terms . . . 366
9.3.1 Counting Orthogonal Pattern Occurrences . . . 371
9.3.2 Counting Linear Pattern Term Occurrences . . . 373
9.3.3 Counting in Non-Variable-Decreasing Pattern Trees . . . 377
9.4 Evaluation . . . 380
9.5 Related Work & Open Problems . . . 381

**10 Automatic Analysis of PTRSs in AProVE 383**
10.1 Capabilities and Input Formats . . . 384
10.2 Running AProVE . . . 386
10.3 The Modular Architecture of AProVE . . . 387

10.4 Strategies and Processors for `AST`, `SASST`, and Expected Complexity . 388
10.5 Evaluation . . . . . 390
10.6 Related Work . . . . . 392

**11 Conclusion** **395**
11.1 Summary of the Contributions . . . . . 395
11.2 Impact on the State-of-the-Art . . . . . 397
11.3 Outlook and Future Work . . . . . 398

**Bibliography** **399**

# 1 Introduction

Programming languages come in many different forms. They all have different purposes, each with its own rationale and motivation. There exist programming languages specifically designed for embedded systems or operating systems like C [KR88], languages for web development like JavaScript [ECM26], scripting languages for automating tasks like bash [RF25], or even languages designed to be unique, comedic, and artistic, like Shakespeare [HÅ01] or Piet [Mor18]. The latter two are called esoteric programming languages [Eso]. Irrespective of their syntax, an important property of programming languages is Turing completeness, i.e., the ability to simulate any Turing machine, which is the standard mathematical model of computation. Every Turing-complete language whose semantics is computable can be simulated by every other Turing-complete language. This means that, e.g., a Java [GJS+25] program can be fully automatically transformed into an equivalent C program.[1]

No matter what programming language was used to produce some software product, the two most important requirements are always the *correctness* and *efficiency* of the software. Nowadays, the software engineers behind the product do not write all of its code themselves, but substantial fragments of software are generated automatically, e.g., through program synthesis, AI-assisted development, or template-based code generators. As a consequence, software engineers may not always fully comprehend what the code in front of them is doing. Hence, formal verification is more pressing than ever, and this verification should ideally be done automatically by the computer itself in the form of mathematically sound proofs instead of simple tests or statistical guesses.

Program verification can be done in several forms. The three most influential ones today are *test-based verification*, *machine-learning-based (or AI-based) verification*, and *mathematically sound formal verification*. Test-based approaches provide correctness and efficiency results in the form of empirical evaluation. Correctness and efficiency can, however, only be guaranteed for the observed test cases. AI-based approaches provide no formal guarantees at all but are able to detect statistically probable errors in code. Both of these approaches are only able to *detect bugs* in software, but are unable to prove the *absence of bugs*. Formal verification solves these issues and provides mathematically sound reasoning for the correctness and efficiency of software. This

[1] Strictly speaking, Turing completeness of C only guarantees that an equivalent C program *exists*, provided that the semantics of Java is computable. For the transformation to be computable, it suffices that Java has a universal program (an interpreter) and that C is an *acceptable programming system* [Rog87], i.e., it additionally satisfies the $s_n^m$ theorem, which allows for computable partial application at the level of programs.

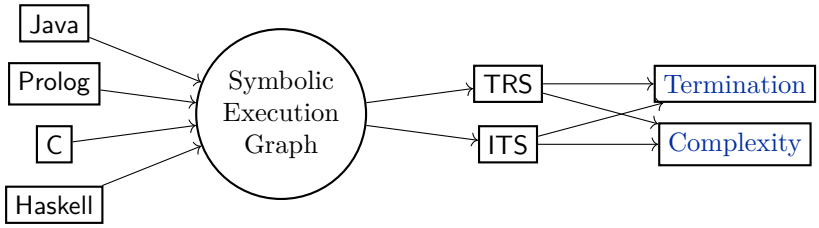



Figure 1.1: A graphical representation of how to analyze complex programming languages based on the analysis of simple programming languages.

should not be mistaken for a form of "supremacy of formal verification" compared to other approaches. There are important caveats for formal verification as well. While formal verification provides stronger guarantees, it is also drastically harder to perform. In fact, by RICE's theorem [Ric53], every non-trivial semantic property of programs in a TURING-complete language is undecidable, so there exists no algorithm that decides the correctness of arbitrary software. Since each approach has its own strengths and costs, combining them for different parts of the software is recommended in practice.

For many programs, the most basic requirement is that they provide a result at the *end of the computation*. However, many programs contain bugs, and sometimes the computation has no end, so that the user waits infinitely long. Therefore, it is crucial to ensure *termination* of the program. Additionally, the program should not only terminate eventually, but also within a reasonable amount of time. Waiting more than a million years for the result of some computation is not feasible for most human beings. Consequently, we not only try to formally prove termination, but we also try to find an upper bound on the runtime of a program. Moreover, the whole analysis should be performed by the machine itself, i.e., everything should be done completely automatically.

SECTION 1.1

## Automatic Formal Termination and Runtime Analysis

Formal termination and runtime analysis of software differs depending on which programming formalism one analyzes. When analyzing C one has to deal with pointers, and when analyzing functional programming languages like Haskell [Pey03] one has to deal with specific evaluation strategies like lazy evaluation. In this thesis, we focus on *term rewriting*, which is a very basic functional programming language based on pattern matching. A term rewrite system (TRS) consists of a set of rewrite rules $\ell \to r$ meaning that subterms of a term $t$ that are instances of $\ell$ can be replaced by the corresponding instance of the right-hand side $r$. Term rewriting has many applications, e.g., in equational reasoning, in computer algebra systems, in the study of computability, and in the implementation of functional programming languages [BN98, BKd03]. More importantly for this thesis, term rewriting can be used as a backend language to analyze other TURING-complete programming languages. Due to its simple definition, it is easier to analyze term rewriting than Java programs or C programs directly. The idea is to put all efforts into developing more powerful analysis techniques for the backend languages and only develop a single transformation from each frontend language (Java, C, Haskell, etc.) to a backend language like TRSs or *integer transition systems* (ITSs), see Figure 1.1. Such transformations for termination analysis should not only be *sound*,

but ideally also *complete*, *practical*, *efficient*, and *explainable*. To illustrate the different properties, assume that we want to transform a Java program $\mathcal{P}$ into a TRS $\mathcal{R}$ to analyze termination of $\mathcal{P}$.

**Sound**: If the TRS $\mathcal{R}$ terminates, then the Java program $\mathcal{P}$ terminates as well. This is the most important property as the main goal is to prove termination of the original Java program.

**Complete**: If the TRS $\mathcal{R}$ does not terminate, then the Java program $\mathcal{P}$ does not terminate either. While soundness ensures that proving termination of $\mathcal{R}$ yields a termination proof of $\mathcal{P}$, completeness ensures that the same relation holds for proving non-termination as well. Finding infinite executions in Java programs helps to detect and fix bugs in software.

**Practical**: If the Java program $\mathcal{P}$ is "simple to analyze", then so is the TRS $\mathcal{R}$. For example, if the Java program $\mathcal{P}$ only consists of a single loop over a single counter variable, then $\mathcal{R}$ should not exhibit overly complex recursion, but should be simple to analyze as well.

**Efficient**: The transformation can be computed in polynomial time, so in particular, the size of the resulting TRS $\mathcal{R}$ is polynomial in the size of the Java program $\mathcal{P}$. This is needed in practice to prevent an exponential blow-up of the time needed for the subsequent termination analysis.

**Explainable**: Whenever we have a termination or non-termination proof for $\mathcal{R}$, then we can use that proof to construct a termination proof or non-termination proof for the Java program $\mathcal{P}$, respectively. So explainability is a constructive strengthening of soundness and completeness that translates the witnesses. For example, from a finite representation of an infinite execution in $\mathcal{R}$ (e.g., a loop), we should be able to compute a corresponding infinite execution in $\mathcal{P}$.

Regarding complexity analysis, one can be interested in lower runtime bounds or upper runtime bounds. If we focus on the latter, then such transformations should be *upper bound sound*, *practical*, *complexity preserving*, *efficient*, and *explainable*.

**Upper Bound Sound**: The runtime complexity of the TRS $\mathcal{R}$ is (asymptotically) an upper bound on the runtime complexity of the Java program $\mathcal{P}$, where the sizes of the inputs of $\mathcal{R}$ and $\mathcal{P}$ have to be related suitably.

**Complexity Preserving**: The runtime complexity of $\mathcal{R}$ should not be "drastically", e.g., exponentially, higher than that of the Java program $\mathcal{P}$.

If we focus on lower bounds, then the two properties need to be adjusted accordingly.

Term rewriting is especially practical when it comes to analyzing algorithms dealing with data structures that can be modeled by first-order terms, e.g., lists or trees. A recurring example of a program that deals with lists and whose termination is based on a size decrease of the considered lists is the well-known quicksort algorithm.

Transformations from frontend languages to a backend language like TRSs or ITSs exist for, e.g., Java [BOvG10, BOG11, OBvG10, BMOG12, FG17b], C [SGB+14, HGFS16, SGB+17, HGFS18, HG23], functional programs like Haskell [GSST06, GRS+11], and logic programs like Prolog [SGST07, SGST09].

```
x ← 0;
while !(list.isEmpty()) do
    x ← x + 1;
    list ← list.getNext();
return x;
```

Figure 1.2: Imperative algorithm to compute the length of a given list.

$\mathcal{R}_{\mathsf{len}}$:

$$\mathsf{len}(\mathsf{nil}) \to \mathsf{0} \tag{1.1}$$
$$\mathsf{len}(\mathsf{cons}(x, xs)) \to \mathsf{s}(\mathsf{len}(xs)) \tag{1.2}$$

Figure 1.3: Term rewrite system to compute the length of a given list.

SECTION 1.2

## Term Rewriting

Since many readers of this thesis are probably familiar with at least one of the programming languages Java or C, or even just with imperative programming languages in general, but might not be familiar with the notion of a *term rewrite system*, we shall briefly illustrate the main ideas for the programming language we want to analyze.

In Figure 1.2, one can see an imperative program to compute the length of a given list. Lists are implemented as a data structure with two operations, namely an isEmpty()-function that checks whether the list is empty, and a getNext()-function that returns the list without its head element. We utilize a while-loop to iterate over the whole list and every time we call getNext() we increase the value of the integer variable $x$ by 1. Once the list is empty, we stop the loop iterations and return the current value of $x$. In Figure 1.3, one can see a corresponding term rewrite system $\mathcal{R}_{\mathsf{len}}$ for computing the length of a list. Term rewrite systems model a functional programming language. Instead of control-flow constructs like loops or if-conditions, the language purely relies on recursion and pattern matching to decide which rules to apply. We can easily represent recursive data structures as terms, e.g., lists are represented via nil (for the empty list) and cons (for a non-empty list), where, e.g., the term $\mathsf{cons}(\mathsf{s}(\mathsf{0}), \mathsf{cons}(\mathsf{s}(\mathsf{0}), \mathsf{cons}(\mathsf{0}, \mathsf{nil})))$ represents the list $[1, 1, 0]$. Note that natural numbers are represented by terms as well and there is no built-in data structure for numbers. Here, natural numbers are represented via the constructors 0 (for zero) and s (for the successor of a number).

A program state of the imperative program is a pair $(\ell, \sigma)$ of the current location $\ell$ within the program and an assignment $\sigma$ of the variables like "$x$" and "list" to concrete values. In term rewriting, a program state is a simple term like $\mathsf{len}(\mathsf{cons}(\mathsf{0}, \mathsf{nil}))$ representing the list $[0]$ inside the only argument of the function len. To evaluate the length function on the list $[0]$ we iteratively apply the rewrite rules of $\mathcal{R}_{\mathsf{len}}$. In the following, we first apply rule (1.2) in the first rewrite step, and then apply rule (1.1) in the second one. For readability, we often highlight the root symbol of the redex that is rewritten next in such rewrite sequences by coloring and underlining it.

$$\mathsf{len}([0]) = \underline{\mathsf{len}}(\mathsf{cons}(\mathsf{0}, \mathsf{nil})) \to_{\mathcal{R}_{\mathsf{len}}} \mathsf{s}(\underline{\mathsf{len}}(\mathsf{nil})) \to_{\mathcal{R}_{\mathsf{len}}} \mathsf{s}(\mathsf{0}) = 1.$$

SECTION 1.3

## Probabilistic Programs and Formalisms

While there are decades of extensive research on termination analysis and runtime analysis for ordinary programs, the automatic analysis of *probabilistic programs* is comparatively recent. Probabilistic programming has two distinct features: *probabilistic choices* and *conditioning*. In this work, as is common in parts of the literature, we focus on the probabilistic choices and do not consider any conditioning constructs. Hence, we consider programs that may flip a coin during execution and depending on the outcome of the coin flip can proceed in two different ways. Probabilistic programming has become actively studied due to its successful usage within machine learning, Bayesian inference, and to deal with uncertainty in data [GHNR14].

One can classify probabilistic algorithms into several types [MR95, BB88]. *Monte Carlo algorithms* always return a result but may be incorrect with small probability. *Las Vegas algorithms* always produce a correct result, but their runtime is random: they terminate with probability 1, even though some runs may not terminate at all. Finally, *Sherwood algorithms* always return a correct result and terminate, while using randomization to improve worst-case efficiency. All these algorithms can improve the runtime behavior of deterministic algorithms.

A well-known example illustrating the idea of a Sherwood algorithm is the randomized quicksort algorithm. If one uses a version where every list element has the same probability of becoming the next pivot element, then the algorithm performs $\mathcal{O}(n \cdot \log(n))$ comparisons in expectation, for every possible input list of length $n$ with pairwise distinct elements. In contrast, for a non-probabilistic version of quicksort that chooses its pivots at fixed positions, for every $n$ there exists a "worst-case" input of length $n$ on which the algorithm needs $n \cdot (n-1)/2$ comparisons, i.e., its worst-case number of comparisons is in $\Omega(n^2)$.

Several probabilistic programming languages have been developed. An example of an imperative probabilistic programming language is the probabilistic guarded command language (pGCL) [MM05], which builds on Kozen's semantics of probabilistic programs [Koz85]. An example of a program written in pGCL is shown in Algorithm 1 on the right. Here, the variable $y \in \mathbb{N}$ is the input of the program, $\oplus_p$ denotes probabilistic choice, where the left program is executed with probability $p$ and the right one with probability $1-p$, and $\square$ denotes demonic non-determinism.[2] In every iteration of the first loop, the probability to leave the first loop and enter the second one is at least $1/2$, and the second loop in turn terminates after a finite number of steps. Hence, the termination probability for Algorithm 1 is at least $1/2 + (1/2)^2 + (1/2)^3 + \cdots = \sum_{i=1}^{\infty} (1/2)^i = 1$. Note that while Algorithm 1 terminates with probability 1 (we say that it is *almost surely terminating*), its expected runtime is infinite for every initial value $y \geq 1$ and every resolution of the non-determinism. So it is not even *positively almost surely terminating*, which

**Algorithm 1:** Program in pGCL

```
x ← 0;
while x = 0 do
    {
      x ← 0 ⊕_{1/2} x ← 1;
      y ← 2 · y;
    }□{
      x ← 0 ⊕_{1/3} x ← 1;
      y ← 3 · y;
    }
while y > 0 do
    y ← y − 1;
```

[2] We often use the color blue to highlight probabilities in algorithms, text, and complex proofs.

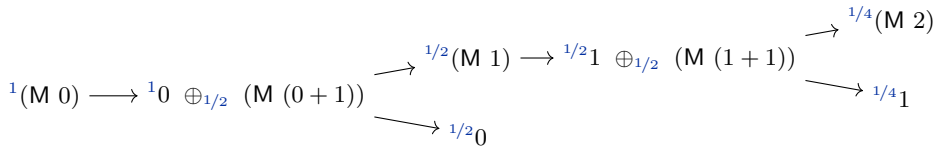



Figure 1.4: Illustration of a possible evaluation starting with the $\lambda$-term $\mathsf{M}\ 0$, where the evaluation of the arithmetic expression $r + 1$ is combined with the probabilistic choice into a single step.

would require a finite expected runtime for every resolution of the non-determinism, let alone *strongly almost surely terminating*, which additionally requires a bound on these expected runtimes that holds uniformly for all resolutions of the non-determinism. This already holds for the program where only the first possibility of the first $\mathsf{while}$-loop is considered (i.e., where $y$ is always doubled in its body). Then for the initial value $y = 1$, the expected number of iterations of the second $\mathsf{while}$-loop, which decrements $y$, is $1/2 \cdot 2 + 1/4 \cdot 4 + 1/8 \cdot 8 + \cdots = 1 + 1 + 1 + \cdots = \infty$. While the non-deterministic choice does not affect whether Algorithm 1 is almost surely terminating or has finite expected runtime, in general it can indeed influence termination and runtime.

A closely related probabilistic programming formalism that is used as a backend language for termination and runtime analysis is that of *probabilistic integer transition systems* [MHG21]. Compared to TRSs, such imperative programming languages on numbers are more practical for termination analysis of programs mainly dealing with arithmetic expressions. Such programs can also be expressed via probabilistic term rewriting, e.g., by encoding every probabilistic transition as a probabilistic rewrite rule, and thus, their analysis via term rewriting is possible in principle. However, as soon as the termination argument is based on number-theoretic justifications, the existing automation for TRSs is less powerful than dedicated techniques for such integer programs.

Another programming formalism is the $\lambda$-calculus, which is the backbone of any higher-order functional programming language. Instead of control structures and locations, the $\lambda$-*calculus* only consists of terms and values, similar to term rewriting as explained above. The main difference is that the $\lambda$-calculus can express *higher-order* terms and the (pure) $\lambda$-calculus uses operational semantics without pattern matching. The relation between the $\lambda$-calculus and constructor term rewrite systems is studied in [DLM12].

In addition to *applications* $(t_1\, t_2)$ and $\lambda$-*abstractions* $(\lambda x.t)$, *let-expressions* $(\mathsf{let}\ x = t_1\ \mathsf{in}\ t_2)$, and *if-expressions* $(\mathsf{if}\ t_1\ \mathsf{then}\ t_2\ \mathsf{else}\ t_3)$, the probabilistic $\lambda$-calculus has an additional *coin-flip operator* $\oplus_{1/2}$ exactly like the one used in pGCL. A term $(\mathsf{M}\ r)$ that is able to rewrite to $r \oplus_{1/2} (\mathsf{M}\ (r+1))$ describes a geometric distribution [DL20], see Figure 1.4. *Type systems*, given by *typing rules*, can guarantee additional properties of well-typed terms. For example, there exists a type-theoretical characterization, based on *intersection types*, of the terms that terminate with probability 1 or have finite expected runtime [DLFDR21]. However, such complete characterizations cannot be fully automated, since almost-sure termination is undecidable (it is even $\Pi^0_2$-complete [KKM19]). The same limitation applies to probabilistic term rewriting, so every fully automatic technique, including the ones developed in this thesis, is necessarily incomplete.

Most of the probabilistic programming languages used in the literature to analyze programs, like the basic form of pGCL, probabilistic integer transition systems, or probabilistic adaptations of the $\lambda$-calculus, do not provide user-defined first-order data structures with pattern matching as primitives. In contrast, TRSs are especially suitable for modeling and analyzing algorithms like quicksort that operate on data structures like lists. They were extended to *probabilistic term rewrite systems* (PTRSs) in [BK02] and further studied in, e.g., [BG05, BG06, Gna07, ADLY20], where the inherent non-deterministic selection of a redex and a rule in TRSs is extended by an additional probabilistic choice afterwards. So the selection of the redex and the rule remains non-deterministic, but after this selection one has multiple probabilistic choices for the reduct. However, instead of a binary probabilistic choice operator $\oplus_{1/2}$, PTRSs allow arbitrary finite distributions with constant real-valued probabilities. For example, instead of $\mathsf{a} \oplus_{1/2} \mathsf{b}$ we directly use the finite distribution $\{^{1/2}\mathsf{a}, {}^{1/2}\mathsf{b}\}$. As shown in [MM05], for pGCL a single constant probabilistic choice operator like $\oplus_{1/2}$ suffices to express every computable distribution over $\mathbb{N}$. A similar result was obtained in [DLZG14], where it was shown that the same class of computable distributions over $\mathbb{N}$ is also characterizable via probabilistic recursive functions. An analogous result holds for PTRSs, which is the reason why we restrict ourselves to such finite distributions on the right-hand sides. To see this, note that PTRSs whose right-hand sides only have the form $\{^{1/2}r_1, {}^{1/2}r_2\}$ or $\{^{1}r\}$ can simulate probabilistic Turing machines, where we use the standard encoding of Turing machines by term rewrite systems, where the symbol for the current state marks the unique redex of the term. Every probabilistic transition of the machine becomes a single rule with two outcomes of probability $1/2$ each. Since every reachable term contains exactly one redex, no non-determinism arises, and thus, the result of [DLZG14] carries over to PTRSs.

The following PTRS $\mathcal{P}_{\mathsf{alg}}$ corresponds to an implementation of Algorithm 1.

$$\mathsf{loop1}(y) \to \{^{1/2}\mathsf{loop1}(\mathsf{double}(y)),\ {}^{1/2}\mathsf{loop2}(\mathsf{double}(y))\} \tag{1.3}$$

$$\mathsf{loop1}(y) \to \{^{1/3}\mathsf{loop1}(\mathsf{triple}(y)),\ {}^{2/3}\mathsf{loop2}(\mathsf{triple}(y))\} \tag{1.4}$$

$$\mathsf{loop2}(\mathsf{s}(y)) \to \{^{1}\mathsf{loop2}(y)\} \qquad \mathsf{triple}(\mathsf{s}(y)) \to \{^{1}\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{triple}(y))))\}$$

$$\mathsf{double}(\mathsf{s}(y)) \to \{^{1}\mathsf{s}(\mathsf{s}(\mathsf{double}(y)))\} \qquad \mathsf{triple}(0) \to \{^{1}0\}$$

$$\mathsf{double}(0) \to \{^{1}0\}$$

Here, natural numbers are represented via the constructors 0 and s, and the non-determinism is modeled by the non-deterministic choice between the two rules (1.3) and (1.4). We show in Chapter 3 how to execute a PTRS like $\mathcal{P}_{\mathsf{alg}}$. Moreover, we show how to automatically analyze almost-sure termination of this algorithm in Chapter 7.

The PTRS $\mathcal{P}_{\mathsf{geo}}$, which consists only of the rule $\mathsf{geo}(x) \to \{^{1/2}\mathsf{geo}(\mathsf{s}(x)),\ {}^{1/2}x\}$, expresses the geometric distribution like the $\lambda$-term $(\mathsf{M}\ r)$. Again, we show how to automatically analyze almost-sure termination and even how to automatically infer upper runtime bounds for this PTRS in Chapter 7 and Chapter 8, respectively.

It is again important to mention that probabilistic term rewriting is not meant to be a superior probabilistic programming formalism. In comparison, pGCL can easily express complex computations on numbers, and the probabilistic $\lambda$-calculus can easily express higher-order functions. Probabilistic term rewriting should be understood as one of many probabilistic programming languages important for formal verification.

# 2 Probability Theory

In this chapter, we introduce all necessary notions and notation from probability theory that one should be familiar with when dealing with stochastic systems. All definitions are accompanied by simple examples, e.g., the well-known stochastic experiment of throwing a single die or multiple dice.[1] Additionally, we always explain the relation to probabilistic programs. In the end, we introduce several fundamental stochastic systems, namely *random walks*, *random walk programs*, *Markov chains*, and *Markov decision processes*. Probabilistic term rewriting, the formalism we analyze in this thesis and introduce in Chapter 3, can directly express Markov chains and Markov decision processes with finitely many states as well as random walk programs with finite support. There are three main reasons why we introduce all these different systems.

1. First, we use these systems to highlight the difficulties that must be taken into account when defining the *probability of termination* and the *expected runtime* of stochastic systems, and moreover, to highlight the difficulties when analyzing such properties of stochastic systems.

2. Second, we use these systems as illustrative instances of probabilistic rewrite systems throughout this whole thesis.

3. Lastly, we use these systems within our analysis of probabilistic term rewriting. For example, we use random walk programs to disprove almost-sure termination and positive almost-sure termination of probabilistic rewrite systems in Chapter 9.

## Structure

We first recapitulate the notions of $\sigma$-fields, measurable spaces, measurable maps, and probability spaces in Section 2.1. Then, in Section 2.2 we recapitulate random variables and their expected values. Afterwards, we introduce four different well-known stochastic systems. In Section 2.3 we introduce random walks and random walk programs. These can be classified based on the (expected) change after a single step into four different types, each with a different termination behavior. Finally, we recapitulate Markov chains and Markov decision processes in Section 2.4. Here, termination analysis boils down to finding the least fixpoint of a set of equations.

[1] A "die" is singular, while "dice" is plural.

SECTION 2.1

## Probability Spaces

We start with a rigorous definition of probability spaces and their axiomatization. Although we can abstract away from most of the formal details in large parts of this work, we still need them to understand the core concepts behind probabilistic programs, which often seem unintuitive at first glance. As we see in this section, without this attention to detail and formality one may encounter inconsistencies in an argument rather quickly without directly noticing them.

To ease readability, we use colors to highlight probabilities. While all colored objects denote probabilities, we may omit the color for probabilities if it would give the false impression that the result of an expression is a probability as well. For example, the expected value requires probabilities in its calculation but does not yield a probability.

A probability space provides the mathematical model of a *stochastic experiment*, which has a fixed set of possible outcomes $\Omega$, called the *sample space*. While it might be natural to assume that a probability space gives a certain probability $\mathbb{P}(\omega)$ to each possible *outcome* $\omega \in \Omega$ and to assume that the probability for the whole sample space is 1, i.e., $\sum_{\omega\in\Omega} \mathbb{P}(\omega) = 1$, this is not always consistent or feasible. If the sample space $\Omega$ is uncountably infinite, then defining an "intuitive" probability space by giving uncountably many elements a probability $p > 0$ is infeasible in general.

> Remark 2.1.1 – Different Types of Infinity.
>
> Given a finite set $A$, we can *count* how many elements it contains to give a notion of "size" to the set $A$, denoted by $|A|$. For example, we have $|\{1, 2, 3, 4\}| = 4$. If the set $A$ is infinite, it becomes more difficult to track its "size". In this thesis, only two types of infinite sets are important: *countably infinite sets* and *uncountably infinite sets*. A set $A$ is *countably infinite* if there exists a bijection $f : \mathbb{N} \to A$. For example, the set $\mathbb{Z}$ is countably infinite due to the bijection $f_{\mathbb{N}\to\mathbb{Z}} : \mathbb{N} \to \mathbb{Z}$ with $x \mapsto \frac{x}{2}$ if $x$ is even, and $x \mapsto -\frac{x+1}{2}$ if $x$ is odd. A set $A$ is *uncountably infinite* if it is infinite and not countably infinite. For example, the set $\mathbb{R}$ of real numbers or just the unit interval $[0, 1] \subseteq \mathbb{R}$ is uncountably infinite. The number of possible executions in most probabilistic programs is uncountably infinite as well.

Counterexample 2.1.2 (Problem of an Uncountable Sample Space, [Tao15]). Suppose we want to draw a real number uniformly at random from the unit interval $[0, 1]$. As mentioned above, we want to have $\sum_{x\in[0,1]} \mathbb{P}(x) = 1$. One would expect that the probability to draw a number in any interval $[a, b] = \{y \in \mathbb{R} \mid a \leq y \leq b\}$ with $0 \leq a \leq b \leq 1$ is $b - a$. But then, we would have $\mathbb{P}(x) = 0$ for every $x \in [0, 1]$, because $x$ is the only number in the interval $[x, x]$, which has probability $x - x = 0$. We get $\sum_{x\in[0,1]} \mathbb{P}(x) = 0$, which is a contradiction. On the other hand, if we have $\mathbb{P}(x) > 0$ for every $x \in [0, 1]$, then $\sum_{x\in[0,1]} \mathbb{P}(x) = \infty \neq 1$.

To avoid the difficulties shown in Counterexample 2.1.2, we now present a definition of probability spaces based on measure theory and the axioms of Kolmogorov. Instead of giving each element its own probability, we define a family $\mathfrak{A}$ of subsets of $\Omega$, called the *event space*, where we can assign a probability to each member. Each element $A \in \mathfrak{A}$ is called an *event*. In order to align with the intuitive meaning of probability, this family has to adhere to certain rules. For any sample space $\Omega$, a family $\mathfrak{A}$ of subsets of $\Omega$ is called a *$\sigma$-field* if it contains the full set $\Omega$ and is closed both under

complement and under countable unions. By $\mathrm{Pow}(\Omega) = \{X \mid X \subseteq \Omega\}$ we denote the *power set* of any set $\Omega$.

Definition 2.1.3 – $\sigma$-Field.
*Let $\Omega$ be a non-empty set and $\mathfrak{A} \subseteq \mathrm{Pow}(\Omega)$. $\mathfrak{A}$ is called a $\sigma$-field over $\Omega$ if*

$$1.\ \Omega \in \mathfrak{A}, \quad 2.\ A \in \mathfrak{A} \implies \Omega \setminus A \in \mathfrak{A}, \text{ and } \quad 3.\ A_i \in \mathfrak{A} \text{ for all } i \in \mathbb{N} \implies \bigcup_{i \in \mathbb{N}} A_i \in \mathfrak{A}.$$

*The pair $(\Omega, \mathfrak{A})$ is called a measurable space, where $\Omega$ is the sample space and $\mathfrak{A}$ is the event space. The elements of $\mathfrak{A}$ are called measurable sets or simply events.*

Example 2.1.4 (Finest and Coarsest $\sigma$-Field). $\mathrm{Pow}(\Omega)$ and $\{\varnothing, \Omega\}$ are always $\sigma$-fields over any non-empty set $\Omega$.

Example 2.1.5. As a running example for this chapter, let $\Omega_{\mathsf{dice}} = \{1, 2, 3, 4, 5, 6\}$, i.e., the sample space of all possible outcomes of throwing a single die. Then, $\mathfrak{A}_{\mathrm{eo}} = \{\Omega_{\mathsf{dice}}, \varnothing, \{1, 3, 5\}, \{2, 4, 6\}\}$ which contains, in addition to $\varnothing$ and $\Omega_{\mathsf{dice}}$, the set of even numbers $\{2, 4, 6\}$ and the set of odd numbers $\{1, 3, 5\}$, is a possible $\sigma$-field over $\Omega_{\mathsf{dice}}$.

A *probability space* $(\Omega, \mathfrak{A}, \mathbb{P})$ extends a measurable space $(\Omega, \mathfrak{A})$ by a *probability measure* $\mathbb{P}$ which maps every event from $\mathfrak{A}$ to a probability between 0 and 1 such that $\mathbb{P}(\Omega) = 1$ (the probability that the outcome of our experiment lies within $\Omega$ is 1), $\mathbb{P}(\varnothing) = 0$ (the probability that the outcome of our experiment lies within $\varnothing$ is 0), and $\mathbb{P}$ is $\sigma$-additive, i.e., $\mathbb{P}(\biguplus_{i \in \mathbb{N}} A_i) = \sum_{i \in \mathbb{N}} \mathbb{P}(A_i)$ for every countable family of pairwise disjoint events $\{A_i \mid i \in \mathbb{N}\} \subseteq \mathfrak{A}$.

Definition 2.1.6 – Probability Space.
*Let $(\Omega, \mathfrak{A})$ be a measurable space. A map $\mathbb{P} : \mathfrak{A} \to \mathbb{R}_{\geq 0}$ is a probability measure if*

$$1.\ \mathbb{P}(\Omega) = 1, \qquad 2.\ \mathbb{P}(\varnothing) = 0, \text{ and}$$

3. *for every countable family of pairwise disjoint sets $\{A_i \mid i \in \mathbb{N}\} \subseteq \mathfrak{A}$ we have* $\mathbb{P}(\biguplus_{i \in \mathbb{N}} A_i) = \sum_{i \in \mathbb{N}} \mathbb{P}(A_i)$ ($\sigma$-additive).

*The triple $(\Omega, \mathfrak{A}, \mathbb{P})$ is called a probability space.*

The map $\mathbb{P}$ gives each event $A \in \mathfrak{A}$ a corresponding probability $\mathbb{P}(A) \in [0, 1]$. Intuitively, $\mathbb{P}(A)$ is the probability that the stochastic experiment yields an outcome $\omega \in \Omega$ that is contained in the event $A \in \mathfrak{A}$. If $\Omega$ is finite and all elements of $\Omega$ have the same probability of being chosen, then the resulting probability space is a *Laplace space*.

Definition 2.1.7 – Laplace Space.
*Let $\Omega$ be a non-empty finite set. The probability space $(\Omega, \mathrm{Pow}(\Omega), \mathbb{P})$ where we have $\mathbb{P} : \mathrm{Pow}(\Omega) \to [0, 1], A \mapsto \frac{|A|}{|\Omega|}$, is called the Laplace space over $\Omega$.*

Example 2.1.8 (Laplace Space). Let $\mathbb{P} : \mathrm{Pow}(\Omega_{\mathsf{dice}}) \to [0, 1], A \mapsto \frac{|A|}{|\Omega_{\mathsf{dice}}|} = \frac{|A|}{6}$. Then $(\Omega_{\mathsf{dice}}, \mathrm{Pow}(\Omega_{\mathsf{dice}}), \mathbb{P})$ is the Laplace space over $\Omega_{\mathsf{dice}}$. This space can be seen as the model of throwing a fair die once. For every $A \subseteq \Omega_{\mathsf{dice}}$, $\mathbb{P}(A)$ is the probability that throwing the die yields a number in $A$.

In probabilistic program verification, $\Omega$ would be the set of all possible program runs. A *program run* is an infinite sequence of configurations, where a configuration is, e.g., a pair consisting of a location and a variable assignment for imperative programs. If one considers probabilistic term rewriting instead, then a configuration is a simple term and $\Omega$ contains all possible rewrite sequences. Since $\Omega$ is in general uncountably infinite, the most precise $\sigma$-field $\mathrm{Pow}(\Omega)$ has the problems mentioned in Counterexample 2.1.2. Instead, we define a $\sigma$-field based on all finite program runs, which we already have to do for the different stochastic systems that we introduce in Section 2.3 and Section 2.4. To do so, we first define how to generate $\sigma$-fields from a given base subset of events and already existing $\sigma$-fields.

Lemma 2.1.9

*Let $\mathfrak{A}$ and $\mathfrak{B}$ be $\sigma$-fields over $\Omega$. Then $\mathfrak{A} \cap \mathfrak{B}$ is a $\sigma$-field over $\Omega$. Furthermore, if $\{\mathfrak{A}_i \mid i \in I\}$ is a family of $\sigma$-fields over $\Omega$ with an arbitrary (possibly uncountable) non-empty index set $I$, then so is $\bigcap_{i \in I} \mathfrak{A}_i$.*

*Proof.* We only show the lemma for the intersection of two $\sigma$-fields. The generalization to arbitrary families is straightforward. First, we have $\Omega \in \mathfrak{A}$ and $\Omega \in \mathfrak{B}$, thus $\Omega \in \mathfrak{A} \cap \mathfrak{B}$. If we have $A \in \mathfrak{A} \cap \mathfrak{B}$, then $A \in \mathfrak{A}$ and $\Omega \setminus A \in \mathfrak{A}$, as $\mathfrak{A}$ is a $\sigma$-field. The same holds for $\mathfrak{B}$, and thus, $\Omega \setminus A \in \mathfrak{A} \cap \mathfrak{B}$. Finally, if we have $A_i \in \mathfrak{A} \cap \mathfrak{B}$ for all $i \in \mathbb{N}$, then again, $A_i \in \mathfrak{A}$ and $A_i \in \mathfrak{B}$ for all $i \in \mathbb{N}$, and thus, $\bigcup_{i \in \mathbb{N}} A_i \in \mathfrak{A}$ and $\bigcup_{i \in \mathbb{N}} A_i \in \mathfrak{B}$ by definition of a $\sigma$-field. This yields $\bigcup_{i \in \mathbb{N}} A_i \in \mathfrak{A} \cap \mathfrak{B}$, and shows that $\mathfrak{A} \cap \mathfrak{B}$ is indeed a $\sigma$-field. *Proof* *QED*

For any set $\mathfrak{E} \subseteq \mathrm{Pow}(\Omega)$, let $\langle \mathfrak{E} \rangle_\sigma \subseteq \mathrm{Pow}(\Omega)$ consist of all elements that are contained in all $\sigma$-fields that are supersets of $\mathfrak{E}$ (there is at least one such $\sigma$-field, namely $\mathrm{Pow}(\Omega)$). We call $\mathfrak{E}$ the *base* of $\langle \mathfrak{E} \rangle_\sigma$.

Definition 2.1.10 – Generating $\sigma$-Fields.

*Let $\mathfrak{E} \subseteq \mathrm{Pow}(\Omega)$. The coarsest $\sigma$-field over $\Omega$ containing the base $\mathfrak{E}$ is*

$$\langle \mathfrak{E} \rangle_\sigma = \bigcap_{\mathfrak{E} \subseteq \mathfrak{A},\ \mathfrak{A}\ \sigma\text{-field over } \Omega} \mathfrak{A}.$$

Due to Lemma 2.1.9, we know that $\langle \mathfrak{E} \rangle_\sigma$ is a valid $\sigma$-field for every $\mathfrak{E} \subseteq \mathrm{Pow}(\Omega)$.

Example 2.1.11 (Generating $\sigma$-Fields). Continuing Example 2.1.5, we have $\mathfrak{A}_{\text{eo}} = \langle \{\{1,3,5\}\} \rangle_\sigma = \langle \{\{2,4,6\}\} \rangle_\sigma$ as any $\sigma$-field over $\Omega_{\text{dice}}$ contains $\varnothing$ and $\Omega_{\text{dice}}$, and if it contains $\{1,3,5\}$ or $\{2,4,6\}$, it has to contain its complement as well. It turns out that this is already a $\sigma$-field over $\Omega_{\text{dice}}$, and therefore it is the coarsest one that contains $\{1,3,5\}$ or $\{2,4,6\}$.

An important sample space is $\Omega = \overline{\mathbb{R}}^n = (\mathbb{R} \cup \{-\infty, \infty\})^n$. We use subsets of $\overline{\mathbb{R}} = \overline{\mathbb{R}}^1$, e.g., $\overline{\mathbb{Z}} = \mathbb{Z} \cup \{-\infty, \infty\} \subseteq \overline{\mathbb{R}}$, when considering the probability of termination or the expected runtime of a probabilistic program. The most important $\sigma$-field for $\overline{\mathbb{R}}^n$ is the Borel-field, which is generated by all open subsets of $\overline{\mathbb{R}}^n$.

Definition 2.1.12 – Borel-Field.
*For $\Omega = \overline{\mathbb{R}}^n$ we use the $n$-dimensional Borel-field $\mathfrak{B}^n$ as its $\sigma$-field, where*

$$\mathfrak{B}^n = \langle \{U \subseteq \overline{\mathbb{R}}^n \mid U \text{ open}\} \rangle_\sigma.$$

By Lemma 2.1.9, $\mathfrak{B}^n$ is indeed a $\sigma$-field over $\overline{\mathbb{R}}^n$, and we write $\mathfrak{B} = \mathfrak{B}^1$. For any subset $X \subseteq \overline{\mathbb{R}}^n$ we can create the corresponding subspace $\sigma$-field by intersecting $X$ with measurable sets from $\mathfrak{B}^n$, i.e., $\mathfrak{B}_X = \{X \cap B \mid B \in \mathfrak{B}^n\}$ is a $\sigma$-field over $X$.

Example 2.1.13 (Borel-Field). Let us consider the real line $\mathbb{R}$ with the corresponding subspace $\sigma$-field $\mathfrak{B}_\mathbb{R} = \{\mathbb{R} \cap B \mid B \in \mathfrak{B}\}$. Then it can be shown that $\mathfrak{B}_\mathbb{R} = \langle \{(a, b) \mid a, b \in \mathbb{R}\} \rangle_\sigma$, i.e., $\mathfrak{B}_\mathbb{R}$ is the coarsest $\sigma$-field over $\mathbb{R}$ that contains all open intervals. So for example $(-3, 7) \in \mathfrak{B}_\mathbb{R}$ and $(-2, 1] = [0, 1] \cup (-2, 0) \in \mathfrak{B}_\mathbb{R}$.

For many mathematical structures there exists a notion of "structure-preserving maps" (or *morphisms*). A function that preserves the structure of a measurable space is called a *measurable map*.

Definition 2.1.14 – Measurable Map.
*Let $(\Omega_1, \mathfrak{A}_1)$ and $(\Omega_2, \mathfrak{A}_2)$ be measurable spaces. A function $f : \Omega_1 \to \Omega_2$ is called an $(\mathfrak{A}_1$–$\mathfrak{A}_2)$-measurable map or just measurable if for all $A \in \mathfrak{A}_2$ we have $f^{-1}(A) = \{\omega \in \Omega_1 \mid f(\omega) \in A\} \in \mathfrak{A}_1$.*
*The pullback $\langle f \rangle_\sigma = f^{-1}(\mathfrak{A}_2) = \{f^{-1}(A) \mid A \in \mathfrak{A}_2\}$ is a $\sigma$-field over $\Omega_1$ and it is the coarsest $\sigma$-field $\mathfrak{A}$ over $\Omega_1$ such that $f$ is $(\mathfrak{A}$–$\mathfrak{A}_2)$-measurable.*

Example 2.1.15 (Measurable Map). Let us again refer to Example 2.1.5 and consider the measurable spaces $(\Omega_{\mathsf{dice}}, \mathrm{Pow}(\Omega_{\mathsf{dice}}))$ and $(\Omega_{\mathsf{dice}}, \mathfrak{A}_{\mathrm{eo}})$. Let $\mathrm{id} : \Omega_{\mathsf{dice}} \to \Omega_{\mathsf{dice}}, i \mapsto i$ be the identity. Then id is $(\mathrm{Pow}(\Omega_{\mathsf{dice}})$–$\mathfrak{A}_{\mathrm{eo}})$-measurable, since for any $A \in \mathfrak{A}_{\mathrm{eo}}$, we have $\mathrm{id}^{-1}(A) = A \in \mathrm{Pow}(\Omega_{\mathsf{dice}})$. We obtain $\langle \mathrm{id} \rangle_\sigma = \mathrm{id}^{-1}(\mathfrak{A}_{\mathrm{eo}}) = \mathfrak{A}_{\mathrm{eo}}$.

Measurable maps are closed under composition, i.e., if $(\Omega_1, \mathfrak{A}_1)$, $(\Omega_2, \mathfrak{A}_2)$, and $(\Omega_3, \mathfrak{A}_3)$ are measurable spaces; $f_1 : \Omega_1 \to \Omega_2$ is an $(\mathfrak{A}_1$–$\mathfrak{A}_2)$-measurable map; and $f_2 : \Omega_2 \to \Omega_3$ is an $(\mathfrak{A}_2$–$\mathfrak{A}_3)$-measurable map, then $f_2 \circ f_1$ is $(\mathfrak{A}_1$–$\mathfrak{A}_3)$-measurable.

Finally, we end this section with an introduction to conditional probability. Assuming that we already know that the outcome of our stochastic experiment lies within an event $B \in \mathfrak{A}$, what is the probability that the outcome lies within another event $A \in \mathfrak{A}$ as well? This question is, however, only well-defined if we ensure that the first event $B$ can occur with a probability $\mathbb{P}(B) > 0$ in the first place. This question gives rise to a whole new probability space, called the *conditional probability space w.r.t. $B$*.

Definition 2.1.16 – Conditional Probability.
*Let $(\Omega, \mathfrak{A}, \mathbb{P})$ be a probability space and $B \in \mathfrak{A}$ such that $\mathbb{P}(B) > 0$. Then the map $\mathbb{P}(\bullet \mid B) : \mathfrak{A} \to \mathbb{R}_{\geq 0}, A \mapsto \mathbb{P}(A \mid B) = \frac{\mathbb{P}(A \cap B)}{\mathbb{P}(B)}$ defines a probability measure on $(\Omega, \mathfrak{A})$. It is called the conditional probability w.r.t. $B$.*

Example 2.1.17 (Conditional Probability). Reconsider the probability space of throwing a die from Example 2.1.8. If we set $B = \{2, 4, 6\}$, then $\mathbb{P}(\{1\} \mid B) = \frac{\mathbb{P}(\varnothing)}{\mathbb{P}(B)} = \frac{0}{1/2} = 0$ is the probability of throwing a 1 when having the information that we will throw an even number.

Additionally, we have $\mathbb{P}(\{2\} \mid B) = \frac{\mathbb{P}(\{2\})}{\mathbb{P}(B)} = \frac{1/6}{1/2} = 1/3$, i.e., the probability of throwing a 2 when having the information that we will throw an even number is $1/3$.

SECTION 2.2

## Random Variables

Usually, not each individual outcome $\omega \in \Omega$ is of interest, but rather a function value $X(\omega)$ for some function $X$ on $\Omega$, where the function value may be the same for different outcomes. A *random variable* $X$ is an $(\mathfrak{A}$–$\mathfrak{A}')$-measurable function $X : \Omega \to \Omega'$ from one measurable space $(\Omega, \mathfrak{A})$ to another $(\Omega', \mathfrak{A}')$. If $\mathbb{P}$ is a probability measure for $\Omega$, then one obtains a probability measure $\mathbb{P}^X$ for $\Omega'$. For $A' \in \mathfrak{A}'$, $\mathbb{P}^X(A')$ is the probability that an element chosen from $\Omega$ is mapped by $X$ to an element contained in $A'$. In other words, instead of regarding the probabilities for choosing elements from $A \subseteq \Omega$, one now regards the probabilities for the values of the random variable $X$.

Remark 2.2.1 – Domain, Codomain, Image, and Preimage.

We distinguish between the *domain*, *codomain*, and the *image* of a function. If $f : A \to B$ is an arbitrary function between two sets $A$ and $B$, then its *domain* is $\mathrm{dom}(f) = A$, its *codomain* is $\mathrm{cod}(f) = B$, and its *image* is the set $\mathrm{im}(f) = \{f(a) \mid a \in A\}$. The *preimage* $f^{-1}(B')$ of a set $B' \subseteq B$ is the set $f^{-1}(B') = \{a \mid a \in A, f(a) \in B'\}$. For example, if $f : \mathbb{R} \to \mathbb{R}, x \mapsto x^2$, then the domain and the codomain are $\mathbb{R}$, but its image is only $\mathbb{R}_{\geq 0}$. Moreover, we have $f^{-1}([1, \infty)) = (-\infty, -1] \cup [1, \infty)$.

Definition 2.2.2 – Random Variable.

*Let $(\Omega, \mathfrak{A}, \mathbb{P})$ be a probability space and $(\Omega', \mathfrak{A}')$ be a measurable space. An $(\mathfrak{A}$–$\mathfrak{A}')$-measurable map $X : \Omega \to \Omega'$ is called a random variable. We denote such a random variable as $X \sim (\Omega, \mathfrak{A}, \mathbb{P}) \to (\Omega', \mathfrak{A}')$. A random variable $X$ is discrete if its image $\mathrm{im}(X)$ is a countable set.*

*The probability measure defined by $\mathbb{P}^X : \mathfrak{A}' \to [0, 1], A \mapsto \mathbb{P}(X^{-1}(A))$ is called the probability measure induced by $X$ on $(\Omega', \mathfrak{A}')$ and $(\Omega', \mathfrak{A}', \mathbb{P}^X)$ is the induced probability space. We often use the notation $\mathbb{P}(X \in A) = \mathbb{P}^X(A)$, and if $A = \{i\}$ is a singleton set, we write $\mathbb{P}(X = i) = \mathbb{P}(X \in \{i\}) = \mathbb{P}^X(\{i\})$.*

*If the codomain is not stated explicitly, a random variable is assumed to take extended real values, i.e., $X : \Omega \to \overline{\mathbb{R}}$. In this case, we denote the random variable by $X \sim (\Omega, \mathfrak{A}, \mathbb{P})$, and instead of saying "$(\mathfrak{A}$–$\mathfrak{B})$-measurable" for the* Borel*-field $\mathfrak{B}$, we simply use the notation "$\mathfrak{A}$-measurable". Moreover, if $\mathrm{cod}(X) \subseteq \overline{\mathbb{R}}$, e.g., $\mathrm{cod}(X) = \overline{\mathbb{Z}}$, then we view $X$ as taking values in the measurable space $(\mathrm{cod}(X), \{\mathrm{cod}(X) \cap B \mid B \in \mathfrak{B}\})$.*

Example 2.2.3 (Delta Variable). Let $(\Omega, \mathfrak{A}, \mathbb{P})$ be a probability space and let $(\Omega', \mathfrak{A}')$ be a measurable space such that $\{x\} \in \mathfrak{A}'$ for all $x \in \Omega'$. For every $x \in \Omega'$, we can define a random variable $\mathbb{1}_x \sim (\Omega, \mathfrak{A}, \mathbb{P}) \to (\Omega', \mathfrak{A}')$ such that we have $\mathbb{P}(\mathbb{1}_x = x) = 1$ and $\mathbb{P}(\mathbb{1}_x = y) = 0$ for all $y \neq x$. We achieve this by mapping every outcome $\omega \in \Omega$ to $x$, i.e., $\mathbb{1}_x(\omega) = x$, which is measurable, since every preimage $\mathbb{1}_x^{-1}(A)$ is either $\Omega$ or $\varnothing$. The random variable $\mathbb{1}_x$ is called the *Dirac delta random variable* with value $x$, and its induced probability measure $\mathbb{P}^{\mathbb{1}_x}$ is the *Dirac measure* at $x$.

Example 2.2.4 (Throwing Multiple Dice). Let us consider throwing a fair die twice, where we are interested in the sum of both throws. So let $\Omega^2_{\mathsf{dice}} = \{1, 2, 3, 4, 5, 6\}^2$

and $(\Omega^2_{\mathsf{dice}}, \mathrm{Pow}(\Omega^2_{\mathsf{dice}}), \mathbb{P})$ be the LAPLACE space over $\Omega^2_{\mathsf{dice}}$. Then, $X : \Omega^2_{\mathsf{dice}} \to \mathbb{R}$ with $(i, j) \mapsto i + j$ is a random variable. The probability that the sum of the two throws is 5 is given by $\mathbb{P}(X = 5)$ and the probability that the sum of the two throws is contained in $\{2, 3, 4\}$ is given by $\mathbb{P}(X \in \{2, 3, 4\})$. We get

$$\mathbb{P}(X = 5) = \mathbb{P}^X(\{5\}) = \mathbb{P}(\{(1,4),(4,1),(2,3),(3,2)\}) = \tfrac{4}{36} = \tfrac{1}{9}, \text{ and}$$
$$\mathbb{P}(X \in \{2,3,4\}) = \mathbb{P}^X(\{2,3,4\}) = \mathbb{P}(\{(1,1),(1,2),(1,3),(2,1),(2,2),(3,1)\}) = \tfrac{6}{36} = \tfrac{1}{6}.$$

In the program verification setting, we regard random variables that map computations to the index of the first configuration in the run that corresponds to a terminating state. In this way, one can define the probability of termination for a probabilistic program. Note that this yields a discrete random variable, i.e., it only takes countably many possible values. In fact, we only encounter discrete random variables throughout this thesis when analyzing probabilistic programs.

If a random variable is discrete, then its *expected value* is a weighted sum.

Definition 2.2.5 – Expected Value.

*Let $X$ be a discrete random variable on a probability space $(\Omega, \mathfrak{A}, \mathbb{P})$. The expected value or expectation of $X$ is*

$$\mathbb{E}(X) = \sum_{r \in \mathrm{im}(X)} r \cdot \mathbb{P}(X = r),$$

*where $\infty \cdot 0 = 0$ and $\infty \cdot p = \infty$ for $p > 0$.*

- *If $X$ only takes values in $\overline{\mathbb{R}}_{\geq 0} = \mathbb{R}_{\geq 0} \cup \{\infty\}$, then $\mathbb{E}(X) \in \overline{\mathbb{R}}_{\geq 0}$ always exists, where $\mathbb{E}(X) = \infty$ if $\mathbb{P}(X = \infty) > 0$ or the sum diverges.*
- *Otherwise, $\mathbb{E}(X) \in \mathbb{R}$ is only defined if $\sum_{r \in \mathrm{im}(X)} |r| \cdot \mathbb{P}(X = r) < \infty$.*

If $\Omega$ is countable, $\mathfrak{A} = \mathrm{Pow}(\Omega)$, and $\mathbb{E}(X)$ is finite, then $\mathbb{E}(X) = \sum_{r \in \mathrm{im}(X)} r \cdot \mathbb{P}(X = r) = \sum_{\omega \in \Omega} X(\omega) \cdot \mathbb{P}(\{\omega\})$.

If $X$ only takes values in $\overline{\mathbb{R}}_{\geq 0}$, then all summands of $\mathbb{E}(X)$ are non-negative, so their order is irrelevant. Otherwise, convergence of $\sum_{r \in \mathrm{im}(X)} |r| \cdot \mathbb{P}(X = r)$ implies convergence of $\mathbb{E}(X) = \sum_{r \in \mathrm{im}(X)} r \cdot \mathbb{P}(X = r)$. So the expected value is well defined. The reason for requiring *absolute* convergence is that otherwise by RIEMANN's rearrangement theorem the order of the summands in the series $\sum_{r \in \mathrm{im}(X)} r \cdot \mathbb{P}(X = r)$ would influence its convergence and its result. If $\mathrm{im}(X)$ is a finite subset of $\mathbb{R}$, then $\mathbb{E}(X)$ is finite.

Example 2.2.6 (Expected Value). Let us continue Example 2.2.4. The image of $X$ is finite, hence the expected value is finite. Note that $\mathbb{P}^X(\{2\}) = \mathbb{P}^X(\{12\}) = \frac{1}{36}$, $\mathbb{P}^X(\{3\}) = \mathbb{P}^X(\{11\}) = \frac{2}{36}$, …, $\mathbb{P}^X(\{7\}) = \frac{6}{36}$. Hence, we obtain

$$\mathbb{E}(X) = \sum_{i=2}^{12} i \cdot \mathbb{P}(X = i) = \tfrac{1\cdot(2+12)+2\cdot(3+11)+3\cdot(4+10)+4\cdot(5+9)+5\cdot(6+8)+6\cdot 7}{36} = \tfrac{252}{36} = 7.$$

So the expected value for the sum of throwing a die twice is 7.

The expected value of a random variable will be used to define the expected runtime of a probabilistic program. Moreover, the expected value of specific random variables gives crucial information about the termination probability and expected runtime of a probabilistic program. We see an example of this statement in Section 2.3, where the expected value after a single transition fully characterizes the termination behavior.

The following lemma shows that we can compose random variables that map into $\mathbb{R}^n$ with measurable maps $g : (\mathbb{R}^n, \mathfrak{B}_{\mathbb{R}^n}) \to (\mathbb{R}, \mathfrak{B}_{\mathbb{R}})$ to obtain further random variables.

Lemma 2.2.7 – Composition of Random Variables.

*Let $X \sim (\Omega, \mathfrak{A}, \mathbb{P}) \to (\mathbb{R}^n, \mathfrak{B}_{\mathbb{R}^n})$ be a discrete random variable and $g : (\mathbb{R}^n, \mathfrak{B}_{\mathbb{R}^n}) \to (\mathbb{R}, \mathfrak{B}_{\mathbb{R}})$ be a $(\mathfrak{B}_{\mathbb{R}^n}\text{–}\mathfrak{B}_{\mathbb{R}})$-measurable map. Then, $g \circ X \sim (\Omega, \mathfrak{A}, \mathbb{P}) \to (\mathbb{R}, \mathfrak{B}_{\mathbb{R}})$ is a discrete random variable as well, and if $\mathbb{E}(g \circ X)$ is finite, then*

$$\mathbb{E}(g \circ X) = \sum_{r \in \mathbb{R}^n} g(r) \cdot \mathbb{P}(X = r).$$

*Proof.* Since measurable functions are closed under composition, we know that $g \circ X$ is measurable, and thus, $g \circ X$ is a random variable. Discreteness follows from the discreteness of $X$. If $\mathbb{E}(g \circ X)$ is finite, then $\sum_{r \in \mathbb{R}} |r| \cdot \mathbb{P}((g \circ X)^{-1}(\{r\})) < \infty$ by Definition 2.2.5. Grouping non-negative terms yields $\sum_{s \in X(\Omega)} |g(s)| \cdot \mathbb{P}(X^{-1}(\{s\})) = \sum_{r \in \mathbb{R}} |r| \cdot \mathbb{P}((g \circ X)^{-1}(\{r\})) < \infty$, so absolute convergence justifies rearranging the sum, and we obtain

$$\begin{aligned}
\mathbb{E}(g \circ X) &= \sum_{r \in \mathbb{R}} r \cdot \mathbb{P}^{g \circ X}(\{r\}) & &= \sum_{r \in \mathbb{R}} r \cdot \mathbb{P}((g \circ X)^{-1}(\{r\})) \\
&= \sum_{r \in \mathbb{R}} r \cdot \mathbb{P}(X^{-1}(g^{-1}(\{r\}))) & &= \sum_{r \in \mathbb{R}} r \cdot \mathbb{P}(X^{-1}(\underbrace{g^{-1}(\{r\}) \cap X(\Omega)}_{\text{discrete}})) \\
&= \sum_{r \in \mathbb{R}} r \cdot \sum_{s \in X(\Omega), g(s) = r} \mathbb{P}(X^{-1}(\{s\})) & &= \sum_{s \in X(\Omega)} g(s) \cdot \mathbb{P}(X^{-1}(\{s\})) \\
&= \sum_{s \in \mathbb{R}^n} g(s) \cdot \mathbb{P}(X^{-1}(\{s\})) & &= \sum_{r \in \mathbb{R}^n} g(r) \cdot \mathbb{P}(X = r).
\end{aligned}$$

*Proof* *QED*

*Remark* 2.2.8. In Lemma 2.2.7, a $(\mathfrak{B}_{\mathbb{R}^n}\text{–}\mathfrak{B}_{\mathbb{R}})$-measurable function $g : \mathbb{R}^n \to \mathbb{R}$ is used. Notice that any continuous function satisfies this condition.

The addition of two numbers $+ : \mathbb{R}^2 \to \mathbb{R}$ is continuous and hence measurable. By defining addition on functions pointwise we can use it to combine random variables. If $X \sim (\Omega, \mathfrak{A}, \mathbb{P})$ and $Y \sim (\Omega, \mathfrak{A}, \mathbb{P})$ are real-valued discrete random variables, then $X + Y \sim (\Omega, \mathfrak{A}, \mathbb{P})$ defined by $\omega \mapsto X(\omega) + Y(\omega)$ is a discrete random variable as well. Moreover, in the case of addition we get a nice modularity result regarding the expected value.

Lemma 2.2.9 – Addition of Expected Values.

*Let $X, Y \sim (\Omega, \mathfrak{A}, \mathbb{P})$ be real-valued discrete random variables. Then, $X + Y$ is also a discrete random variable, and if both $\mathbb{E}(X)$ and $\mathbb{E}(Y)$ are finite, then*

$$\mathbb{E}(X + Y) = \mathbb{E}(X) + \mathbb{E}(Y).$$

*Proof.* The map $(X, Y) : \Omega \to \mathbb{R}^2$ defined by $(X, Y)(\omega) = (X(\omega), Y(\omega))$ is a discrete random variable. Since addition $+ : \mathbb{R}^2 \to \mathbb{R}$ is continuous, it is measurable, and thus *Proof*

$X + Y = + \circ (X, Y)$ is a discrete random variable by Lemma 2.2.7. Moreover, we have $\sum_{(r_1,r_2)\in\mathbb{R}^2} |r_1 + r_2| \cdot \mathbb{P}((X, Y) = (r_1, r_2)) \leq \sum_{r_1\in\mathbb{R}} |r_1| \cdot \mathbb{P}(X = r_1) + \sum_{r_2\in\mathbb{R}} |r_2| \cdot \mathbb{P}(Y = r_2) < \infty$, so $\mathbb{E}(X+Y)$ is finite, and using Lemma 2.2.7 and absolute convergence, we obtain

$$\begin{aligned}
&\mathbb{E}(X + Y) \\
&= \sum_{(r_1,r_2)\in\mathbb{R}^2} (r_1 + r_2) \cdot \mathbb{P}((X, Y) = (r_1, r_2)) \\
&= \sum_{(r_1,r_2)\in\mathbb{R}^2} r_1 \cdot \mathbb{P}((X, Y) = (r_1, r_2)) + \sum_{(r_1,r_2)\in\mathbb{R}^2} r_2 \cdot \mathbb{P}((X, Y) = (r_1, r_2)) \\
&= \sum_{r_1\in\mathbb{R}} r_1 \cdot \mathbb{P}(X = r_1) + \sum_{r_2\in\mathbb{R}} r_2 \cdot \mathbb{P}(Y = r_2) \\
&= \mathbb{E}(X) + \mathbb{E}(Y).
\end{aligned}$$

*QED*

Example 2.2.10. We continue Example 2.2.4 and consider throwing two dice simultaneously. Both throws can be seen as two random variables $X_1, X_2 \sim (\Omega^2_{\mathsf{dice}}, \mathrm{Pow}(\Omega^2_{\mathsf{dice}}), \mathbb{P})$ over the Laplace space from Example 2.2.4 with $X_1(i, j) = i$ and $X_2(i, j) = j$. The expected value of both $X_1$ and $X_2$ is

$$\mathbb{E}(X_1) = \mathbb{E}(X_2) = \sum_{i=1}^{6} i \cdot \mathbb{P}(X_1 = i) = \tfrac{1+2+3+4+5+6}{6} = \tfrac{21}{6} = \tfrac{7}{2}.$$

The random variable $X$ from Example 2.2.4 can be defined as $X = X_1 + X_2$, and thus, we have $\mathbb{E}(X) = \mathbb{E}(X_1 + X_2) = \mathbb{E}(X_1) + \mathbb{E}(X_2) = \frac{7}{2} + \frac{7}{2} = 7$.

To get the same modularity result regarding multiplication, we have to require *independence* of the random variables. Intuitively, two random variables $X$ and $Y$ are independent if the value of $X$ does not depend on the value of $Y$, and vice versa.

Definition 2.2.11 – Independence.

*A family $(X_i)_{i\in I}$ of random variables $X_i \sim (\Omega, \mathfrak{A}, \mathbb{P})$ is independent if for every finite subset $J \subseteq I$ and all measurable sets $A_j \in \mathfrak{B}$ we have*

$$\mathbb{P}\Big(\bigcap_{j\in J} \{X_j \in A_j\}\Big) = \prod_{j\in J} \mathbb{P}(X_j \in A_j).$$

So, independence of two discrete random variables $X$ and $Y$ simply means $\mathbb{P}(X = x, Y = y) = \mathbb{P}(X = x) \cdot \mathbb{P}(Y = y)$ for all values $x$ and $y$, where $\mathbb{P}(X = x, Y = y)$ denotes $\mathbb{P}(\{X = x\} \cap \{Y = y\})$. We get the following two important lemmas for independent random variables concerning conditional probability and the expected value of $X \cdot Y$.

Example 2.2.12. Both die throws $X_1, X_2 \sim (\Omega^2_{\mathsf{dice}}, \mathrm{Pow}(\Omega^2_{\mathsf{dice}}), \mathbb{P})$ from Example 2.2.10 are two independent random variables. For example, we have $\mathbb{P}(X_1 = 1, X_2 = 1) = \mathbb{P}(X_1 = 1) \cdot \mathbb{P}(X_2 = 1) = 1/6 \cdot 1/6 = 1/36$.

Lemma 2.2.13 – Independent Variables and Conditional Probability.

*Let $X \sim (\Omega, \mathfrak{A}, \mathbb{P})$ and $Y \sim (\Omega, \mathfrak{A}, \mathbb{P})$ be independent random variables. Then, for all $x, y \in \mathbb{R}$ we have $\mathbb{P}(X = x \mid Y = y) = \mathbb{P}(X = x)$ if $\mathbb{P}(Y = y) > 0$, and*

$\mathbb{P}(Y = y \mid X = x) = \mathbb{P}(Y = y)$ *if* $\mathbb{P}(X = x) > 0$.

*Proof.* We have

$$\mathbb{P}(X = x \mid Y = y) = \frac{\mathbb{P}(X = x, Y = y)}{\mathbb{P}(Y = y)} = \frac{\mathbb{P}(X = x) \cdot \mathbb{P}(Y = y)}{\mathbb{P}(Y = y)} = \mathbb{P}(X = x),$$

where the first equation is the definition of conditional probability and the second one follows by independence of $X$ and $Y$. The other equation can be shown by swapping $X$ and $Y$. *QED*

Lemma 2.2.14 – Multiplication of Independent Random Variables.

*Let $X \sim (\Omega, \mathfrak{A}, \mathbb{P})$ and $Y \sim (\Omega, \mathfrak{A}, \mathbb{P})$ be real-valued discrete random variables. Then, $X \cdot Y$ is also a discrete random variable, and if $X$ and $Y$ are independent and both $\mathbb{E}(X)$ and $\mathbb{E}(Y)$ are finite, we have*

$$\mathbb{E}(X \cdot Y) = \mathbb{E}(X) \cdot \mathbb{E}(Y).$$

*Proof.* Similar to the proof of Lemma 2.2.9 we can show that $X \cdot Y$ is indeed a discrete random variable. By independence, the series $\sum_{(r_1,r_2)\in\mathbb{R}^2} |r_1 \cdot r_2| \cdot \mathbb{P}^{(X,Y)}(\{(r_1, r_2)\})$ is equal to $\left(\sum_{r_1\in\mathbb{R}} |r_1| \cdot \mathbb{P}^X(\{r_1\})\right) \cdot \left(\sum_{r_2\in\mathbb{R}} |r_2| \cdot \mathbb{P}^Y(\{r_2\})\right) < \infty$, so $\mathbb{E}(X \cdot Y)$ is finite and we may rearrange its sum. Moreover, we get

$$\begin{aligned}
&\mathbb{E}(X \cdot Y) \\
&= \sum_{(r_1,r_2)\in\mathbb{R}^2} (r_1 \cdot r_2) \cdot \mathbb{P}^{(X,Y)}(\{(r_1, r_2)\}) \overset{(*)}{=} \sum_{(r_1,r_2)\in\mathbb{R}^2} (r_1 \cdot r_2) \cdot \mathbb{P}^X(\{r_1\}) \cdot \mathbb{P}^Y(\{r_2\}) \\
&= \sum_{r_1\in\mathbb{R}} \sum_{r_2\in\mathbb{R}} r_1 \cdot \mathbb{P}^X(\{r_1\}) \cdot r_2 \cdot \mathbb{P}^Y(\{r_2\}) = \left(\sum_{r_1\in\mathbb{R}} r_1 \cdot \mathbb{P}^X(\{r_1\})\right) \cdot \left(\sum_{r_2\in\mathbb{R}} r_2 \cdot \mathbb{P}^Y(\{r_2\})\right) \\
&= \mathbb{E}(X) \cdot \mathbb{E}(Y),
\end{aligned}$$

where equation $(*)$ is due to independence of $X$ and $Y$. *QED*

Example 2.2.15. We continue Example 2.2.10, but instead of summing the results, we multiply the results of both throws. Both throws are independent of each other, and thus, we have $\mathbb{E}(X_1 \cdot X_2) = \mathbb{E}(X_1) \cdot \mathbb{E}(X_2) = \frac{7}{2} \cdot \frac{7}{2} = \frac{49}{4}$.

SECTION 2.3

## Random Walks and Random Walk Programs

In the remainder of this chapter, we recapitulate multiple important and well-known stochastic systems from the literature following a uniform structure for each of them. First, we present the system and define its core syntax. Then, we show how we can define a probability space for the system. Using the probability space, we can then explain what termination means for the system. To be precise, we explain how to measure the probability of termination or the expected runtime, which is the expected number of steps one has to perform before reaching a terminal state. If the probability of termination is 1, then we call the system *almost surely terminating*. Moreover, if the expected runtime is finite, then the system is *positively almost surely terminating*. In the end, we show for each of the systems how we can prove (positive) almost-sure termination completely automatically. All the results in this section follow from classical results on random walks and can be found in, e.g., [Spi01, LL10, GGH19].

### 2.3.1 Random Walks

We start by considering one of the most important stochastic processes used in this thesis: *random walks*, also known as *drunkard's walks*, often occurring in public after fairs or carnivals like the "All Saints' Day Fair" in Soest.[2] We consider only one-dimensional random walks on $\mathbb{Z}$ in this thesis. For a more detailed introduction to higher-dimensional random walks, see [Spi01] for random walks on $\mathbb{Z}^d$ and [LL10] for a modern approach to random walks on arbitrary universes.

Before introducing random walks, we have to introduce the fundamental concept of a *stochastic process*. A stochastic process is a sequence of random variables that describe how a stochastic system evolves over time. It is used as a mathematical model for uncertainties in real-world scenarios in, e.g., biology or chemistry.

Definition 2.3.1 – Stochastic Process.
*Let $T$ be some index set, and let $X_t \sim (\Omega, \mathfrak{A}, \mathbb{P})$ be a random variable for all $t \in T$. The family $\hat{X} = (X_t)_{t \in T}$ is called a stochastic process.*

Again, we are only interested in the discrete-time case, where $T$ is countable, e.g., $T = \{x \in \mathbb{N} \mid 1 \leq x \leq 10\}$, $T = \mathbb{N}$, or $T = \mathbb{Z}$.

Example 2.3.2 (Coin Flipping). Consider a program that flips a fair coin. If the coin lands on tails, we throw the coin again, and if the coin lands on heads, we stop.

The termination behavior of the program can be described by a stochastic process $(X_n)_{n \in \mathbb{N}}$ with $X_0 = \mathbb{1}_0$ whose distributions satisfy $\mathbb{P}(X_{n+1} = 1) = \mathbb{P}(X_n = 1) + 1/2 \cdot \mathbb{P}(X_n = 0)$, and $\mathbb{P}(X_{n+1} = 0) = 1/2 \cdot \mathbb{P}(X_n = 0)$. Here, $\mathbb{P}(X_n = 1)$ models the probability that we have stopped before or at the $n$-th iteration, and $\mathbb{P}(X_n = 0)$ models the probability that the program performs at least $n + 1$ coin flips.

One particular type of stochastic process that we are interested in is *random walks*. A random walk results from the iterative summation of independent and identically distributed random variables $Y_i$ (i.e., we have $\mathbb{P}^{Y_i} = \mathbb{P}^{Y_j}$ for all $i, j \in \mathbb{N}_{>0}$), where the initial value is given by an additional Dirac delta variable $\mathbb{1}_x$ with $x \in \mathbb{Z}$.

Definition 2.3.3 – Random Walk.
*Let $(\Omega, \mathfrak{A}, \mathbb{P})$ be a probability space, let $\hat{Y} = (Y_n)_{n \in \mathbb{N}_{>0}}$ be independent and identically distributed random variables with $Y_i \sim (\Omega, \mathfrak{A}, \mathbb{P}) \to \mathbb{Z}$ for all $i \in \mathbb{N}_{>0}$, and let $x_0 \in \mathbb{Z}$ be the initial state. The stochastic process $\hat{S} = (S_n)_{n \in \mathbb{N}}$ where $S_n = \mathbb{1}_{x_0} + Y_1 + \ldots + Y_n = \mathbb{1}_{x_0} + \sum_{i=1}^{n} Y_i$ for all $n \in \mathbb{N}$ is called a random walk. We often denote the random walk by $\hat{S} = (\mathbb{1}_{x_0}, \hat{Y})$.*

Example 2.3.4 (Symmetric Random Walk). Imagine a drunk person who tries to go home after a fair. For simplicity, we assume that their way home is a simple line, and they have to take 1000 steps to reach their home. Since the person is drunk, there is only a 50% chance that they move towards their home. In the other 50%, they take a step backwards, away from home. We can view this mathematically as the random walk $(\mathbb{1}_{1000}, \hat{Y})$ with $\mathbb{P}(Y_i = 1) = \mathbb{P}(Y_i = -1) = 1/2$ for all $i \in \mathbb{N}_{>0}$.

Due to our formal introduction of probability spaces and random variables, we do not

[2] The "All Saints' Day Fair" in Soest is the largest old town fair in Europe and at least 687 years old. Moreover, it is the largest fair in the author's hometown.

have to present another definition of the probability space for a random walk, but the probability space after the $n$-th iteration is given by the induced probability space $(\mathbb{Z}, \{\mathbb{Z} \cap B \mid B \in \mathfrak{B}\}, \mathbb{P}^{S_n})$ of the random variable $S_n \sim (\Omega, \mathfrak{A}, \mathbb{P}) \to \mathbb{Z}$.

**Notions of Termination for Random Walks** Stochastic processes, and more specifically random walks, evolve infinitely whenever the index set $T$ is infinite, e.g., $T = \mathbb{N}$. For such an infinitely running system, we have to define a "termination criterion" that specifies when the system is considered to have terminated. To this end, we fix a target set of terminal states, i.e., a subset of $\mathbb{Z}$ for random walks, and we compute the probability to "hit" this set of terminal states eventually. The random variable that describes how long it takes to hit the terminal states is called the *hitting time*. In the following, we often consider a random walk to terminate once it hits a non-positive integer. Therefore, we use the set of terminal states $\mathbb{Z}_{\leq 0}$.

Definition 2.3.5 – Hitting Time.

*Let $F \subseteq \mathbb{Z}$ be a set of terminal states or goal states. The hitting time for a random walk $\hat{S} = (S_n)_{n\in\mathbb{N}}$ w.r.t. the set $F$ is the random variable $T_F \sim (\Omega, \mathfrak{A}, \mathbb{P}) \to \mathbb{N} \cup \{\infty\}$ with $T_F(\omega) = \inf\{j \in \mathbb{N} \mid S_j(\omega) \in F\}$, where $\inf \varnothing = \infty$.*

Example 2.3.6 (Hitting Time). We continue Example 2.3.4, and assume that the house of the drunk person is at position 0. Therefore, we can consider the terminal states $F = \mathbb{Z}_{\leq 0}$. They first visit their own home after $T_F$ steps. Interesting questions that arise are: "Will the person reach their home with probability 1?" ($\mathbb{P}(T_F < \infty) = 1$; termination probability) and "Do they reach their home in finitely many steps in expectation?" ($\mathbb{E}(T_F) < \infty$; expected runtime).

The *drift* of a random walk, i.e., the expected change in each step of the random walk, gives insights about the termination probability and the runtime, and is an important measure for the behavior of the random walk in general.

Definition 2.3.7 – Drift.

*Let $\hat{S} = (\mathbb{1}_{x_0}, (Y_n)_{n\in\mathbb{N}_{>0}})$ be a random walk with $\mathbb{E}(|Y_1|) < \infty$. By $\mathbb{E}(Y_1)$ we denote the drift of $\hat{S}$.*

Example 2.3.8 (Drift). We have $\mathbb{E}(Y_1) = 1/2 \cdot (-1) + 1/2 \cdot 1 = 0$ for the random variable $Y_1$ from Example 2.3.4. Therefore, the random walk $(\mathbb{1}_{1000}, \hat{Y})$ has drift 0, i.e., the drunk person will neither move towards their home nor move away from it in expectation.

Using the drift of a random walk and the distribution of its steps $Y_1$, we can classify each random walk into four different types. Here, we restrict the random walks to have finite drift. This is important to fully classify the different types of random walks in terms of termination and runtime behavior.

Definition 2.3.9 – Classification of Random Walks with Finite Drift.

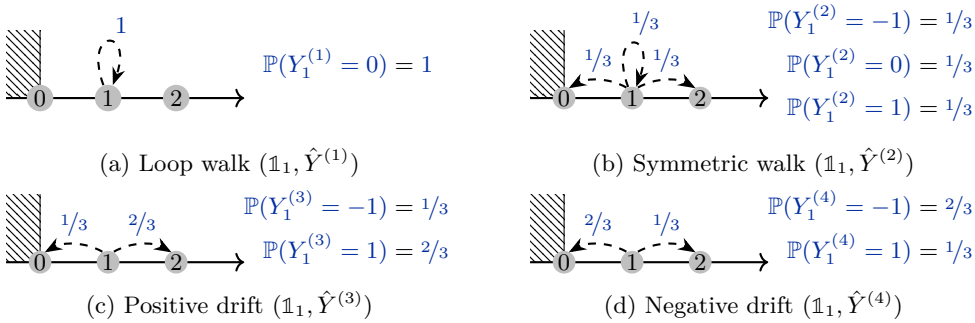
*Let $\hat{S} = (\mathbb{1}_{x_0}, (Y_n)_{n\in\mathbb{N}_{>0}})$ be a random walk such that $\mathbb{E}(|Y_1|) < \infty$.*

- *If $\mathbb{E}(Y_1) = 0$ and $\mathbb{P}(Y_1 = 0) = 1$, then we say $\hat{S}$ is a loop walk.*
- *If $\mathbb{E}(Y_1) = 0$ and $\mathbb{P}(Y_1 = 0) < 1$, then $\hat{S}$ is a symmetric random walk.*
- *If $\mathbb{E}(Y_1) > 0$, then $\hat{S}$ is a positively drifting random walk.*

- *If $\mathbb{E}(Y_1) < 0$, then $\hat{S}$ is a negatively drifting random walk.*

Example 2.3.10. The random walk from Example 2.3.4 regarding the drunken person is a symmetric random walk, because we have $\mathbb{E}(Y_1) = 0$ and $\mathbb{P}(Y_1 = 0) = 0 < 1$.

In Figure 2.1 one can see examples for all four different types of random walks.




(a) Loop walk $(\mathbb{1}_1, \hat{Y}^{(1)})$

(b) Symmetric walk $(\mathbb{1}_1, \hat{Y}^{(2)})$

(c) Positive drift $(\mathbb{1}_1, \hat{Y}^{(3)})$

(d) Negative drift $(\mathbb{1}_1, \hat{Y}^{(4)})$

Figure 2.1: Four different random walks $(\mathbb{1}_1, \hat{Y}^{(1)})$, $(\mathbb{1}_1, \hat{Y}^{(2)})$, $(\mathbb{1}_1, \hat{Y}^{(3)})$, and $(\mathbb{1}_1, \hat{Y}^{(4)})$.

For all four types of random walks (with finite drift) we can precisely state whether the hitting time is finite (with probability 1) or whether the expected number of steps to hit $\mathbb{Z}_{\leq 0}$ is finite.

Theorem 2.3.11 – Drift and Hitting Time, [Spi01, Thm. 17.1, Prop. 18.1].

Let $\hat{S} = (\mathbb{1}_{x_0}, \hat{Y})$ be a random walk such that $\mathbb{E}(|Y_1|) < \infty$ and $x_0 > 0$. Let $T_{\mathbb{Z}_{\leq 0}}$ be the hitting time for $\hat{S}$ and the set $\mathbb{Z}_{\leq 0}$.

- If $\hat{S}$ is a loop walk, then $\mathbb{P}(T_{\mathbb{Z}_{\leq 0}} = \infty) > 0$ and $\mathbb{E}(T_{\mathbb{Z}_{\leq 0}}) = \infty$.
- If $\hat{S}$ is symmetric, then $\mathbb{P}(T_{\mathbb{Z}_{\leq 0}} = \infty) = 0$ and $\mathbb{E}(T_{\mathbb{Z}_{\leq 0}}) = \infty$.
- If $\hat{S}$ is positively drifting, then $\mathbb{P}(T_{\mathbb{Z}_{\leq 0}} = \infty) > 0$ and $\mathbb{E}(T_{\mathbb{Z}_{\leq 0}}) = \infty$.
- If $\hat{S}$ is negatively drifting, then $\mathbb{P}(T_{\mathbb{Z}_{\leq 0}} = \infty) = 0$ and $\mathbb{E}(T_{\mathbb{Z}_{\leq 0}}) < \infty$.

The two conditions $\mathbb{P}(T_{\mathbb{Z}_{\leq 0}} = \infty) = 0$ and $\mathbb{E}(T_{\mathbb{Z}_{\leq 0}}) < \infty$ are called *almost-sure termination* and *positive almost-sure termination*, respectively. These are the two most important properties for stochastic systems, and the properties we want to analyze automatically in this thesis.

Definition 2.3.12 – Almost-Sure Termination.

*Let $\hat{S} = (\mathbb{1}_{x_0}, \hat{Y})$ be a random walk such that $\mathbb{E}(|Y_1|) < \infty$. Let $T_{\mathbb{Z}_{\leq 0}}$ be the hitting time for $\hat{S}$ and the set $\mathbb{Z}_{\leq 0}$. We call $\hat{S}$ almost surely terminating (*`AST`*) if $\mathbb{P}(T_{\mathbb{Z}_{\leq 0}} = \infty) = 0$.*

Definition 2.3.13 – Positive Almost-Sure Termination.

*Let $\hat{S} = (\mathbb{1}_{x_0}, \hat{Y})$ be a random walk such that $\mathbb{E}(|Y_1|) < \infty$. Let $T_{\mathbb{Z}_{\leq 0}}$ be the hitting time for $\hat{S}$ and the set $\mathbb{Z}_{\leq 0}$. We call $\hat{S}$ positively almost surely terminating (*`PAST`*) if $\mathbb{E}(T_{\mathbb{Z}_{\leq 0}}) < \infty$.*

We get the following corollary, which shows that neither AST nor PAST depends on the concrete initial state $x_0 > 0$.

Corollary 2.3.14 – Drift and Hitting Time, [Spi01].

*Let $\hat{S} = (\mathbb{1}_{x_0}, \hat{Y})$ be a random walk such that $\mathbb{E}(|Y_1|) < \infty$ and $x_0 > 0$.*

- *If $\hat{S}$ is a loop walk, then $\hat{S}$ is neither AST nor PAST.*
- *If $\hat{S}$ is symmetric, then $\hat{S}$ is AST but not PAST.*
- *If $\hat{S}$ is positively drifting, then $\hat{S}$ is neither AST nor PAST.*
- *If $\hat{S}$ is negatively drifting, then $\hat{S}$ is both AST and PAST.*

Example 2.3.15. Reconsider the random walks from Figure 2.1. Based on Corollary 2.3.14, $(\mathbb{1}_1, \hat{Y}^{(1)})$ and $(\mathbb{1}_1, \hat{Y}^{(3)})$ are neither AST nor PAST. The random walk $(\mathbb{1}_1, \hat{Y}^{(2)})$ is AST but not PAST, and $(\mathbb{1}_1, \hat{Y}^{(4)})$ is both AST and PAST.

While AST and PAST of random walks are easy to characterize, a random walk, as a stochastic process, has no explicit algorithmic structure. Therefore, we now define a probabilistic program that resembles these random walks, where we can then automatically infer AST and PAST via Corollary 2.3.14.

### 2.3.2 Random Walk Programs

In this subsection, we simplify the notion of a random walk to discover an algorithmic structure that can be characterized in terms of AST and PAST similar to ordinary random walks in Corollary 2.3.14. Since we are mostly interested in termination probabilities and expected runtimes, we do not need to consider the remaining executions of a run that has already hit one of the terminal states. Moreover, we are mostly interested in the terminal states $\mathbb{Z}_{\leq 0}$. Therefore, we simplify the notion of a random walk by fixing the terminal states and stopping once we are not positive anymore. Additionally, we remove the initial value from the basic syntax, as we have seen in Corollary 2.3.14 that it is irrelevant for AST and PAST. This leads to the notion of a *random walk program*, which is a simple *distribution function* $\mu : \mathbb{Z} \to \mathbb{R}_{\geq 0}$ whose *support* $\mathrm{Supp}(\mu) = \{x \in \mathbb{Z} \mid \mu(x) > 0\}$ satisfies $\sum_{x \in \mathrm{Supp}(\mu)} \mu(x) = 1$.

Definition 2.3.16 – Distribution Function, Expected Value.

*Let $A$ be a non-empty set. A distribution function $\mu : A \to \mathbb{R}_{\geq 0}$ satisfies $\sum_{x \in \mathrm{Supp}(\mu)} \mu(x) = 1$, where $\mathrm{Supp}(\mu) = \{x \in A \mid \mu(x) > 0\}$. We call $\mu$ a distribution on $\mathbb{Z}$ if $A = \mathbb{Z}$. If $A \subseteq \overline{\mathbb{R}}$, then the expected value $\mathbb{E}(\mu)$ of $\mu$ is the expected value of the identity on the probability space $(\mathrm{Supp}(\mu), \mathrm{Pow}(\mathrm{Supp}(\mu)), \mathbb{P}_\mu)$ with $\mathbb{P}_\mu(B) = \sum_{x \in B} \mu(x)$, i.e., $\mathbb{E}(\mu) = \sum_{x \in \mathrm{Supp}(\mu)} x \cdot \mu(x)$. For a distribution $\mu$ on $\mathbb{Z}$, $\mathbb{E}(\mu)$ is also called the drift of $\mu$. If $\mathrm{Supp}(\mu) = \{x_1, \ldots, x_k\}$ is finite with pairwise distinct $x_i$, we write $\mu = \{p_1 x_1, \ldots, p_k x_k\}$ if $\mu(x_i) = p_i$ for all $1 \leq i \leq k$.*

Instead of stochastic processes, we can simply consider a distribution on $\mathbb{Z}$ to model the transition relation of a random walk on $\mathbb{Z}$, where we stop at any non-positive number in $\mathbb{Z}_{\leq 0}$. Again, we only consider random walk programs that have finite drift.

Definition 2.3.17 – Random Walk Program.

*A random walk program (RWP) $\mu$ is a distribution on $\mathbb{Z}$ such that $\mathbb{E}(\mu)$ is defined and fininte.*

**Algorithm 2:**
Random walk program

$x \leftarrow 1$
**while** $x > 0$ **do**
  $x \leftarrow_{p_{-k}} x - k$
  $\dots$
  $x \leftarrow_{p_{-1}} x - 1$
  $x \leftarrow_{p_0} x$
  $x \leftarrow_{p_1} x + 1$
  $\dots$
  $x \leftarrow_{p_h} x + h$

Example 2.3.18 (Random Walk Program). Random walk programs with finite support can be visualized as the imperative program seen in Algorithm 2. Here, we initialize the variable $x$ with 1, and then, we keep adjusting the value of $x$ according to the distribution $\mu = \{{}^{p_{-k}}{-k}, \dots, {}^{p_{-1}}{-1}, {}^{p_0}0, {}^{p_1}1, \dots, {}^{p_h}h\}$ with $k, h \in \mathbb{N}$ and $p_{-k} + \dots + p_h = 1$ until eventually the value of $x$ becomes negative or 0.

The function $\mu$ represents the transition relation of a random walk on $\mathbb{Z}$. If the current value of the random walk is $y > 0$, then for any $x \in \mathbb{Z}$, $\mu(x)$ is the probability that the random walk transitions from $y$ to $y + x$. A random walk program stops when reaching a number $\leq 0$ and does not perform any transition steps anymore.

**Probability Space of a Random Walk Program** Given a random walk program, we want to define a probability space on all infinite sequences of integers $\mathbb{Z}^\omega$ that are possible *executions* or *runs* of the program. However, $\mathbb{Z}^\omega$ is uncountable, and as discussed in Section 2.1, we cannot simply take $\mathrm{Pow}(\mathbb{Z}^\omega)$ as our $\sigma$-field. We have to come up with a suitable $\sigma$-field that is able to express the event of all terminating runs. Since all terminating runs are finite, we can consider a $\sigma$-field based on the finite runs in a random walk program, which are only countably many. To lift this construction from finite to infinite runs, we consider the *cylinder set* of a finite run $\pi$, which contains all runs that have $\pi$ as prefix.

Remark 2.3.19 – Finite Words, Infinite Words, Prefix, Concatenation.

We fix some notation for (potentially infinite) words. Let $X$ be some set. A *non-empty finite word over* $X$ is a sequence $\pi = x_0 \dots x_k$ such that $x_i \in X$ for all $0 \leq i \leq k$, and its *length* is defined as $|x_0 \dots x_k| = k + 1$. The *empty word* is denoted by $\epsilon$ with $|\epsilon| = 0$. By $X^+$ we denote the set of all non-empty finite words over $X$, and by $X^* = X^+ \cup \{\epsilon\}$ we denote the set of all finite words over $X$. An *infinite word over* $X$ is a sequence $x_0 x_1 \dots$ such that $x_i \in X$ for all $i \in \mathbb{N}$. By $X^\omega$ we denote the set of all infinite words over $X$. A set of words $L \subseteq X^* \cup X^\omega$ is called a *language*. A word $x_0 \dots x_k$ is a prefix of a finite word $y_0 \dots y_{k'}$ if $k \leq k'$ and $x_i = y_i$ for all $0 \leq i \leq k$. Analogously, a word $x_0 \dots x_k$ is a prefix of an infinite word $y_0 \dots$ if $x_i = y_i$ for all $0 \leq i \leq k$. The set of all finite prefixes of a word $\pi \in X^* \cup X^\omega$ is denoted by $\mathrm{pref}(\pi)$.
We can *concatenate* words and even languages to form new words and languages, respectively. Given a finite word $\pi = x_0 \dots x_k$ and a (potentially infinite) word $\tau = y_0 y_1 \dots$, the concatenation of both is the word $\pi\tau = \pi \cdot \tau = x_0 \dots x_k y_0 y_1 \dots$, where $\epsilon\tau = \tau$. Given two languages $L_1 \subseteq X^*$ and $L_2 \subseteq X^* \cup X^\omega$, the concatenation is $L_1 \cdot L_2 = \{xy \mid x \in L_1, y \in L_2\}$.

Definition 2.3.20 – Run, Prefix Run, Cylinder Set (Random Walk Program).

*An infinite run in a random walk program $\mu$ is an infinite sequence of states $s_0 s_1 \ldots \in \mathbb{Z}^\omega$ such that $\mu(s_{i+1} - s_i) > 0$ and $s_i > 0$ for all $i$. A finite run (or prefix run) is a finite sequence of states $s_0 \ldots s_k \in \mathbb{Z}^*$ such that $\mu(s_{i+1} - s_i) > 0$ and $s_i > 0$ for all $0 \leq i \leq k-1$. We say that a finite run $\pi = s_0 \ldots s_k$ terminates if $s_k \leq 0$. The set of all maximal runs in $\mu$ is denoted by $\mathrm{Runs}(\mu)$, the set of all finite runs in $\mu$ is denoted by $\mathrm{Runs}_{\mathrm{fin}}(\mu)$, and the set of all terminating runs is $\mathrm{Runs}_{\mathtt{SN}}(\mu)$. The cylinder set of a finite run $\pi = s_0 \ldots s_n \in \mathrm{Runs}_{\mathrm{fin}}(\mu)$ is defined as $\mathrm{Cyl}(\pi) = \{\pi' \in \mathrm{Runs}(\mu) \mid \pi \in \mathrm{pref}(\pi')\}$.*

While an infinite run can only reach positive integers, a finite run may end in any integer. This is due to the fact that we consider any number $\leq 0$ as a terminal state and do not perform any transition steps anymore.

Note that the set of all cylinder sets $\{\mathrm{Cyl}(\pi) \mid \pi \in \mathrm{Runs}_{\mathrm{fin}}(\mu)\}$ is countable, and we can define the $\sigma$-field generated by it. The following classical result from MDP theory [BK08] shows that it suffices to give probabilities to every $\mathrm{Cyl}(\pi)$, or even simpler, to every $\pi \in \mathrm{Runs}_{\mathrm{fin}}(\mu)$, to obtain a unique probability measure $\mathbb{P}^\mu_{x_0}$ for the random walk program $\mu$ (given some start value $x_0 \in \mathbb{Z}$). We formulate it for an arbitrary countable set $X$ of states with a transition function $\mathcal{P}$, where some states may be *terminal*, i.e., they have no successors.

Theorem 2.3.21 – Probability Measures on Runs, [BK08].

Let $X$ be a countable set and let $\mathcal{P} : X \times X \to [0,1]$ such that for every $x \in X$, we either have $\sum_{y \in X} \mathcal{P}(x,y) = 1$, or $\mathcal{P}(x,y) = 0$ for all $y \in X$ (then $x$ is *terminal*). Let Runs be the set of all infinite words $x_0 x_1 \ldots \in X^\omega$ and of all finite words $x_0 \ldots x_k \in X^+$ where $x_k$ is terminal, such that $\mathcal{P}(x_i, x_{i+1}) > 0$ for all $i$, let $\mathrm{Runs}_{\mathrm{fin}}$ be the set of all finite words $x_0 \ldots x_k \in X^+$ with $\mathcal{P}(x_i, x_{i+1}) > 0$ for all $0 \leq i < k$, and let $\mathrm{Cyl}(\pi) = \{\pi' \in \mathrm{Runs} \mid \pi \in \mathrm{pref}(\pi')\}$ for all $\pi \in \mathrm{Runs}_{\mathrm{fin}}$. Then for every $x \in X$, there is a unique probability measure $\mathbb{P}_x$ on $\langle\{\mathrm{Cyl}(\pi) \mid \pi \in \mathrm{Runs}_{\mathrm{fin}}\}\rangle_\sigma$ such that for all $x_0 \ldots x_k \in \mathrm{Runs}_{\mathrm{fin}}$, we have $\mathbb{P}_x(\mathrm{Cyl}(x_0 \ldots x_k)) = 0$ if $x_0 \neq x$, and $\mathbb{P}_x(\mathrm{Cyl}(x_0 \ldots x_k)) = \prod_{i=0}^{k-1} \mathcal{P}(x_i, x_{i+1})$ otherwise.

We shall see several applications of this theorem. Here, we use Theorem 2.3.21 to define the probability space of a random walk program.

Definition 2.3.22 – Probability Space of a Random Walk Program.

*Let $\mu$ be a random walk program, let $x_0 \in \mathbb{Z}$ be some initial state, and let $\mathfrak{A} = \langle\{\mathrm{Cyl}(\pi) \mid \pi \in \mathrm{Runs}_{\mathrm{fin}}(\mu)\}\rangle_\sigma$. By $\mathbb{P}^\mu_{x_0}$ we denote the unique probability measure on $\mathfrak{A}$ from Theorem 2.3.21 for $X = \mathbb{Z}$, $\mathcal{P}(s, s') = \mu(s' - s)$ if $s > 0$, and $\mathcal{P}(s, s') = 0$ if $s \leq 0$, i.e., for every finite run $\pi = s_0 \ldots s_k \in \mathrm{Runs}_{\mathrm{fin}}(\mu)$, we have $\mathbb{P}^\mu_{x_0}(\mathrm{Cyl}(\pi)) = 0$ if $s_0 \neq x_0$, and $\mathbb{P}^\mu_{x_0}(\mathrm{Cyl}(\pi)) = \prod_{i=0}^{k-1} \mu(s_{i+1} - s_i)$ otherwise.*
*The probability space of the random walk program $\mu$ starting in $x_0$ is given by $(\mathrm{Runs}(\mu), \mathfrak{A}, \mathbb{P}^\mu_{x_0})$.*

Example 2.3.23 (Probability Space of a Random Walk Program). Consider the random walk program $\mu = \{^1/_2{-}1, {}^1/_2 0\}$ (representing the coin flip from Example 2.3.2). Possible finite runs that start at 2 for this program include $210 \in \mathrm{Runs}_{\mathrm{fin}}(\mu)$ and $22110 \in \mathrm{Runs}_{\mathrm{fin}}(\mu)$. An infinite run would be $2111 \ldots \in \mathrm{Runs}(\mu)$.

The probability that the run $s_0 s_1 s_2 = 210 \in \mathrm{Runs}_{\mathrm{fin}}(\mu)$ occurs if we start at 2 is

$$\mu(s_1 - s_0) \cdot \mu(s_2 - s_1) = \mu(1-2) \cdot \mu(0-1) = \mu(-1) \cdot \mu(-1) = (1/2)^2 = 1/4.$$

Therefore, we have $\mathbb{P}_2^\mu(\mathrm{Cyl}(210)) = 1/4$.

**Notions of Termination for Random Walk Programs** Next, we introduce the different notions of termination, namely almost-sure termination and positive almost-sure termination, in terms of random walk programs again.

Note that random walk programs do not necessarily run forever, and therefore, they resemble ordinary programs more closely than random walks do. Hence, we could also define a notion of termination where we always terminate no matter how the probabilistic choices resolve. This notion of termination is called *certain termination*.

Definition 2.3.24 – Certain Termination (Random Walk Program).

*Let $\mu$ be a random walk program. We say that $\mu$ certainly terminates if there is no infinite run for $\mu$, i.e.,* $\mathrm{Runs}(\mu) \cap \mathbb{Z}^\omega = \varnothing$.

Example 2.3.25 (Certain Termination). Remember the random walk program from Example 2.3.23, namely $\mu = \{1/2{-}1, 1/2 0\}$. An infinite run of $\mu$ would be $2111\ldots \in \mathrm{Runs}(\mu) \cap \mathbb{Z}^\omega$, and therefore, $\mu$ does not certainly terminate.

Certain termination holds if and only if we have $\mu(x) = 0$ for all $x \geq 0$. Since this is a rather restrictive requirement for stochastic systems, we are more interested in the *termination probability*, and specifically, whether the program terminates with probability 1 or not. In the following definition, as usual, "SN" stands for "strong normalization", which is equivalent to "termination".

Definition 2.3.26 – Probability of Termination.

*Let $\mu$ be a random walk program, let $x_0 \in \mathbb{Z}$, and let* $\mathrm{Cyl}(\mathtt{SN}) = \biguplus_{\pi \in \mathrm{Runs}_{\mathtt{SN}}(\mu)} \mathrm{Cyl}(\pi)$ *be the event of all terminating runs. Then the probability of termination of $\mu$ starting in $x_0$ is* $\mathbb{P}_{x_0}^\mu(\mathrm{Cyl}(\mathtt{SN})) = \sum_{\pi \in \mathrm{Runs}_{\mathtt{SN}}(\mu)} \mathbb{P}_{x_0}^\mu(\mathrm{Cyl}(\pi))$.

Definition 2.3.27 – Almost-Sure Termination (Random Walk Program).

*A random walk program $\mu$ is almost surely terminating (AST) if we have $\mathbb{P}_{x_0}^\mu(\mathrm{Cyl}(\mathtt{SN})) = 1$ for all $x_0 \in \mathbb{Z}$.*

Example 2.3.28 (Probability of Termination). Reconsider the random walk program $\mu = \{1/2{-}1, 1/2 0\}$ from Example 2.3.23, but starting at 1. Then, the event of all terminating runs that start at 1 is $\mathrm{Cyl}(10) \cup \mathrm{Cyl}(110) \cup \cdots$, and all other terminating runs have probability 0. The termination probability is

$$\mathbb{P}_1^\mu(\mathrm{Cyl}(\mathtt{SN})) = \mathbb{P}_1^\mu(\mathrm{Cyl}(10)) + \mathbb{P}_1^\mu(\mathrm{Cyl}(110)) + \cdots = 1/2 + (1/2)^2 + (1/2)^3 + \cdots = 1.$$

Indeed, by Corollary 2.3.33, $\mu$ is AST, and thus, terminates on every possible initial value with probability 1.

In addition, one is also interested in the expected number of steps it takes for the program to terminate.

Definition 2.3.29 – Expected Derivation Length (Random Walk Program).

*We define the expected derivation length* $\mathrm{edl}(\mu, x_0) \in \mathbb{R}_{\geq 0} \cup \{\infty\}$ *of a random walk program* $\mu$ *starting in* $x_0$ *as* $\mathrm{edl}(\mu, x_0) = \infty$ *if* $\mathbb{P}^{\mu}_{x_0}(\mathrm{Cyl}(\mathtt{SN})) < 1$*, and otherwise, as*

$$\mathrm{edl}(\mu, x_0) = \sum_{\pi \in \mathrm{Runs}_{\mathtt{SN}}(\mu)} (|\pi| - 1) \cdot \mathbb{P}^{\mu}_{x_0}(\mathrm{Cyl}(\pi)).$$

Note that a terminating run $\pi = s_0 \dots s_k$ has length $|\pi| = k + 1$, but $\pi$ consists of only $|\pi| - 1 = k$ transition steps. Therefore, we use $|\pi| - 1$ in the definition of the expected derivation length.

Definition 2.3.30 – Positive Almost-Sure Termination (Random Walk Program).

*A random walk program* $\mu$ *is positively almost surely terminating (*PAST*) if we have* $\mathrm{edl}(\mu, x_0) < \infty$ *for all* $x_0 \in \mathbb{Z}$.

Example 2.3.31 (Expected Derivation Length). Reconsider the random walk program $\mu = \{^{1/2}-1, ^{1/2}0\}$ from Example 2.3.23 starting at 1. The termination probability is 1, and therefore, we can compute the expected runtime by

$$\mathrm{edl}(\mu, 1) = 1 \cdot \mathbb{P}^{\mu}_{1}(\mathrm{Cyl}(10)) + 2 \cdot \mathbb{P}^{\mu}_{1}(\mathrm{Cyl}(110)) + \dots = 1/2 + 2 \cdot (1/2)^2 + 3 \cdot (1/2)^3 + \dots = 2.$$

So, in expectation we have to perform 2 steps before we terminate. Again, by Corollary 2.3.33, $\mu$ is PAST, and thus, has a finite expected derivation length for every possible initial value.

Depending on $\mu$ and its *expected value* $\mathbb{E}(\mu)$, we get the same characterization as before.

Definition 2.3.32 – Classification of Random Walk Programs.

*Let* $\mu$ *be a random walk program. We classify* $\mu$ *into the following four types.*

- *If* $\mu(0) = 1$*, then we say* $\mu$ *is a loop walk.*
- *If* $\mu(0) < 1$ *and* $\mathbb{E}(\mu) = 0$*, then* $\mu$ *is a symmetric random walk.*
- *If* $\mu(0) < 1$ *and* $\mathbb{E}(\mu) > 0$*, then* $\mu$ *is a positively drifting random walk.*
- *If* $\mu(0) < 1$ *and* $\mathbb{E}(\mu) < 0$*, then* $\mu$ *is a negatively drifting random walk.*

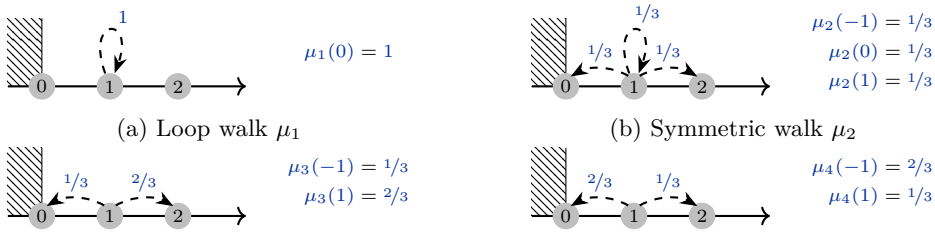
In Figure 2.2 one can see the examples from Figure 2.1 for all four different types, where we additionally state the corresponding random walk program.

Corollary 2.3.33 – AST and PAST of Random Walk Programs.

*Let* $\mu$ *be a random walk program.*

1. *If* $\mu$ *is a loop walk, then* $\mu$ *is neither* AST *nor* PAST.
2. *If* $\mu$ *is a symmetric random walk, then* $\mu$ *is* AST *but not* PAST.
3. *If* $\mu$ *is a positively drifting random walk, then* $\mu$ *is neither* AST *nor* PAST.
4. *If* $\mu$ *is a negatively drifting random walk, then* $\mu$ *is* AST *and* PAST.

*Proof.* The proof transfers the classical results on random walks [Spi01] to random walk programs. It is closely related to [GGH19], where AST and PAST of random walk programs with finite support are characterized in the same way. *Proof*




(a) Loop walk $\mu_1$ (b) Symmetric walk $\mu_2$

(c) Positive drift $\mu_3$ (d) Negative drift $\mu_4$

Figure 2.2: The four different random walk programs $\mu_1$, $\mu_2$, $\mu_3$, and $\mu_4$, corresponding to the random walks $(\mathbb{1}_1, \hat{Y}^{(1)})$, $(\mathbb{1}_1, \hat{Y}^{(2)})$, $(\mathbb{1}_1, \hat{Y}^{(3)})$, and $(\mathbb{1}_1, \hat{Y}^{(4)})$ from Figure 2.1, respectively.

Let $x_0 \in \mathbb{Z}$. W.l.o.g. let $x_0 > 0$, as otherwise the only run starting in $x_0$ is the terminating run $x_0$. Let $\hat{Y} = (Y_n)_{n \in \mathbb{N}_{>0}}$ be independent random variables on some probability space $(\Omega, \mathfrak{A}, \mathbb{P})$ with $\mathbb{P}(Y_n = x) = \mu(x)$ for all $n \in \mathbb{N}_{>0}$ and $x \in \mathbb{Z}$. Moreover, let $\hat{S} = (\mathbb{1}_{x_0}, \hat{Y})$ be the corresponding random walk and let $T = T_{\mathbb{Z}_{\leq 0}}$ be its hitting time for $\mathbb{Z}_{\leq 0}$. We map every outcome $\omega \in \Omega$ to $\Phi(\omega) = S_0(\omega)S_1(\omega)\dots S_{T(\omega)}(\omega)$ if $T(\omega) < \infty$, and to $\Phi(\omega) = S_0(\omega)S_1(\omega)\dots$ otherwise. The set of all $\omega \in \Omega$ where $\Phi(\omega)$ is not a run in $\mathrm{Runs}(\mu)$ has probability 0, and we map these outcomes to an arbitrary fixed run. For every finite run $\pi = s_0 \dots s_k \in \mathrm{Runs}_{\mathrm{fin}}(\mu)$ with $s_0 = x_0$, we have $\Phi(\omega) \in \mathrm{Cyl}(\pi)$ iff $S_i(\omega) = s_i$ for all $0 \leq i \leq k$, and thus, by independence, $\mathbb{P}(\Phi \in \mathrm{Cyl}(\pi)) = \prod_{i=0}^{k-1} \mathbb{P}(Y_{i+1} = s_{i+1} - s_i) = \prod_{i=0}^{k-1} \mu(s_{i+1} - s_i) = \mathbb{P}^{\mu}_{x_0}(\mathrm{Cyl}(\pi))$. For finite runs that do not start in $x_0$, both probabilities are 0. Hence, the image measure of $\mathbb{P}$ under $\Phi$ coincides with $\mathbb{P}^{\mu}_{x_0}$ on all cylinder sets, and by the uniqueness in Theorem 2.3.21, it is equal to $\mathbb{P}^{\mu}_{x_0}$. In particular, we have $\mathbb{P}^{\mu}_{x_0}(\mathrm{Cyl}(\mathtt{SN})) = \mathbb{P}(T < \infty)$, and since every terminating run $\pi = \Phi(\omega)$ consists of $|\pi| - 1 = T(\omega)$ steps, we obtain $\mathrm{edl}(\mu, x_0) = \mathbb{E}(T)$. So $\mu$ is `AST` (`PAST`) iff $\mathbb{P}(T = \infty) = 0$ ($\mathbb{E}(T) < \infty$) holds for all $x_0 > 0$. All in all, the claim follows from Corollary 2.3.14. *QED*

Example 2.3.34. Based on Corollary 2.3.33, $\mu_1$ and $\mu_3$ are neither `AST` nor `PAST`. The random walk program $\mu_2$ is `AST` but not `PAST`, and $\mu_4$ is both `AST` and `PAST`.

In this case, deciding whether a random walk program given by a distribution function $\mu$ is `AST` or `PAST` can be done by first computing the expected value of $\mu$ and comparing it with 0, and second, by checking whether there exists some $x < 0$ with $\mu(x) > 0$. Hence, `AST` and `PAST` are decidable for random walk programs with finite support and rational probabilities. Next, we introduce even more complex stochastic systems, where `AST` and `PAST` are still decidable (if the underlying transition system is finite and all probabilities are rational).

SECTION 2.4

## Markov Chains and Markov Decision Processes

In this section, we introduce Markov chains and Markov decision processes, i.e., certain forms of transition systems, where the transition relation is enriched with probabilistic choices. On the one hand, we use Markov chains and Markov decision processes to demonstrate the difficulties one has to take into account when defining probability spaces, termination probabilities, and expected runtimes for

complex stochastic systems. On the other hand, they serve as illustrative instances of probabilistic rewrite systems throughout this thesis.

Compared to random walks, we do not have the same distribution at every state of the system, and moreover, the state space is not a straight line anymore but can be an arbitrary transition system. To be precise, Markov chains are (potentially infinite) transition systems that have a probability distribution for every state describing their transition function. While they are useful as a statistical model for many real-world processes, they are not the appropriate model for randomized distributed systems since they have no notion of control and no non-determinism. Markov decision processes additionally allow for the combination of non-determinism and probabilistic choice. For further details, see [BK08, Put94], from which most of our definitions for this section are taken and where all results can be found with corresponding proofs.[3]

### 2.4.1 Markov Chains

We start by recapitulating the syntax of the system in question, namely a discrete-time Markov chain. Markov chains are ordinary transition systems where the non-deterministic choices for the successor relation are replaced with a distribution giving every possible successor a certain probability.

Definition 2.4.1 – Markov Chain.
*A Markov chain (MC) is a tuple $\mathcal{M} = (S, \mathcal{P})$, where*

- *$S$ is a countable, non-empty set of states, and*
- *$\mathcal{P} : S \times S \to [0, 1]$ is the transition function such that for all states $s \in S$ we have $\sum_{s' \in S} \mathcal{P}(s, s') = 1$.*

*We call $\mathcal{M}$ finite if $S$ is finite.*

The transition function $\mathcal{P}$ defines for each pair of states $(s, s')$ how likely it is to move from $s$ to $s'$. The probabilities over all targets $s' \in S$ for a fixed start $s$ sum up to 1, so that the function $\mathcal{P}(s, \cdot) : S \to [0, 1]$ defines a proper distribution over the state set $S$. By definition the transition probabilities only depend on the current state and not on the history of states that we have already visited in an execution. This is known as the "Markov property" or the "memoryless property".

We can view an MC either as a directed graph with transitions that have a corresponding label containing the probability of this transition, or simply as a *(column) stochastic matrix* that describes the transition probabilities, i.e., a matrix where the entries of each column sum up to 1.

Example 2.4.2. An example of a graphical visualization of an MC can be seen in Figure 2.3. Here, we can see the MC $\mathcal{M} = (S, \mathcal{P})$ with $S = \{s_0, s_1, s_2\}$ and the stochastic matrix $\mathbf{P} = (\mathbf{P}_{t,s})_{t,s \in S}$ shown in Figure 2.4, whose entry $\mathbf{P}_{t,s}$ in row $t$ and column $s$ is $\mathcal{P}(s, t)$. The distribution $\mathcal{P}(s_2, \cdot)$ is given via $s_0 \mapsto 0$, $s_1 \mapsto 2/3$, and $s_2 \mapsto 1/3$.

Example 2.4.3 (Simulating a Die by a Fair Coin). A classical problem in stochastic

[3] Because we mostly consider universal termination in this thesis, we have no initial state or initial configuration. Additionally, we do not need any atomic propositions for the states, as we only consider reachability constraints for a given set of states.

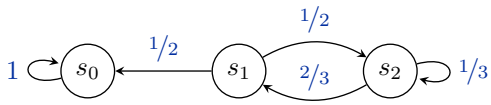



Figure 2.3: MC as a graph.

$$\mathbf{P} = \begin{pmatrix} 1 & 1/2 & 0 \\ 0 & 0 & 2/3 \\ 0 & 1/2 & 1/3 \end{pmatrix}$$

Figure 2.4: MC as a matrix.

systems is only having access to certain types of distributions. For example, assume we want to write a program for a board game that can throw a fair die according to the distribution presented in Example 2.1.8, but we only have access to a fair coin flip in our probabilistic programming language. The MC shown in Figure 2.5, as originally proposed by Knuth and Yao [KY76], only uses the probability $1/2$ in its transition distribution to model a fair die.

If we start in the state $s_{\text{init}}$, then the first coin flip decides whether the die rolls a number strictly greater than 3 or not. Afterwards, we still have three different options left and cannot directly split equally anymore. Instead, we have to add the two states $s_{\{1,2,3\}}$ and $s'_{\{1,2,3\}}$. From $s_{\{1,2,3\}}$ we can either move to state $s'_{\{1,2,3\}}$ or to the state $s_{\{2,3\}}$, where only $s_2$ and $s_3$ are reachable with equal probability. From $s'_{\{1,2,3\}}$ we can either directly move to $s_1$ or move back to $s_{\{1,2,3\}}$. Due to this loop in the graph, we may have infinite runs that never reach one of the states $s_1$–$s_6$. Nevertheless, we will almost surely, i.e., with probability 1, reach one of the six states $s_1$–$s_6$, no matter where we start in the transition system. Moreover, if we start at $s_{\text{init}}$, then the probability of reaching $s_i$ is $1/6$ for every $1 \leq i \leq 6$. In the remainder of this section, we explain how to formally establish that these properties indeed hold.

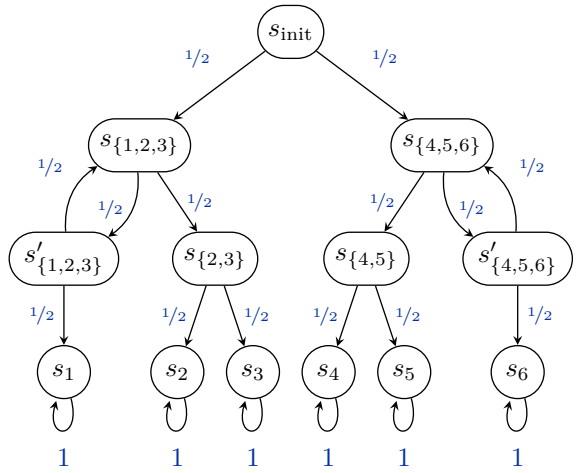



Figure 2.5: MC for simulating a die by a fair coin.

**Probability Space of a Markov Chain** We define the corresponding probability space of an MC as for random walk programs via the $\sigma$-field induced by the cylinder sets of finite runs within the MC.

Definition 2.4.4 – Run, Prefix Run, Cylinder Set (Markov Chain).

*A run in an MC $\mathcal{M}$ is an infinite sequence of states $s_0 s_1 \ldots \in S^\omega$ such that $\mathcal{P}(s_i, s_{i+1}) > 0$ for all $i \geq 0$. A finite run (or prefix run) in $\mathcal{M}$ is a finite sequence of states $s_0 \ldots s_k \in S^*$ such that $\mathcal{P}(s_i, s_{i+1}) > 0$ for all $0 \leq i \leq k-1$. The set of all runs in $\mathcal{M}$ is denoted by $\mathrm{Runs}(\mathcal{M})$, and the set of all finite runs in $\mathcal{M}$ is denoted by $\mathrm{Runs}_{\mathrm{fin}}(\mathcal{M})$. The cylinder set of $\pi = s_0 \ldots s_n \in \mathrm{Runs}_{\mathrm{fin}}(\mathcal{M})$ is defined as $\mathrm{Cyl}(\pi) = \{\pi' \in \mathrm{Runs}(\mathcal{M}) \mid \pi \in \mathrm{pref}(\pi')\}$.*

Example 2.4.5. An infinite run that starts with $s_{\mathrm{init}}$ in the MC $\mathcal{M}$ from Figure 2.5 would be $s_{\mathrm{init}} s_{\{1,2,3\}} s'_{\{1,2,3\}} s_{\{1,2,3\}} s'_{\{1,2,3\}} s_1 s_1 \ldots \in \mathrm{Runs}(\mathcal{M})$ and a finite run would be $s_{\mathrm{init}} s_{\{1,2,3\}} s_{\{2,3\}} s_2 \in \mathrm{Runs}_{\mathrm{fin}}(\mathcal{M})$.

Similar to random walks, we have to fix an *initial state* $s$ for the final probability space. Each such initial state $s \in S$ gives rise to a different probability space.

Definition 2.4.6 – Probability Space of a Markov Chain.

*Let $s \in S$ be an initial state and let $\mathfrak{A} = \langle \{\mathrm{Cyl}(\pi) \mid \pi \in \mathrm{Runs}_{\mathrm{fin}}(\mathcal{M})\} \rangle_\sigma$. By $\mathbb{P}_s^{\mathcal{M}}$ we denote the unique probability measure on $\mathfrak{A}$ from Theorem 2.3.21 for $X = S$ and the transition function $\mathcal{P}$, i.e., for every finite run $s_0 \ldots s_k \in \mathrm{Runs}_{\mathrm{fin}}(\mathcal{M})$, we have $\mathbb{P}_s^{\mathcal{M}}(\mathrm{Cyl}(s_0 \ldots s_k)) = 0$ if $s_0 \neq s$, and $\mathbb{P}_s^{\mathcal{M}}(\mathrm{Cyl}(s_0 \ldots s_k)) = \prod_{i=0}^{k-1} \mathcal{P}(s_i, s_{i+1})$ otherwise. Moreover, we also write $\mathbb{P}_s^{\mathcal{M}}(s_0 \ldots s_k) = \mathbb{P}_s^{\mathcal{M}}(\mathrm{Cyl}(s_0 \ldots s_k))$.*
*The probability space of the MC $\mathcal{M}$ starting in $s$ is given by $(\mathrm{Runs}(\mathcal{M}), \mathfrak{A}, \mathbb{P}_s^{\mathcal{M}})$.*

Example 2.4.7. Consider the MC $\mathcal{M}$ for throwing a die from Example 2.4.3. The finite run $s_{\mathrm{init}} s_{\{1,2,3\}} s_{\{2,3\}} s_2 \in \mathrm{Runs}_{\mathrm{fin}}(\mathcal{M})$ has the probability

$$\mathcal{P}(s_{\mathrm{init}}, s_{\{1,2,3\}}) \cdot \mathcal{P}(s_{\{1,2,3\}}, s_{\{2,3\}}) \cdot \mathcal{P}(s_{\{2,3\}}, s_2) = 1/2 \cdot 1/2 \cdot 1/2 = (1/2)^3 = 1/8$$

in the probability space $(\mathrm{Runs}(\mathcal{M}), \mathfrak{A}, \mathbb{P}_{s_{\mathrm{init}}}^{\mathcal{M}})$, where we start in $s_{\mathrm{init}}$. The cylinder set $\mathrm{Cyl}(s_{\mathrm{init}} s_{\{1,2,3\}} s_{\{2,3\}} s_2)$ is given by

$$\mathrm{Cyl}(s_{\mathrm{init}} s_{\{1,2,3\}} s_{\{2,3\}} s_2) = \{s_{\mathrm{init}} s_{\{1,2,3\}} s_{\{2,3\}} s_2^\omega\}.$$

**Notions of Termination for Markov Chains** Once again, we have a system that by definition always runs infinitely long. Hence, we cannot directly talk about termination of MCs. Instead, one considers how likely it is to reach a certain set of states, which we consider as the *goal states* or *terminal states*. Additionally, we are also interested in how fast we reach the terminal states in expectation. This is similar to random walks, where we run infinitely long, but consider termination once we "hit" the set $\mathbb{Z}_{\leq 0}$. But instead of hitting times, we consider *reachability probabilities* for a given set of states $F \subseteq S$ in the MC. In the literature, the set $F$ may also denote a set of *bad states* or *error states*, which one should avoid. Since this thesis only considers termination analysis and not, e.g., safety analysis, our goal states $F$ are always "good" and we want to reach them as fast as possible.

Definition 2.4.8 – Reachability Probabilities in a Markov Chain.

*Let $\mathcal{M}$ be an MC, let $s \in S$ be the initial state, and $F \subseteq S$ be a set of goal states. By $\mathrm{Runs}_{\mathrm{reach}}(\mathcal{M}, s, F)$ we denote the set of all paths starting in $s$ and ending in a state $s' \in F$ without seeing a state from $F$ beforehand, i.e., $\mathrm{Runs}_{\mathrm{reach}}(\mathcal{M}, s, F) = \{s_0 \ldots s_k \in \mathrm{Runs}_{\mathrm{fin}}(\mathcal{M}) \mid s_0 = s, s_k \in F, s_i \notin F$ for all $0 \leq$*

$i < k\}$. *The reachability probability of reaching a state $s' \in F$ from $s$ is defined by*

$$\begin{aligned}\mathbb{P}^{\mathcal{M}}_{\text{Reach}}(s, F) &= \sum_{s_0 \dots s_k \in \text{Runs}_{\text{reach}}(\mathcal{M}, s, F)} \mathbb{P}^{\mathcal{M}}_s(\text{Cyl}(s_0 \dots s_k)) \\ &= \sum_{s_0 \dots s_k \in \text{Runs}_{\text{reach}}(\mathcal{M}, s, F)} \mathbb{P}^{\mathcal{M}}_s(s_0 \dots s_k).\end{aligned}$$

Example 2.4.9. Consider the MC $\mathcal{M}$ for throwing a die from Example 2.4.3 with initial state $s_{\text{init}}$. The reachability probability of the state set $F = \{s_1\}$ is

$$\begin{aligned}\mathbb{P}^{\mathcal{M}}_{\text{Reach}}(s_{\text{init}}, \{s_1\}) &= \sum_{s_{\text{init}} \dots s_1 \in \text{Runs}_{\text{reach}}(\mathcal{M}, s_{\text{init}}, \{s_1\})} \mathbb{P}^{\mathcal{M}}_{s_{\text{init}}}(s_{\text{init}} \dots s_1) \\ &= \mathbb{P}^{\mathcal{M}}_{s_{\text{init}}}(s_{\text{init}} s_{\{1,2,3\}} s'_{\{1,2,3\}} s_1) \\ &\quad + \mathbb{P}^{\mathcal{M}}_{s_{\text{init}}}(s_{\text{init}} s_{\{1,2,3\}} s'_{\{1,2,3\}} s_{\{1,2,3\}} s'_{\{1,2,3\}} s_1) \\ &\quad + \mathbb{P}^{\mathcal{M}}_{s_{\text{init}}}(s_{\text{init}} s_{\{1,2,3\}} s'_{\{1,2,3\}} s_{\{1,2,3\}} s'_{\{1,2,3\}} s_{\{1,2,3\}} s'_{\{1,2,3\}} s_1) \\ &\quad + \dots \\ &= (1/2)^3 + (1/2)^5 + (1/2)^7 + \dots \\ &= (1/2)^3 \cdot \sum_{i=0}^{\infty} (1/2)^{2i} = 1/6.\end{aligned}$$

While it is possible in some cases to compute the reachability probability via its definition and the infinite sums directly, this becomes increasingly difficult the more different paths lead to the goal states. In the following, we explain a technique based on solving linear equations to compute such reachability probabilities.

First, note that if for a state $s \in S$ there exists no outgoing path that ends in a goal state $s' \in F$, then the reachability probability has to be 0. Here, by $\text{Pre}^*(F)$ we denote the set of all states that can reach a state in $F$, i.e., $s \in \text{Pre}^*(F)$ if $\text{Runs}_{\text{reach}}(\mathcal{M}, s, F) \neq \varnothing$. Another easy case is the reachability probability of a goal state itself, which can obviously reach itself with probability 1 by doing nothing. Lastly, for the other states, we can consider the definition via all the possible paths and try to unroll the definition once. For a state $s \in S \setminus F$ that is not a goal state, the probability of reaching the goal states is at least the probability of reaching the goal states in a single step, i.e., $\mathbb{P}^{\mathcal{M}}_{\text{Reach}}(s, F) \geq \mathcal{P}(s, F) = \sum_{t \in F} \mathcal{P}(s, t)$. The remaining probability mass can be described by all paths that require more than a single step to reach the goal states. But then, for each of those paths we are again in a state $s' \in S \setminus F$ after a single step and can use the same argumentation again.

Let $(x_s)_{s \in S} = (\mathbb{P}^{\mathcal{M}}_{\text{Reach}}(s, F))_{s \in S}$ be the vector of all reachability probabilities for all states. According to the investigation in the previous paragraph, we get the following linear equations that $(x_s)_{s \in S}$ has to satisfy. For all $s \in F$ we have $x_s = 1$, for all $s \notin \text{Pre}^*(F)$ we have $x_s = 0$, and for all remaining states $s \in \text{Pre}^*(F) \setminus F$ we have $x_s = \sum_{t \in \text{Supp}(\mathcal{P}(s,\cdot))} \mathcal{P}(s, t) \cdot x_t$. The solution of this equational system is unique for finite MCs. Nevertheless, we characterize the desired probability vector as the *least solution* in $[0, 1]^S$, since this is the form that carries over to expected runtimes (Theorem 2.4.16) and to MDPs (Theorem 2.4.33).

Theorem 2.4.10 – Computing Reachability Probabilities (Markov Chain).

Let $\mathcal{M} = (S, \mathcal{P})$ be a finite MC, and let $F \subseteq S$ be a set of goal states. The vector of all reachability probabilities $(x_s)_{s \in S} = (\mathbb{P}^{\mathcal{M}}_{\text{Reach}}(s, F))_{s \in S}$ is the least solution in $[0,1]^S$ satisfying

$$x_s = \begin{cases} 1, & \text{if } s \in F, \\ 0, & \text{if } s \notin \text{Pre}^*(F), \\ \sum_{t \in \text{Supp}(\mathcal{P}(s,\cdot))} \mathcal{P}(s,t) \cdot x_t, & \text{if } s \in \text{Pre}^*(F) \setminus F. \end{cases}$$

Example 2.4.11. Consider the MC $\mathcal{M}$ for throwing a die from Example 2.4.3. We want to compute the reachability probabilities $(x_s)_{s \in S} = (\mathbb{P}^{\mathcal{M}}_{\text{Reach}}(s, \{s_1\}))_{s \in S}$ of reaching $s_1$ for each state. We have $\text{Pre}^*(\{s_1\}) = \{s_1, s'_{\{1,2,3\}}, s_{\{1,2,3\}}, s_{\text{init}}\}$. For the state $s_1$ itself we have $x_{s_1} = 1$. For all states $s \in S \setminus \text{Pre}^*(\{s_1\})$ that cannot reach $s_1$, e.g., $s_2$ or $s_{\{4,5,6\}}$, we have $x_s = 0$. Finally, for the remaining states in $\text{Pre}^*(\{s_1\})$ we have to solve the three equations

$$\begin{aligned} x_{s_{\text{init}}} &= 1/2 \cdot x_{s_{\{1,2,3\}}} \\ x_{s_{\{1,2,3\}}} &= 1/2 \cdot x_{s'_{\{1,2,3\}}} \\ x_{s'_{\{1,2,3\}}} &= 1/2 \cdot x_{s_{\{1,2,3\}}} + 1/2. \end{aligned}$$

The (unique) solution of these equations is $(x_{s_{\text{init}}}, x_{s_{\{1,2,3\}}}, x_{s'_{\{1,2,3\}}}) = (1/6, 1/3, 2/3)$.

We say that the MC almost surely terminates if we can reach a given set of goal states almost surely, i.e., with probability 1, no matter what the initial state is.

Definition 2.4.12 – Almost-Sure Termination (Markov Chain).

*An MC $\mathcal{M} = (S, \mathcal{P})$ is almost surely terminating w.r.t. a set of goal states $F \subseteq S$ ($\texttt{AST}(F)$) if we have $\mathbb{P}^{\mathcal{M}}_{\text{Reach}}(s, F) = 1$ for all initial states $s \in S$.*

Example 2.4.13. Consider the MC $\mathcal{M}$ for throwing a die from Example 2.4.3. If we consider the states $s_1$–$s_6$ as our goal states, i.e., $F = \{s_1, s_2, s_3, s_4, s_5, s_6\}$, then $\mathcal{M}$ is $\texttt{AST}(F)$. However, if we only consider $F' = \{s_1\}$ as our goal states, then $\mathcal{M}$ is not $\texttt{AST}(F')$ anymore, since, e.g., $\mathbb{P}^{\mathcal{M}}_{\text{Reach}}(s_{\text{init}}, \{s_1\}) = 1/6 < 1$ and $\mathbb{P}^{\mathcal{M}}_{\text{Reach}}(s_2, \{s_1\}) = 0 < 1$.

Next, we consider the expected number of steps to reach the goal states and show that we can compute it via a system of linear equations as well.

Definition 2.4.14 – Expected Runtimes in a Markov Chain.

*Let $s \in S$ be the initial state and $F \subseteq S$ be a set of goal states. The expected number of steps to reach a state $s' \in F$ from $s$ is defined by $\text{ert}^{\mathcal{M}}(s, F) = \infty$ if $\mathbb{P}^{\mathcal{M}}_{\text{Reach}}(s, F) < 1$, and otherwise, by*

$$\begin{aligned} \text{ert}^{\mathcal{M}}(s, F) &= \sum_{s_0 \ldots s_k \in \text{Runs}_{\text{reach}}(\mathcal{M}, s, F)} k \cdot \mathbb{P}^{\mathcal{M}}_s(\text{Cyl}(s_0 \ldots s_k)) \\ &= \sum_{s_0 \ldots s_k \in \text{Runs}_{\text{reach}}(\mathcal{M}, s, F)} k \cdot \mathbb{P}^{\mathcal{M}}_s(s_0 \ldots s_k). \end{aligned}$$

Example 2.4.15. Consider the MC $\mathcal{M}$ for throwing a die from Example 2.4.3 with initial state $s_{\text{init}}$. The expected number of steps to reach the states $F = \{s_1, s_2, s_3, s_4, s_5, s_6\}$

is

$$\begin{aligned}\operatorname{ert}^{\mathcal{M}}(s_{\text{init}}, F) &= \sum_{s_{\text{init}}\dots s \in \operatorname{Runs}_{\text{reach}}(\mathcal{M}, s_{\text{init}}, F)} (|s_{\text{init}}\dots s| - 1) \cdot \mathbb{P}^{\mathcal{M}}_{s_{\text{init}}}(s_{\text{init}}\dots s) \\ &= 6 \cdot \left(3 \cdot (1/2)^3 + 5 \cdot (1/2)^5 + 7 \cdot (1/2)^7 + \cdots\right) \\ &= 6 \cdot \sum_{i=0}^{\infty} (3+2i) \cdot (1/2)^{3+2i} = \frac{11}{3}.\end{aligned}$$

Again, we can solve the problem of computing the expected number of steps to reach the goal states for each state by finding a solution to a set of linear equations. Let $(x_s)_{s\in S} = (\operatorname{ert}^{\mathcal{M}}(s,F))_{s\in S}$ be the vector of all expected runtimes. For all $s \in F$ we have $x_s = 0$, since there are no more steps to perform, and for all $s \notin \operatorname{Pre}^*(F)$ we have $x_s = \infty$, because we will never reach the goal states. All the remaining states $s \in \operatorname{Pre}^*(F) \setminus F$ satisfy $x_s = 1 + \sum_{t\in\operatorname{Supp}(\mathcal{P}(s,\cdot))} \mathcal{P}(s,t) \cdot x_t$, where we account for the step that we perform from the state $s$ to get to one of its successors. Note that this equation system may have several solutions in $(\overline{\mathbb{R}}_{\geq 0})^S$ (e.g., setting $x_s = \infty$ for all $s \in \operatorname{Pre}^*(F) \setminus F$ that are not forced to a finite value often satisfies the equations as well), but our desired vector containing the expected runtimes is again the least solution.

**Theorem 2.4.16** – Computing Expected Runtimes (Markov Chain).

Let $\mathcal{M} = (S, \mathcal{P})$ be a finite MC, and let $F \subseteq S$ be a set of goal states. The expected number of steps to reach a state $s' \in F$ is the least solution $(x_s)_{s\in S} = (\operatorname{ert}^{\mathcal{M}}(s,F))_{s\in S}$ in $(\overline{\mathbb{R}}_{\geq 0})^S$ of the linear equation system

$$x_s = \begin{cases} 0, & \text{if } s \in F, \\ \infty, & \text{if } s \notin \operatorname{Pre}^*(F), \\ 1 + \sum_{t\in\operatorname{Supp}(\mathcal{P}(s,\cdot))} \mathcal{P}(s,t) \cdot x_t, & \text{if } s \in \operatorname{Pre}^*(F) \setminus F. \end{cases}$$

Example 2.4.17. Consider the MC $\mathcal{M}$ for throwing a die from Example 2.4.3. We want to compute the expected runtimes $(x_s)_{s\in S} = (\operatorname{ert}^{\mathcal{M}}(s,F))_{s\in S}$ of reaching one of the states $F = \{s_1, s_2, s_3, s_4, s_5, s_6\}$ for each initial state. We have $\operatorname{Pre}^*(F) = S$, and thus, we have to solve the equations $x_{s_i} = 0$ for all $1 \leq i \leq 6$ and

$$\begin{aligned} x_{s_{\text{init}}} &= 1 + 1/2 \cdot x_{s_{\{1,2,3\}}} + 1/2 \cdot x_{s_{\{4,5,6\}}} \\ x_{s_{\{1,2,3\}}} &= 1 + 1/2 \cdot x_{s'_{\{1,2,3\}}} + 1/2 \cdot x_{s_{\{2,3\}}} \\ x_{s'_{\{1,2,3\}}} &= 1 + 1/2 \cdot x_{s_{\{1,2,3\}}} + 1/2 \cdot x_{s_1} \\ x_{s_{\{2,3\}}} &= 1 + 1/2 \cdot x_{s_2} + 1/2 \cdot x_{s_3} \\ x_{s_{\{4,5,6\}}} &= 1 + 1/2 \cdot x_{s'_{\{4,5,6\}}} + 1/2 \cdot x_{s_{\{4,5\}}} \\ x_{s'_{\{4,5,6\}}} &= 1 + 1/2 \cdot x_{s_{\{4,5,6\}}} + 1/2 \cdot x_{s_6} \\ x_{s_{\{4,5\}}} &= 1 + 1/2 \cdot x_{s_4} + 1/2 \cdot x_{s_5}. \end{aligned}$$

The least solution of these equations satisfies, e.g., $x_{s_{\text{init}}} = 11/3$, $x_{s_{\{1,2,3\}}} = 8/3$, and $x_{s_{\{2,3\}}} = 1$.

We say that the MC positively almost surely terminates if we can reach a given set of goal states in finitely many steps in expectation from every initial state.

Definition 2.4.18 – Positive Almost-Sure Termination (Markov Chain).

*An MC $\mathcal{M} = (S, \mathcal{P})$ is positively almost surely terminating w.r.t. a set of goal states $F \subseteq S$ ($\texttt{PAST}(F)$) if we have $\mathrm{ert}^{\mathcal{M}}(s, F) < \infty$ for all initial states $s \in S$.*

Example 2.4.19. Consider the MC $\mathcal{M}$ for throwing a die from Example 2.4.3. If we consider the states $s_1$–$s_6$ as our goal states, i.e., $F = \{s_1, s_2, s_3, s_4, s_5, s_6\}$, then $\mathcal{M}$ is $\texttt{PAST}(F)$. However, if we only consider $F' = \{s_1\}$ as our goal states, then $\mathcal{M}$ is not $\texttt{PAST}(F')$ anymore, since, e.g., $\mathrm{ert}^{\mathcal{M}}(s_2, \{s_1\}) = \infty$.

Thus, $\texttt{AST}(F)$ and $\texttt{PAST}(F)$ are decidable for finite MCs with rational transition probabilities, since the least solutions of the linear equation systems in Theorems 2.4.10 and 2.4.16 can be computed exactly [BK08].

### 2.4.2 Markov Decision Processes

In Markov chains we have no notion of *external control*. The system always behaves and evolves exactly as the MC describes. *Markov decision processes* allow for an additional non-determinism, i.e., multiple transition probabilities for the same state within the system, that can be controlled by some external source. The problem of termination no longer depends only on a deterministic system, but on the resolution of the occurring non-determinisms as well. A *scheduler* precisely describes how to resolve all occurring non-determinism in an execution within the Markov decision process. In this thesis, we only consider *demonic* non-determinism and always assume that there is an evil adversary that tries to resolve the non-determinisms in the worst possible way. This means that we have to compute the minimal termination probability (or a lower bound on the termination probabilities for all possible schedulers) and the maximal expected runtime for all possible schedulers (or an upper bound on the expected runtimes for all possible schedulers).

We start by recapitulating the basic syntax of a *Markov decision process*, which is an MC that has an additional set of *actions* for each state, and for each such action we have a corresponding transition distribution.

Definition 2.4.20 – Markov Decision Process.

*A Markov decision process (MDP) is a tuple $\mathcal{M} = (S, \mathrm{Act}, \mathcal{P})$, where*

- *$S$ is a finite, non-empty set of states,*
- *$\mathrm{Act}$ is a finite, non-empty set of actions, and*
- *$\mathcal{P} : S \times \mathrm{Act} \times S \to [0, 1]$ is the transition function such that for all states $s \in S$ and actions $\alpha \in \mathrm{Act}$ we have $\sum_{s' \in S} \mathcal{P}(s, \alpha, s') \in \{0, 1\}$.*

*An action $\alpha$ is enabled in state $s \in S$ if we have $\sum_{s' \in S} \mathcal{P}(s, \alpha, s') = 1$. By $\mathrm{Act}(s)$ we denote the set of all enabled actions in state $s \in S$. We require that every state has at least one enabled action, i.e., $\mathrm{Act}(s) \neq \varnothing$ for all $s \in S$.*

Every finite MC can be seen as an MDP where we only have a single action, i.e., $|\mathrm{Act}| = 1$.

Example 2.4.21. An example of a graphical visualization of a Markov decision process can be seen in Figure 2.6. Here, we have an MDP $\mathcal{M} = (S, \mathrm{Act}, \mathcal{P})$ with $S = \{s_0, s_1, s_2\}$ and $\mathrm{Act} = \{\alpha, \beta\}$. The distribution $\mathcal{P}(s_2, \alpha, \cdot)$ is given via $s_0 \mapsto 0$, $s_1 \mapsto 2/3$, and

$s_2 \mapsto 1/3$. The distribution $\mathcal{P}(s_2, \beta, \cdot)$ is given via $s_0 \mapsto 0$, $s_1 \mapsto 1/2$, and $s_2 \mapsto 1/2$. If we remove the action $\beta$ from Act, then the resulting MDP only has a single action and can be seen as an MC. In fact, in this way we get the MC from Example 2.4.2.

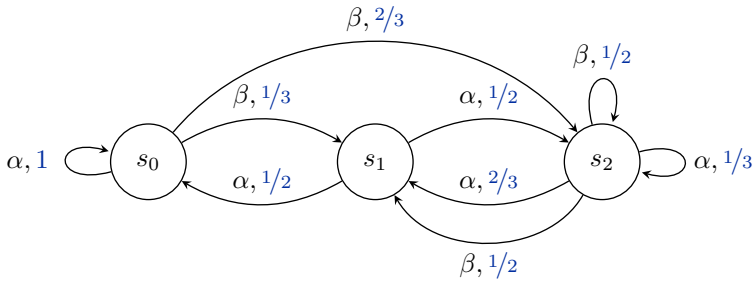



Figure 2.6: Markov decision process as a graph.

**Probability Space of a Markov Decision Process** Not only does every MDP with $|\mathrm{Act}(s)| = 1$ for every $s \in S$ correspond to an MC, but every MDP gives rise to infinitely many MCs depending on how we resolve the occurring non-determinisms that arise due to the different enabled actions at each state. A *scheduler* describes how to resolve all such non-determinisms. But before we can define schedulers and the corresponding probability space of MDPs, we have to define the notion of runs again that now incorporate actions between two states.

Definition 2.4.22 – Run, Prefix Run (MDP).

*A run in an MDP $\mathcal{M}$ is an infinite sequence of states and actions $s_0\alpha_0 s_1\alpha_1 \ldots \in (S \cdot \mathrm{Act})^\omega$ such that $\mathcal{P}(s_i, \alpha_i, s_{i+1}) > 0$ for all $i$, and a finite run (or prefix run) is a finite sequence $s_0\alpha_0 \ldots \alpha_{k-1}s_k \in (S \cdot \mathrm{Act})^* \cdot S$ such that $\mathcal{P}(s_i, \alpha_i, s_{i+1}) > 0$ for all $0 \leq i \leq k-1$. As for MCs, the set of all runs in $\mathcal{M}$ is denoted by $\mathrm{Runs}(\mathcal{M})$, and the set of all finite runs in $\mathcal{M}$ is denoted by $\mathrm{Runs}_{\mathrm{fin}}(\mathcal{M})$.*

Example 2.4.23. An infinite run that starts with $s_0$ in the MDP $\mathcal{M}$ from Figure 2.6 would be $s_0\beta s_2\alpha s_1\alpha s_0\beta s_2 \ldots \in \mathrm{Runs}(\mathcal{M})$ and a finite run would be $s_0\beta s_2\beta s_2 \in \mathrm{Runs}_{\mathrm{fin}}(\mathcal{M})$.

Definition 2.4.24 – Scheduler.

*Let $\mathcal{M} = (S, \mathrm{Act}, \mathcal{P})$ be an MDP. A scheduler for $\mathcal{M}$ is a function $\mathfrak{S} : S^+ \to \mathrm{Act}$ such that $\mathfrak{S}(s_0 \ldots s_k) \in \mathrm{Act}(s_k)$ for all $s_0 \ldots s_k \in S^+$.*

Example 2.4.25. A scheduler for the MDP $\mathcal{M}$ from Figure 2.6 would be $\mathfrak{S}(t_0 \ldots t_k) = \beta$ if $k \geq 1$ and $t_{k-1} = t_k = s_2$, and $\mathfrak{S}(t_0 \ldots t_k) = \alpha$ otherwise. This scheduler always chooses the action $\alpha$ for all histories, except when we visited $s_2$ at least twice directly after one another.

The scheduler that we can see in Example 2.4.25 only has to remember the last two states for every execution to choose its action. If the scheduler only depends on the current state and can completely ignore the history of the execution, then we have the Markov property once again. Such schedulers are called *memoryless schedulers*.

Definition 2.4.26 – Memoryless Scheduler.

*Let $\mathcal{M} = (S, \mathrm{Act}, \mathcal{P})$ be an MDP. A memoryless scheduler for $\mathcal{M}$ is a scheduler for $\mathcal{M}$ such that $\mathfrak{S}(s_0 \dots s_k) = \mathfrak{S}(s'_0 \dots s'_j)$ whenever $s_k = s'_j$ with $k, j \in \mathbb{N}$.*

Example 2.4.27. A memoryless scheduler for the MDP $\mathcal{M}$ from Figure 2.6 would be $\mathfrak{S}(t_0 \dots t_k) = \alpha$ for all $t_0 \dots t_k \in S^+$.

The probability space of a Markov decision process is defined via the infinite Markov chains induced by every possible scheduler. To this end, we use MCs from Definition 2.4.1 with the countably infinite set of states $S^+$.

Definition 2.4.28 – Probability Space of a Markov Decision Process.

*Let $\mathcal{M} = (S, \mathrm{Act}, \mathcal{P})$ be an MDP and $\mathfrak{S}$ be a scheduler for $\mathcal{M}$. The Markov chain $\mathcal{M}_{\mathfrak{S}}$ is given by $\mathcal{M}_{\mathfrak{S}} = (S^+, \mathcal{P}_{\mathfrak{S}})$ with $\mathcal{P}_{\mathfrak{S}}(s_1 \dots s_k, s_1 \dots s_k s_{k+1}) = \mathcal{P}(s_k, \mathfrak{S}(s_1 \dots s_k), s_{k+1})$, and $\mathcal{P}_{\mathfrak{S}}(\pi, \pi') = 0$ for all other pairs $\pi, \pi' \in S^+$. The probability space of $\mathcal{M}$ under $\mathfrak{S}$ starting in $s \in S$ is the probability space $(\mathrm{Runs}(\mathcal{M}_{\mathfrak{S}}), \mathfrak{A}, \mathbb{P}_s^{\mathcal{M}_{\mathfrak{S}}})$ of the MC $\mathcal{M}_{\mathfrak{S}}$ from Definition 2.4.6.*

Example 2.4.29. The MC $\mathcal{M}_{\mathfrak{S}} = (S^+, \mathcal{P}_{\mathfrak{S}})$ for the memoryless scheduler $\mathfrak{S}$ from Example 2.4.27 is not the same as the MC depicted in Figure 2.3, because we have an infinite state space in $(S^+, \mathcal{P}_{\mathfrak{S}})$, while the MC depicted in Figure 2.3 is finite. Nevertheless, the infinite MC is repetitive and behaves like the finite MC depicted in Figure 2.3.

**Notions of Termination for Markov Decision Processes** When it comes to the termination probability of an MDP, we have to decide on how to deal with all the possible schedulers. This mostly depends on whether we consider hitting the set of goal states to be "good" or "bad". In our case, we want to analyze termination, which means that we want the system to reach the goal states as fast as possible. One can imagine an adversary that tries to resolve the non-determinisms in the worst possible way, i.e., such that the termination probability is minimal. Therefore, the reachability probability of an MDP refers to the *minimal* reachability probability over all possible schedulers. If we instead tried to avoid reaching the goal states, e.g., when trying to prove that a bad state is not likely to be reached, then we would consider the maximum over all possible schedulers. In this case, the adversary would choose a scheduler that maximizes the probability to reach the bad states.

Definition 2.4.30 – Reachability Probability in MDPs.

*Let $s \in S$ be the initial state and $F \subseteq S$ be a set of goal states. The minimal reachability probability of reaching a state $s' \in F$ from $s$ over all schedulers is defined by $\mathbb{P}^{\mathcal{M}}_{\mathrm{MinReach}}(s, F) = \inf_{\mathfrak{S}} \mathbb{P}^{\mathcal{M}_{\mathfrak{S}}}_{\mathrm{Reach}}(s, F')$, where we use the adjusted goal states $F' = \{s_0 \dots s_k \in S^+ \mid s_k \in F\}$.*

Note that $F \subseteq S$ is a set of states, while the new set of goal states $F' \subseteq S^+$ is a set of non-empty sequences of states.

An important result regarding such simple reachability probabilities is the fact that the infimum $\inf_{\mathfrak{S}} \mathbb{P}^{\mathcal{M}_{\mathfrak{S}}}_{\mathrm{Reach}}(s, F')$ is attained by some scheduler, so the infimum can be replaced by a minimum. Moreover, we do not have to consider all (infinitely many) possible schedulers, because the minimum is attained by a memoryless scheduler.

Lemma 2.4.31 – Reachability Probabilities in MDPs via Memoryless Schedulers.

*Let $s \in S$ be the initial state and $F \subseteq S$ be a set of goal states. There exists a memoryless scheduler $\mathfrak{S}$ such that $\mathbb{P}^{\mathcal{M}}_{\text{MinReach}}(s, F) = \mathbb{P}^{\mathcal{M}_{\mathfrak{S}}}_{\text{Reach}}(s, F')$, where $F' = \{s_0 \dots s_k \in S^+ \mid s_k \in F\}$.*

Example 2.4.32. Consider the MDP $\mathcal{M}$ from Figure 2.6, and let the set of goal states be $F = \{s_1\}$. Then, the scheduler that minimizes the reachability probability from $s_0$ chooses action $\alpha$ instead of $\beta$, because in that way, we can never leave the state $s_0$. Therefore, we have a reachability probability of 0, i.e., $\mathbb{P}^{\mathcal{M}}_{\text{MinReach}}(s_0, F) = 0$.

If we use $F = \{s_0\}$, then we have $\mathbb{P}^{\mathcal{M}}_{\text{MinReach}}(s_i, F) = 1$ for all $0 \leq i \leq 2$, and no scheduler can prevent reaching the goal set with probability 1.

Since it suffices to consider only memoryless schedulers, we can adapt the constraints on the reachability probabilities by simply adding a minimum around the possible steps. This yields the following theorem. Here, $\text{Pre}^*(F)$ denotes the set of all states $s \in S$ such that there exists a finite run in $\mathcal{M}$ from $s$ to a state in $F$.

Theorem 2.4.33 – Computing Reachability Probabilities (MDPs).

Let $\mathcal{M} = (S, \text{Act}, \mathcal{P})$ be an MDP, and let $F \subseteq S$ be a set of goal states. The vector $(x_s)_{s \in S}$ with $x_s = \mathbb{P}^{\mathcal{M}}_{\text{MinReach}}(s, F)$ is the least solution in $[0,1]^S$ of the following equation system

$$x_s = \begin{cases} 1, & \text{if } s \in F, \\ 0, & \text{if } s \in S \setminus \text{Pre}^*(F), \\ \min\{\sum_{t \in \text{Supp}(\mathcal{P}(s,\alpha,\cdot))} \mathcal{P}(s,\alpha,t) \cdot x_t \mid \alpha \in \text{Act}(s)\}, & \text{otherwise.} \end{cases}$$

Example 2.4.34. Consider the MDP $\mathcal{M}$ from Figure 2.6, and let the set of goal states be $F = \{s_0\}$. We want to compute the reachability probabilities $(x_s)_{s \in S} = (\mathbb{P}^{\mathcal{M}}_{\text{MinReach}}(s, \{s_0\}))_{s \in S}$ of reaching $s_0$ for each state. We have $\text{Pre}^*(\{s_0\}) = S$. Theorem 2.4.33 yields the equations $x_{s_0} = 1$, and

$$x_{s_1} = 1/2 \cdot x_{s_2} + 1/2$$
$$x_{s_2} = \min\{2/3 \cdot x_{s_1} + 1/3 \cdot x_{s_2}, 1/2 \cdot x_{s_1} + 1/2 \cdot x_{s_2}\}.$$

The least solution of these equations is $(x_{s_0}, x_{s_1}, x_{s_2}) = (1, 1, 1)$.

For the runtime, we are interested in the maximal expected number of steps we have to perform before we terminate.

Definition 2.4.35 – Expected Runtimes in an MDP.

*Let $s \in S$ be the initial state and $F \subseteq S$ be a set of goal states. The maximal expected number of steps to reach a state $s' \in F$ from $s$ is defined by $\text{ert}^{\mathcal{M}}(s, F) = \sup_{\mathfrak{S}} \text{ert}^{\mathcal{M}_{\mathfrak{S}}}(s, F')$, where $F' = \{s_0 \dots s_k \in S^+ \mid s_k \in F\}$.*

Again, it suffices to only consider memoryless schedulers.

Lemma 2.4.36 – Expected Runtime in MDPs via Memoryless Schedulers.

*Let $s \in S$ be the initial state and $F \subseteq S$ be a set of goal states. There exists a memoryless scheduler $\mathfrak{S}$ such that $\text{ert}^{\mathcal{M}}(s, F) = \text{ert}^{\mathcal{M}_{\mathfrak{S}}}(s, F')$, where $F' = \{s_0 \dots s_k \in S^+ \mid s_k \in F\}$.*

Example 2.4.37. Consider the MDP $\mathcal{M}$ from Figure 2.6, and let the set of goal states be $F = \{s_0\}$. The scheduler that maximizes the expected runtime from $s_2$ chooses action $\beta$ instead of $\alpha$, because then, we leave the state $s_2$ more slowly.

Computing the expected runtimes in an MDP can once again be done by solving a system of equations. Instead of minimizing, we maximize over all possible schedulers, because a higher runtime is considered worse (and we still consider demonic nondeterminism).

Theorem 2.4.38 – Computing Expected Runtimes (MDP).

Let $\mathcal{M} = (S, \mathrm{Act}, \mathcal{P})$ be an MDP, and let $F \subseteq S$ be a set of goal states. The expected number of steps $(x_s)_{s\in S} = (\mathrm{ert}^{\mathcal{M}}(s, F))_{s\in S}$ to reach a state $s' \in F$ is the least solution in $(\overline{\mathbb{R}}_{\geq 0})^S$ of the equation system

$$x_s = \begin{cases} 0, & \text{if } s \in F, \\ \infty, & \text{if } s \notin \mathrm{Pre}^*(F), \\ 1 + \max\left\{\sum_{t\in \mathrm{Supp}(\mathcal{P}(s,\alpha,\cdot))} \mathcal{P}(s,\alpha,t) \cdot x_t \mid \alpha \in \mathrm{Act}(s)\right\}, & \text{if } s \in \mathrm{Pre}^*(F) \setminus F. \end{cases}$$

Example 2.4.39. Consider the MDP $\mathcal{M}$ from Figure 2.6, and let the set of goal states be $F = \{s_0\}$. We want to compute the expected runtimes $(x_s)_{s\in S} = (\mathrm{ert}^{\mathcal{M}}(s, F))_{s\in S}$ of reaching $s_0$ for each state. Theorem 2.4.38 yields the equations $x_{s_0} = 0$, and

$$\begin{aligned} x_{s_1} &= 1 + 1/2 \cdot x_{s_2} \\ x_{s_2} &= 1 + \max\{2/3 \cdot x_{s_1} + 1/3 \cdot x_{s_2}, 1/2 \cdot x_{s_1} + 1/2 \cdot x_{s_2}\}. \end{aligned}$$

As explained in Example 2.4.37, the maximum is attained by the second argument here. The least solution of these equations is $(x_{s_0}, x_{s_1}, x_{s_2}) = (0, 4, 6)$.

Finally, we lift almost-sure termination and positive almost-sure termination to MDPs, where we have to consider all schedulers.

Definition 2.4.40 – (Positive) Almost-Sure Termination (MDP).

*Let $\mathcal{M} = (S, \mathrm{Act}, \mathcal{P})$ be an MDP, let $F \subseteq S$ be a set of goal states, and let $F' = \{s_0 \dots s_k \in S^+ \mid s_k \in F\}$. $\mathcal{M}$ is almost surely terminating w.r.t. $F$ ($\mathtt{AST}(F)$) if we have $\mathbb{P}^{\mathcal{M}}_{\mathrm{MinReach}}(s, F) = 1$ for all initial states $s \in S$. $\mathcal{M}$ is positively almost surely terminating w.r.t. $F$ ($\mathtt{PAST}(F)$) if we have $\mathrm{ert}^{\mathcal{M}_{\mathfrak{S}}}(s, F') < \infty$ for all schedulers $\mathfrak{S}$ and all initial states $s \in S$.*

By Lemma 2.4.36, an MDP $\mathcal{M}$ is $\mathtt{PAST}(F)$ iff $\mathrm{ert}^{\mathcal{M}}(s, F) < \infty$ holds for all $s \in S$, since the supremum over all schedulers is attained by a memoryless scheduler. Thus, $\mathtt{AST}(F)$ and $\mathtt{PAST}(F)$ are also decidable for MDPs with rational transition probabilities, since the least solutions of the equation systems in Theorems 2.4.33 and 2.4.38 can be computed exactly via linear programming [BK08, Put94].

# 3 Probabilistic Term Rewriting

In this chapter, we introduce all necessary notions and notation regarding term rewriting and its probabilistic extension. This chapter forms the backbone of all following chapters as it contains all the important definitions about the programming formalism of interest. In order to understand probabilistic term rewriting and all its aspects, we present four different formalisms:

1. **Abstract Reduction System (ARS)**. An ARS $(A, \to)$ consists of a countable universe $A$ and a binary reduction relation $\to \subseteq A \times A$. It provides the most basic formalism for studying abstract properties of rewrite systems such as termination, without imposing any structure on the objects being transformed. For example, the relation $\to_1 = \{n + 1 \to n \mid n \in \mathbb{N}\}$ on the natural numbers defines a terminating ARS in which every element eventually reduces to 0.

2. **Term Rewrite System (TRS)**. A TRS is an ARS whose universe is the set of first-order terms over some signature $\Sigma$. Reduction is governed by a set of rewrite rules $\ell \to r$, where $\ell$ (the left-hand side) is a pattern to be replaced and $r$ (the right-hand side) is the replacement. For example, the two rules $\mathsf{plus}(\mathsf{0}, y) \to y$ and $\mathsf{plus}(\mathsf{s}(x), y) \to \mathsf{s}(\mathsf{plus}(x, y))$ define addition in Peano arithmetic and reduce $\mathsf{plus}(\mathsf{s}(\mathsf{0}), \mathsf{s}(\mathsf{0}))$ to $\mathsf{s}(\mathsf{s}(\mathsf{0}))$ in two steps.

3. **Probabilistic Abstract Reduction System (PARS)**. A PARS extends an ARS by replacing each right-hand side with a finite multi-distribution over $A$, i.e., the reduction relation becomes $\to \subseteq A \times \mathrm{FDist}(A)$. Rule selection remains non-deterministic, but once a rule is applied the outcome is drawn from the associated probability distribution. For example, a symmetric random walk on $\mathbb{Z}$ defined by $n \to \{{}^{1}/_{2}\, n + 1, {}^{1}/_{2}\, n - 1\}$ for $n \geq 1$ forms an almost surely terminating PARS.

4. **Probabilistic Term Rewrite System (PTRS)**. A PTRS lifts TRSs to the probabilistic setting, just as PARSs lift ARSs. Its rules take the form $\ell \to \{{}^{p_1}r_1, \ldots, {}^{p_k}r_k\}$, where the left-hand side $\ell$ is matched against a subterm, which is then replaced by the corresponding instance of $r_i$ with probability $p_i$. For example, the rule $\mathsf{s}(x) \to \{{}^{1}/_{2}\mathsf{s}(\mathsf{s}(x)), {}^{1}/_{2}x\}$ once again describes a symmetric random walk that either increments or decrements the successor of $x$ with equal probability.

We have already seen in the previous chapter that there are several notions of "termination" in the probabilistic setting (see Sections 2.3 and 2.4). We can investigate the probability to terminate (almost-sure termination), or we can investigate the expected runtime (positive almost-sure termination). There is even an additional notion, called strong almost-sure termination, that we present in this chapter when defining the different notions for probabilistic programming languages that use both probabilistic choices and non-deterministic choices.

All in all, this chapter introduces the essential notions and notation for (probabilistic) term rewriting that are used throughout the rest of this thesis. It establishes the formal foundations for probabilistic abstract reduction systems, probabilistic term rewrite systems, and the different concepts of termination and runtime in the probabilistic setting that we want to analyze in this thesis.

### Contributions

We present the following contributions in this chapter. The results marked with !⋆! are novel, while the remaining contributions can be found in the following peer-reviewed publications: ⋆[1] [KG25], ⋆[2] [KSG25], and ⋆[3] [KG23, KG26].

- ⋆[1] Presentation of the concepts of (probabilistic) abstract reduction systems and (probabilistic) term rewrite systems in the context of termination analysis by means of examples.
- !⋆! Unified treatment of three different ways to formally describe stochastic reductions (based on rewriting distributions, trees, and stochastic processes), and establishment of their equivalence for all three notions of termination (almost-sure termination, positive almost-sure termination, and strong almost-sure termination) (Section 3.3.1). Parts of these results have already been shown in [ADLY20].
- ⋆[1] Proof of equivalence of positive almost-sure termination and strong almost-sure termination for most probabilistic term rewrite systems in practice (Section 3.4.4).
- ⋆[3] Proof technique for almost-sure termination of PTRSs based on algebraic interpretations of the terms (Section 3.4.5). The notion of collapsibility used for this technique in [KG26] is corrected in this thesis.
- ⋆[2] Derivation of upper bounds on the expected derivational and runtime complexity via proofs of strong almost-sure termination (Section 3.4.5). The technique to prove strong almost-sure termination itself was already developed in [ADLY20].

## Structure

We first formally recapitulate the notion of an abstract reduction system in Section 3.1. Then, we present term rewriting as a specific instance of an abstract reduction system in Section 3.2. Afterwards, we integrate probabilistic choices within the system after the rule selection. We first introduce probabilistic abstract reduction systems in Section 3.3, and finally, we present probabilistic term rewriting in Section 3.4. We end this chapter

by discussing related work and giving important open problems for future work in Section 3.5.

SECTION 3.1

## Abstract Reduction Systems

In this first section, we formally introduce *abstract reduction systems* (ARSs) together with all important properties of an ARS. See, e.g., [BN98, BKd03, Ohl02a] for a thorough introduction to rewriting in general and further details. An ARS is a binary relation $\to \subseteq A \times A$ on some countable[1] set $A$.

Definition 3.1.1 – Abstract Reduction System.

*An abstract reduction system (ARS) $(A, \to)$ is a countable set $A$ (called the universe) together with a binary relation $\to \subseteq A \times A$. The relation $\to$ is called a reduction relation. We often write $\to$ for the ARS $(A, \to)$. Additionally, if the relation has an index as in $\to_\alpha$, then we sometimes refer to $\to_\alpha$ simply as $\alpha$.*

At its core, rewriting is the systematic transformation of one object into another. In general, these objects can be anything from strings to higher-order lambda terms or plain integers. In the following sections we will consider specific instances of ARSs where the universe contains first-order terms, while in this section we introduce all basic definitions that can be used for every ARS without fixing any specific universe.

Example 3.1.2 (Abstract Reduction System). For the remainder of this section, we will consider the following ARSs on $\mathbb{N}$ in our examples.

- $\to_1 = \{n + 1 \to n \mid n \in \mathbb{N}\}$,
- $\to_2 = \{n \to m \mid n, m \in \mathbb{N}, m|n\}$ (i.e., $n \to_2 m$ if $m$ is a divisor of $n$), and
- $\to_3 = \{n \to 2n \mid n \in \mathbb{N}, 2|n\} \cup \{n \to 3n \mid n \in \mathbb{N}, 3|n\} \cup \{1 \to 2, 1 \to 3\}$.

If we compare ARSs to MCs and MDPs from Section 2.4, then the universe $A$ of an ARS forms the (possibly infinite) state space, while the reduction relation $\to$ indicates the possible transitions within the state space. We can combine multiple reduction steps to form *reduction sequences*, i.e., paths in the state space.

Definition 3.1.3 – Reduction.

*Let $(A, \to)$ be an ARS.*

a) *If $(a, b) \in \to$, then we write $a \to b$ and say that $b$ is a direct successor (or a reduct) of $a$.*

b) *A $\to$-reduction sequence (or $\to$-reduction) is a finite or infinite sequence $a_0 \to a_1 \to a_2 \to \cdots$.*

Example 3.1.4 (Reduction). For the ARSs from Example 3.1.2 we have the finite reductions $3 \to_1 2 \to_1 1 \to_1 0$ and $8 \to_2 4 \to_2 2 \to_2 1$, and we have the infinite reductions $8 \to_2 8 \to_2 8 \to_2 \cdots$ and $2 \to_3 4 \to_3 8 \to_3 16 \to_3 \cdots$.

[1] While some literature also investigates uncountable universes, we restrict ourselves to countable ones, as in term rewriting one uses the set of all first-order terms as the countable universe.

Next, we introduce several important *closures* of a given binary relation, as well as its *inverse*. The closure of a relation $\to$ for some property $P$ is the smallest relation $\to'$ that contains the given relation as a subset, i.e., $\to \subseteq \to'$, and that satisfies the given property $P$.

Definition 3.1.5 – Closures.
*Let $(A, \to)$ be an ARS. For any $n \in \mathbb{N}$, we define $\to^n$ as $\to^0 = \{(a, a) \mid a \in A\}$ and $\to^{n+1} = \to^n \circ \to$, where "$\circ$" denotes composition of relations, i.e., if $R_1, R_2 \subseteq A \times A$, then $R_1 \circ R_2 = \{(a, c) \mid \exists b \in A.\ (a, b) \in R_1, (b, c) \in R_2\}$.*

- $\to^= = \to^0 \cup \to^1$ *is the reflexive closure of* $\to$,
- $\to^+ = \bigcup_{n \in \mathbb{N}_{>0}} \to^n$ *is the transitive closure of* $\to$,
- $\to^* = \bigcup_{n \in \mathbb{N}} \to^n$ *is the transitive, reflexive closure of* $\to$,
- $\leftrightarrow = \{(a, b) \mid a \to b \text{ or } b \to a\}$ *is the symmetric closure of* $\to$, *and*
- $\leftarrow = \{(b, a) \mid a \to b\}$ *is the inverse of* $\to$.

Example 3.1.6 (Closures). For the ARSs from Example 3.1.2 we have $3 \to_1^* 0$ and $3 \to_1^+ 0$ due to the reduction $3 \to_1 2 \to_1 1 \to_1 0$. Additionally, we have $3 \to_3^= 3$, $2 \leftrightarrow_3^* 3$, and $3 \,{}^*\!\leftarrow_3 3$.

In contrast to MCs and MDPs, an ARS may have terminal states, i.e., objects $a \in A$ such that there is no $b \in A$ with $a \to b$. Such objects are called *normal forms*.

Definition 3.1.7 – Normal Form, Termination.
*Let $(A, \to)$ be an ARS.*

a) *An object $a \in A$ is a normal form (w.r.t. $\to$) if there is no $a' \in A$ with $a \to a'$, and an object $b \in A$ is a normal form of $a$ if $b$ is a normal form and $a \to^* b$. By $\mathtt{NF}_\to \subseteq A$ we denote the set of all normal forms w.r.t. $\to$.*

b) *An object $a \in A$ is normalizing if there exists a normal form $b$ of $a$. An object $a \in A$ is strongly normalizing if there is no infinite $\to$-reduction that starts with $a$.*

c) *The ARS $(A, \to)$ is weakly normalizing (denoted by $\mathtt{WN}(\to)$) if every $a \in A$ is normalizing.*

d) *The ARS $(A, \to)$ is strongly normalizing or terminating (denoted by $\mathtt{SN}(\to)$) if there is no infinite $\to$-reduction, i.e., if every $a \in A$ is strongly normalizing.*

Remark 3.1.8 – Properties of ARSs.
We often use the notation $P(\to)$ to indicate that the ARS $\to$ has the property $P$. If $\to$ does not have property $P$, then we write $\neg P(\to)$. So a mathematical statement like "$\mathtt{SN}(\to) \implies \mathtt{SN}(\to')$" should be read as "Termination of $\to$ implies termination of $\to'$", and a statement like "$\neg\mathtt{SN}(\to)$" means that $\to$ is not terminating.

Example 3.1.9 (Normal Forms and Termination). Continuing Example 3.1.4, $\to_2$ and $\to_3$ are not terminating, due to the infinite reductions $8 \to_2 8 \to_2 8 \to_2 \cdots$ and $2 \to_3 4 \to_3 8 \to_3 16 \to_3 \cdots$, respectively. Moreover, $\to_1$ is terminating, and we have $\mathtt{NF}_{\to_1} = \{0\}$. Written mathematically we have $\mathtt{SN}(\to_1)$, $\neg\mathtt{SN}(\to_2)$, and $\neg\mathtt{SN}(\to_3)$.

In addition to the qualitative analysis of whether an ARS terminates, we are interested in the quantitative analysis of how fast an ARS terminates. For a precise definition of "complexity" of an ARS, we have to give a meaning of "size" to the objects of the universe.[2] The size should be measurable by a natural number, e.g., the size of a list $[x_1, \ldots, x_k]$ could be its number of elements $k \in \mathbb{N}$. We use a size function $|\,.\,| : A \to \mathbb{N}$ to map each object $a \in A$ to its size $|a| \in \mathbb{N}$. Let $\sup M$ denote the *supremum* or *least upper bound* of $M$ for every $M \subseteq \mathbb{N} \cup \{\infty\}$, where $\sup \varnothing = 0$. Then the *derivation height* of an object $a \in A$ is the supremum of the number of steps over all $\to$-reductions starting with $a$. The *derivational complexity* of an ARS is a function that maps every $n \in \mathbb{N}$ to the supremum of the derivation heights of all objects of "size" at most $n$, i.e., $|a| \leq n$.

Definition 3.1.10 – Derivation Height, Derivational Complexity.

*Let $(A, \to)$ be an ARS. The derivation height $\mathrm{dh}_\to : A \to \mathbb{N} \cup \{\infty\}$ of an $a \in A$ is $\mathrm{dh}_\to(a) = \sup\{m \mid \exists a' \in A.\ a \to^m a'\}$.*
*Let $|\,.\,| : A \to \mathbb{N}$ be a size function. The derivational complexity $\mathrm{dc}_\to : \mathbb{N} \to \mathbb{N} \cup \{\infty\}$ of $\to$ is defined as*

$$\mathrm{dc}_\to(n) = \sup\{\mathrm{dh}_\to(a) \mid a \in A, |a| \leq n\}.$$

Example 3.1.11 (Complexity). For natural numbers, we use the identity as the size function, i.e., $|m| = m$. Then, we have $\mathrm{dc}_{\to_2}(n) = \mathrm{dc}_{\to_3}(n) = \infty$ for every $n \in \mathbb{N}$, and $\mathrm{dc}_{\to_1} \in \mathcal{O}(n)$. More precisely, we have $\mathrm{dc}_{\to_1}(n) = \mathrm{dh}_{\to_1}(n) = n$ for every $n \in \mathbb{N}$.

Given an ARS $\to$, the goal of complexity analysis is to determine an upper bound on the *asymptotic complexity* of the function $\mathrm{dc}_\to$.

Definition 3.1.12 – Asymptotic Complexities.

*We consider a set of complexities $\mathfrak{C} = \{\mathbf{Pol}_0, \mathbf{Pol}_1, \mathbf{Pol}_2, \ldots, \mathbf{Exp}, \mathbf{2\text{-}Exp}, \mathbf{Fin}, \mathbf{Inf}\}$ with the order $\mathbf{Pol}_0 \sqsubset \mathbf{Pol}_1 \sqsubset \mathbf{Pol}_2 \sqsubset \ldots \sqsubset \mathbf{Exp} \sqsubset \mathbf{2\text{-}Exp} \sqsubset \mathbf{Fin} \sqsubset \mathbf{Inf}$, where $\sqsubseteq$ is the reflexive closure of $\sqsubset$. For any function $f : \mathbb{N} \to \mathbb{R}_{\geq 0} \cup \{\infty\}$, we define its complexity $\iota(f) \in \mathfrak{C}$ as follows. If there is an $n \in \mathbb{N}$ with $f(n) = \infty$, then $\iota(f) = \mathbf{Inf}$. Otherwise, $f$ maps into $\mathbb{R}_{\geq 0}$, and we define*

$$\iota(f) = \begin{cases} \mathbf{Pol}_a, & \text{if } a \in \mathbb{N} \text{ is the smallest number with } f(n) \in \mathcal{O}(n^a), \\ \mathbf{Exp}, & \text{if no such } a \text{ exists, but there is a} \\ & \text{polynomial } \mathrm{pol}(n) \text{ such that } f(n) \in \mathcal{O}(2^{\mathrm{pol}(n)}), \\ \mathbf{2\text{-}Exp}, & \text{if no such polynomial exists, but there is a} \\ & \text{polynomial } \mathrm{pol}(n) \text{ such that } f(n) \in \mathcal{O}(2^{2^{\mathrm{pol}(n)}}), \\ \mathbf{Fin}, & \text{if no such polynomial exists.} \end{cases}$$

*For any ARS $\to$, its (asymptotic) derivational complexity $\iota_\to$ is $\iota(\mathrm{dc}_\to)$.*

Example 3.1.13. For the ARSs from Example 3.1.2 we have $\iota(\mathrm{dc}_{\to_1}) = \mathbf{Pol}_1$ and $\iota(\mathrm{dc}_{\to_2}) = \iota(\mathrm{dc}_{\to_3}) = \mathbf{Inf}$.

Apart from termination and complexity, another important property of ARSs is *confluence*. While termination ensures the existence of a normal form for every object,

[2] Complexity in this thesis always refers to a notion of *algorithmic complexity* or *time complexity*, i.e., we want to analyze the time it takes to terminate for a given program.

confluence ensures uniqueness of the reachable normal form. Then, resolving occurring non-determinisms leads in any case to the same final result. Similar to termination, confluence is an undecidable property in general.

Definition 3.1.14 – Confluence.
*Let $(A, \to)$ be an ARS.*

a) *We say $a, b \in A$ are joinable via $\to$ if there exists a $c \in A$ such that $a \to^* c \,{}^*\!\leftarrow b$.*

b) *We say $a \in A$ is weakly confluent if for all $b, c \in A$ with $b \leftarrow a \to c$ there exists a $d \in A$ such that $b \to^* d \,{}^*\!\leftarrow c$.*

c) *The ARS $(A, \to)$ is weakly confluent or weakly Church-Rosser (denoted $\mathtt{WCR}(\to)$) if every $a \in A$ is weakly confluent.*

d) *We say $a \in A$ is confluent if for all $b, c \in A$ with $b \,{}^*\!\leftarrow a \to^* c$ (called a peak) there exists a $d \in A$ such that $b \to^* d \,{}^*\!\leftarrow c$.*

e) *The ARS $(A, \to)$ is confluent or Church-Rosser (denoted $\mathtt{CR}(\to)$) if every $a \in A$ is confluent.*

Example 3.1.15 (Confluence). We have $\mathtt{CR}(\to_1)$ and $\mathtt{CR}(\to_2)$. For example, the peak $6 \to_2 2$ and $6 \to_2 3$ can be joined again by $2 \to_2 1$ and $3 \to_2 1$. Moreover, we have $\neg\mathtt{CR}(\to_3)$, because the peak $1 \to_3 2$ and $1 \to_3 3$ cannot be joined again w.r.t. $\to_3$.

SECTION 3.2

## Term Rewriting

*Term rewriting* deals with first-order terms and uses rules like $\ell \to r$, where $\ell$ (the left-hand side) is a pattern to be replaced and $r$ (the right-hand side) is the replacement. In contrast to strings, first-order terms have an inherent tree-like structure, and compared to $\lambda$-terms, they do not contain any higher-order or any bound variables. Again, we refer to [BN98, BKd03, Ohl02a] for further details.

In the following subsections, we introduce term rewriting and present several basic results that are important for the remainder of this thesis. Additionally, we illustrate ideas that are important for the probabilistic setting. More precisely, in Section 3.2.1, we first define the syntax and semantics of first-order terms. A *term rewrite system* is then a set of rewrite rules describing how we can rewrite one term into another via matching and systematic term replacements. In addition to the derivational complexity defined in Section 3.1, for term rewriting there is a second important notion of complexity, namely *runtime complexity*, which we introduce and explain in Section 3.2.2. Several evaluation strategies exist for applying a rewrite rule to a term, e.g., innermost rewriting, where one is only allowed to replace innermost reducible expressions in a term. We present innermost rewriting and other evaluation strategies in Section 3.2.3. Then, we present several basic results regarding witnesses for non-termination in Section 3.2.4. These results form the backbone of the proofs of many results for proving and disproving termination. Finally, in Section 3.2.5, we recapitulate a first approach on how to automatically prove termination and how to infer upper bounds on the derivational complexity and runtime complexity from these termination proofs.

### 3.2.1 Syntax and Semantics

The *signature* of a term rewrite system indicates what symbols can occur in a term and how many arguments each symbol has. As the name suggests, first-order terms are terms over a (first-order) signature, as in first-order predicate logic.

Definition 3.2.1 – Signature.

*A signature* $\Sigma = \biguplus_{n\in\mathbb{N}} \Sigma_n$ *is a union of pairwise disjoint sets such that* $\Sigma$ *is countable. An element* $f \in \Sigma_n$ *is called a function symbol of arity* $n \in \mathbb{N}$*. The arity indicates how many arguments the corresponding function symbol has. The elements of* $\Sigma_0$ *are called constant symbols or constants and have no arguments.*

Example 3.2.2 (Signature for Natural Numbers). An example for a signature that we use throughout this subsection is $\Sigma_{plus} = \biguplus_{n\in\mathbb{N}} \Sigma_n$ with $\Sigma_0 = \{\mathsf{0}\}$, $\Sigma_1 = \{\mathsf{s}\}$, $\Sigma_2 = \{\mathsf{plus}\}$, and $\Sigma_n = \varnothing$ for all $n \geq 3$. With the signature $\Sigma_{plus}$ we can represent natural numbers $n \in \mathbb{N}$ (represented by the successor function $\mathsf{s}$ and the constant $\mathsf{0}$) and arbitrarily nest them within a binary $\mathsf{plus}$-symbol. For example, the numbers 0, 1, and 2 are represented by the terms $\mathsf{0}$, $\mathsf{s}(\mathsf{0})$, and $\mathsf{s}(\mathsf{s}(\mathsf{0}))$, respectively.

Instead of $\mathsf{s}$ and $\mathsf{0}$, one can use the infinite signature $\Sigma_{nat} = \Sigma_0 = \mathbb{N}$ to represent natural numbers, consisting of a constant for each number. However, we mostly use *finite* signatures for the *finite* programs that we want to analyze.

Example 3.2.3 (Signature for Lists). As a second signature used in this section, consider $\Sigma_{list} = \biguplus_{n\in\mathbb{N}} \Sigma_n$ with $\Sigma_0 = \{\mathsf{nil}\}$, $\Sigma_1 = \{\mathsf{len}\}$, $\Sigma_2 = \{\mathsf{cons}\}$, and $\Sigma_n = \varnothing$ for all $n \geq 3$. This signature is used to describe lists (represented by the empty list $\mathsf{nil}$ and non-empty lists $\mathsf{cons}(x, xs)$ with first element $x$ and remaining list $xs$) with an additional length function $\mathsf{len}$ that computes the length of a list.

Example 3.2.4 (Signature for Binary Trees). Another important signature is $\Sigma_{tree} = \biguplus_{n\in\mathbb{N}} \Sigma_n$ with $\Sigma_0 = \{\mathsf{leaf}\}$, $\Sigma_1 = \varnothing$, $\Sigma_2 = \{\mathsf{node}\}$, and $\Sigma_n = \varnothing$ for all $n \geq 3$. This signature is used to construct binary trees (represented by leaf nodes $\mathsf{leaf}$ and inner nodes $\mathsf{node}(x, y)$ with children $x$ and $y$).

When analyzing termination and complexity, one typically imposes certain restrictions on the signature, e.g., a typical assumption for analyzing programs is that $\Sigma$ is finite and that there exists at least one constant, i.e., $\Sigma_0 \neq \varnothing$. We will add such restrictions in future chapters, but consider arbitrary signatures without any further restrictions for the introduction of (probabilistic) term rewriting in this chapter.

Terms are built from the signature and an additional countable set $\mathcal{V}$ of variables. Throughout the remaining work, function symbols are typeset in sans serif font (e.g., $\mathsf{f}$, $\mathsf{g}$, $\mathsf{plus}$, etc.), while variables are in italic font (e.g., $x$, $y$, $z$, $xs$, $ys$, $zs$, or, if more variables are needed, with indices $x_1$, $x_2$, $x_3$, etc.). Due to the natural order of the arguments of function symbols, terms gain an inherent tree-like structure.

Definition 3.2.5 – Term.

*Let* $\Sigma$ *be a signature and* $\mathcal{V}$ *be a countable set of variables with* $\mathcal{V} \cap \Sigma = \varnothing$*. The set of terms* $\mathcal{T}(\Sigma, \mathcal{V})$ *over a signature* $\Sigma$ *and a set of variables* $\mathcal{V}$ *is the smallest set with*

- $\mathcal{V} \subseteq \mathcal{T}(\Sigma, \mathcal{V})$*, and*

- *if* $f \in \Sigma_n$ *and* $t_1, \ldots, t_n \in \mathcal{T}(\Sigma, \mathcal{V})$*, then* $f(t_1, \ldots, t_n) \in \mathcal{T}(\Sigma, \mathcal{V})$.

*If* $\Sigma$ *and* $\mathcal{V}$ *are irrelevant or clear from the context, we simply write* $\mathcal{T} = \mathcal{T}(\Sigma, \mathcal{V})$.

Example 3.2.6 (Term). Let $\Sigma_{plus}$ be the signature from Example 3.2.2 and let $\{x, y\} \subseteq \mathcal{V}$. Then we have $t_1 = \mathsf{0} \in \mathcal{T}(\Sigma_{plus}, \mathcal{V})$, $t_2 = \mathsf{s}(x) \in \mathcal{T}(\Sigma_{plus}, \mathcal{V})$, and $t_3 = \mathsf{plus}(\mathsf{s}(y), \mathsf{s}(\mathsf{s}(\mathsf{0}))) \in \mathcal{T}(\Sigma_{plus}, \mathcal{V})$.

Remark 3.2.7 – Graphs, Trees, and the Relation to ARSs.

Graphs are instances of ARSs, but we want to clearly distinguish between the ARS we want to analyze, and graphs that we use within our analysis or that occur within explanations and proofs of our results. Throughout this thesis, graphs always refer to *directed graphs*.

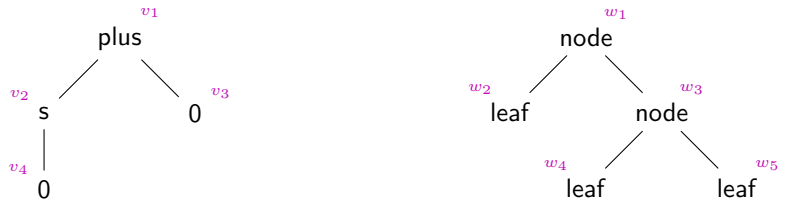
a) A *graph* is a pair $(N, E)$ consisting of a set of *nodes* $N$ and a set of *edges* $E \subseteq N \times N$. For $v \in N$, we write $vE = \{w \in N \mid (v, w) \in E\}$ for the set of *direct successors* of $v$.

b) *Paths* within a graph are $E$-reductions. We often write $v_1 v_2 v_3 \ldots$ for a path $v_1 E v_2 E v_3 E \ldots$ in the graph. For nodes $v, w \in N$, a *path from* $v$ *to* $w$ is a finite path $v = v_1 \; E \; \ldots \; E \; v_k = w$ with $k \geq 1$, i.e., of length $k - 1 \geq 0$; in particular, there is always the path of length 0 from a node to itself.

c) A (rooted) *tree* is a graph $(N, E)$ with a dedicated *root node* $\mathfrak{r} \in N$ such that for every $v \in N$ there exists exactly one path from $\mathfrak{r}$ to $v$.

d) We call a graph *labeled* if there is an additional *labeling function* $L : N \to A$ from the nodes into some set $A$.

e) The *depth* $\mathrm{d}(v)$ of a node $v \in N$ in a tree $(N, E)$ is the length of the (unique) $E$-reduction sequence from the root $\mathfrak{r}$ to the node $v$ itself. The *height* $\mathrm{h}(T)$ of a tree $T = (N, E)$ is the supremum over the depths of all nodes, i.e., $\mathrm{h}(T) = \sup\{\mathrm{d}(v) \mid v \in N\}$.




(a) Tree structure of $\mathsf{plus}(\mathsf{s}(\mathsf{0}), \mathsf{0})$. (b) Tree structure of $\mathsf{node}(\mathsf{leaf}, \mathsf{node}(\mathsf{leaf}, \mathsf{leaf}))$.

Figure 3.1: Tree representations of two terms using the signature $\Sigma_{plus}$ from Example 3.2.2 in Figure 3.1a, and the signature $\Sigma_{tree}$ from Example 3.2.4 in Figure 3.1b.

Example 3.2.8 (Terms as Trees). Terms represent labeled trees, as seen in Figure 3.1. In Figure 3.1a, we have a tree $(N, E)$ with $N = \{v_1, v_2, v_3, v_4\}$. The labels $L(v)$ of each node for the labeling function $L : N \to \Sigma$ are written inside the graphical node. For example, we have $L(v_1) = \mathsf{plus}$. The root node is $v_1$, and we have $\mathrm{d}(v_4) = 2$.

An important restriction that is sometimes imposed on terms is that they contain no variables. Such terms without variables are called *ground terms*. Due to the inductive

definition of terms, we can define most notions regarding terms, like the set of variables a term contains, recursively. Visually, this means that we start the definition at the root of the tree, and have to define how to handle inner nodes and how to handle leaves while traversing the tree.

Definition 3.2.9 – Variables, Ground Term.

*For any term $t$, let $\mathcal{V}(t)$ be the set of variables occurring in $t$ defined by*

- *if $t = x \in \mathcal{V}$ is a variable, then $\mathcal{V}(t) = \{x\}$, and*
- *if $t = f(t_1, \ldots, t_n) \in \mathcal{T}(\Sigma, \mathcal{V})$, then $\mathcal{V}(t) = \bigcup_{1 \leq i \leq n} \mathcal{V}(t_i)$.*

*A term $t$ is called a ground term if we have $\mathcal{V}(t) = \varnothing$. We often write $\mathcal{T}^{\text{gr}} = \mathcal{T}(\Sigma, \varnothing)$ for the set of ground terms.*

Example 3.2.10 (Variables, Ground Term). Considering the terms $t_1 = \mathsf{0}$, $t_2 = \mathsf{s}(x)$, and $t_3 = \mathsf{plus}(\mathsf{s}(y), \mathsf{s}(\mathsf{s}(\mathsf{0})))$ again, only $t_1$ is a ground term, as the other two contain variables. To be precise, we have $\mathcal{V}(t_1) = \varnothing$, $\mathcal{V}(t_2) = \{x\}$, and $\mathcal{V}(t_3) = \{y\}$.

Next, we introduce several notations to ease describing terms, describing operations on terms, and creating new terms. The leaves of our tree representation of terms can only be constants or variables. While constants denote a concrete end of the term, variables work as a placeholder that can potentially be *instantiated* by further terms. Instantiation is done via a *substitution*, which is a function that maps variables to terms. Applying a substitution $\delta$ to a term $t$ can then be seen as extending the leaves of the tree labeled with a variable $x$ by the corresponding tree of $\delta(x)$.

Definition 3.2.11 – Substitution.

*A substitution is a function $\sigma : \mathcal{V} \to \mathcal{T}(\Sigma, \mathcal{V})$ with $\sigma(x) = x$ for all but finitely many $x \in \mathcal{V}$. Therefore, $\sigma$ can be identified with $[x_1/\sigma(x_1), \ldots, x_k/\sigma(x_k)]$ if we have $x_i \neq \sigma(x_i)$ for all $1 \leq i \leq k$ and $y = \sigma(y)$ for all other variables $y \notin \{x_1, \ldots, x_k\}$. We often write $x\sigma$ instead of $\sigma(x)$. Substitutions homomorphically extend to terms. If $t = f(t_1, \ldots, t_n) \in \mathcal{T}(\Sigma, \mathcal{V})$, then $t\sigma = f(t_1\sigma, \ldots, t_n\sigma)$.*

Example 3.2.12 (Substitution). Reconsider the terms $t_1 = \mathsf{0}$, $t_2 = \mathsf{s}(x)$, and $t_3 = \mathsf{plus}(\mathsf{s}(y), \mathsf{s}(\mathsf{s}(\mathsf{0})))$ and consider the substitution $\sigma = [x/\mathsf{s}(\mathsf{0}), y/\mathsf{plus}(x, x)]$. Then $t_1\sigma = \mathsf{0}$, $t_2\sigma = \mathsf{s}(\mathsf{s}(\mathsf{0}))$, and $t_3\sigma = \mathsf{plus}(\mathsf{s}(\mathsf{plus}(x, x)), \mathsf{s}(\mathsf{s}(\mathsf{0}))))$.

Substitutions induce a quasi-order on terms, where a term $s \in \mathcal{T}$ *matches* another term $t \in \mathcal{T}$ if there exists a substitution $\sigma$ such that $s\sigma = t$. Substitutions themselves can also be quasi-ordered component-wise. Sometimes we are interested in a common upper bound w.r.t. this order, which means that both terms are allowed to be instantiated by substitutions, which yields the notion of *unification*.

Definition 3.2.13 – Matching, Unification, Most General Unifier.

*A term $s \in \mathcal{T}$ matches a term $t \in \mathcal{T}$ if there exists a substitution $\sigma$ such that $s\sigma = t$. Two terms $s, t \in \mathcal{T}$ are unifiable if there exists a substitution $\sigma$, called a unifier, such that $s\sigma = t\sigma$. A most general unifier (mgu) is a unifier $\sigma$ such that for every other unifier $\sigma'$ there exists a substitution $\delta$ with $\sigma' = \sigma\delta$.*

Example 3.2.14 (Matching and Unification). The term $\mathsf{plus}(\mathsf{s}(x), \mathsf{s}(x))$ matches $\mathsf{plus}(\mathsf{s}(\mathsf{s}(\mathsf{0})), \mathsf{s}(\mathsf{s}(\mathsf{0})))$ using the substitution $[x/\mathsf{s}(\mathsf{0})]$. The term $\mathsf{plus}(\mathsf{s}(\mathsf{s}(\mathsf{0})), y)$ also

matches $\mathsf{plus}(\mathsf{s}(\mathsf{0}), \mathsf{s}(\mathsf{s}(\mathsf{0})))$ using the substitution $[y/\mathsf{s}(\mathsf{s}(\mathsf{0}))]$. Finally, the two terms $\mathsf{plus}(\mathsf{s}(x), \mathsf{s}(x))$ and $\mathsf{plus}(\mathsf{s}(\mathsf{s}(\mathsf{0})), y)$ are unifiable and the most general unifier is $[x/\mathsf{s}(\mathsf{0}), y/\mathsf{s}(\mathsf{s}(\mathsf{0}))]$.

It is a well-known fact that for two unifiable first-order terms, there always exists an mgu, which is unique up to renaming of variables, and one can check for unification and compute an mgu in polynomial time. Moreover, whether a term $s$ matches a term $t$ can be decided in linear time w.r.t. the size of the target term $t$.

Instead of changing a term in the variables, we can also add a *context* around the term, creating a new root of the corresponding tree. Formally, a context is a term $\mathcal{C} \in \mathcal{T}(\Sigma \uplus \{\Box\}, \mathcal{V})$, where $\Box$ is a *fresh* constant ($\Box \notin \Sigma \cup \mathcal{V}$), called the *hole*.

Definition 3.2.15 – Context.
*A term $\mathcal{C} \in \mathcal{T}(\Sigma \uplus \{\Box\}, \mathcal{V})$ is called a context if there is at least one occurrence of $\Box$ inside $\mathcal{C}$. If $\mathcal{C}$ contains exactly $n$ holes and $t_1, \ldots, t_n \in \mathcal{T}(\Sigma, \mathcal{V})$, then $\mathcal{C}[t_1, \ldots, t_n]$ denotes the result of replacing the holes of $\mathcal{C}$ from left to right by $t_1, \ldots, t_n$.*

Example 3.2.16 (Context). An example of a context over the signature $\Sigma_{plus}$ would be $\mathcal{C} = \mathsf{plus}(\Box, \mathsf{s}(\mathsf{0}))$. Here, we have $\mathcal{C}[\mathsf{s}(\mathsf{0})] = \mathsf{plus}(\mathsf{s}(\mathsf{0}), \mathsf{s}(\mathsf{0}))$.

In Definition 3.2.15 we already considered an intuitive notion of order by replacing symbols "from left to right". Next, we want to formalize this intuitive notion via *positions* in a term and by defining an order on these positions.

Definition 3.2.17 – Position.
*For a term $t \in \mathcal{T}$, the set of positions $\mathrm{pos}(t)$ is defined to be the smallest subset of $\mathbb{N}^*$ satisfying*

- *$\varepsilon \in \mathrm{pos}(t)$, and*
- *if $t = f(t_1, \ldots, t_n)$ then for all $1 \leq i \leq n$ and all $\pi \in \mathrm{pos}(t_i)$ we have $i.\pi \in \mathrm{pos}(t)$.*

*The position $\varepsilon$ is called the root position of the term $t$ and the symbol at this position is called the root symbol, denoted by $\mathrm{root}(t)$.*

Example 3.2.18 (Position). For $t_1 = \mathsf{0}$ we have $\mathrm{pos}(t_1) = \{\varepsilon\}$, for $t_2 = \mathsf{s}(x)$ we have $\mathrm{pos}(t_2) = \{\varepsilon, 1\}$, and finally, for $t_3 = \mathsf{plus}(\mathsf{s}(y), \mathsf{s}(\mathsf{0}))$ we have $\mathrm{pos}(t_3) = \{\varepsilon, 1, 2, 1.1, 2.1, 2.1.1\}$.

Positions are partially ordered via the prefix ordering. Additionally, positions that are incomparable via the prefix order can be ordered from left to right.

Definition 3.2.19 – Prefix Order.
*We define a partial order on $\mathbb{N}^*$ by $\pi \leq \tau$ if there exists a $\chi \in \mathbb{N}^*$ such that $\pi.\chi = \tau$. This is called the prefix order. We say that $\pi$ is above $\tau$ ($\tau$ is below $\pi$) if $\pi \leq \tau$. We say that two positions $\pi, \tau \in \mathbb{N}^*$ are orthogonal or parallel (denoted by $\pi \perp \tau$) if they are incomparable in the prefix order, i.e., $\pi \not\leq \tau$ and $\tau \not\leq \pi$.*
*For two orthogonal positions $\pi$ and $\tau$ we define $\pi \prec \tau$ if we have $i < j$ for the unique $i, j$ such that $\chi.i \leq \pi$ and $\chi.j \leq \tau$, where $\chi$ is the longest common prefix of $\pi$ and $\tau$. Here, we say that $\pi$ is left of $\tau$ ($\tau$ is right of $\pi$) if $\pi \prec \tau$.*

Example 3.2.20 (Prefix Order). Consider the three positions $1, 2, 1.1 \in \mathbb{N}^*$. We have $1 \perp 2$, and additionally, 1 is left of 2 ($1 \prec 2$). Moreover, 1 is above 1.1 ($1 \leq 1.1$).

With all these notions, we can define how to transform or *replace* terms.

Definition 3.2.21 – Term Replacement.
*If $\pi \in \mathrm{pos}(t)$ then $t|_\pi$ denotes the subterm at position $\pi$, i.e.,*

- $t|_\varepsilon = t$, *and*
- *if* $t = f(t_1, \ldots, t_n)$ *and* $\pi = i.\pi'$ *then* $t|_\pi = t_i|_{\pi'}$.

*Furthermore, if $r \in \mathcal{T}$ and $\pi \in \mathrm{pos}(t)$ then $t[r]_\pi$ denotes the term that results from replacing the subterm $t|_\pi$ with the term $r$, i.e.,*

- $t[r]_\varepsilon = r$, *and*
- *if* $t = f(t_1, \ldots, t_i, \ldots, t_n)$ *and* $\pi = i.\pi'$ *then* $t[r]_\pi = f(t_1, \ldots, t_i[r]_{\pi'}, \ldots, t_n)$.

Example 3.2.22 (Term Replacement). For $t_3 = \mathsf{plus}(\mathsf{s}(y), \mathsf{s}(\mathsf{s}(0)))$ we have $t_3|_2 = \mathsf{s}(\mathsf{s}(0))$, $t_3[y]_2 = \mathsf{plus}(\mathsf{s}(y), y)$, and $t_3[\mathsf{s}(0)]_{1.1} = \mathsf{plus}(\mathsf{s}(\mathsf{s}(0)), \mathsf{s}(\mathsf{s}(0)))$.

Following the intuitive illustration of terms as labeled trees, we want to be able to define a *containment* relation that holds whenever some tree is a subtree of another one. We can either consider a new root node in the tree representation of a term and then take all the nodes reachable from this new root, which yields the notion of a *subterm*, or we can pick a connected subset of nodes in the tree representation, which yields the notion of a *term occurrence*.

Definition 3.2.23 – Subterm.
*Let $t, s \in \mathcal{T}$ be two terms. We say that $s$ is a subterm of $t$ at position $\pi$ (denoted by $s \trianglelefteq_\pi t$) if $s = t|_\pi$. We call $s$ a proper subterm (denoted by $s \triangleleft_\pi t$) if $\pi \neq \varepsilon$. We often omit the position $\pi$ and simply write $s \trianglelefteq t$.*

Example 3.2.24 (Subterm). Reconsider the terms $t_1 = \mathsf{0}$, $t_2 = \mathsf{s}(x)$, and $t_3 = \mathsf{plus}(\mathsf{s}(y), \mathsf{s}(\mathsf{s}(0)))$. Here, we have $t_1 \triangleleft t_3$, and $t_i \trianglelefteq t_i$ for all $i$, but there are no other subterm relations.

Definition 3.2.25 – Term Occurrence.
*Let $t, s \in \mathcal{T}$ be two terms. We say that $t$ occurs in $s$ at position $\pi$ (denoted $t \blacktriangleleft_\pi s$) if $s|_\pi = t\sigma$ for some substitution $\sigma$. We often omit the position $\pi$ and write $t \blacktriangleleft s$.*

Example 3.2.26 (Term Occurrence). For example, we have $\mathsf{g}(x) \blacktriangleleft_\pi \mathsf{g}(\mathsf{g}(\mathsf{g}(x)))$ for all $\pi \in \{\varepsilon, 1, 1.1\}$, and $\mathsf{g}(\mathsf{g}(x)) \blacktriangleleft_\pi \mathsf{g}(\mathsf{g}(\mathsf{g}(x)))$ holds for $\pi \in \{\varepsilon, 1\}$.

Finally, with all these definitions, we can consider rewriting according to a fixed set of *rewrite rules*, which yields the notion of a *term rewrite system*.

Definition 3.2.27 – Term Rewrite System.
*For $\ell, r \in \mathcal{T}$, $\ell \to r$ is called a rewrite rule if $\mathcal{V}(r) \subseteq \mathcal{V}(\ell)$ and $\ell \notin \mathcal{V}$. A term rewrite system (TRS) $\mathcal{R}$ is a set of rewrite rules.*
*The TRS $\mathcal{R}$ induces an abstract reduction system $(\mathcal{T}, \to_\mathcal{R})$ with the rewrite relation*

$\to_{\mathcal{R}} \subseteq \mathcal{T} \times \mathcal{T}$ *via*

$$s \to_{\mathcal{R}} t, \qquad \text{if } s|_\pi = \ell\sigma \text{ and } t = s[r\sigma]_\pi,$$

*for a position* $\pi \in \mathrm{pos}(s)$, *a rule* $\ell \to r \in \mathcal{R}$, *and a substitution* $\sigma$. *To highlight the used position, or rule, we sometimes include them in the index of* $\to_{\mathcal{R}}$ *and may omit* $\mathcal{R}$ *if the used rule is provided, e.g.,* $\to_{\mathcal{R},\pi}$ *or* $\to_{\ell\to r,\pi}$. *A subterm* $s|_\pi$ *that is matched by the left-hand side of a rule is called a redex (for "reducible expression"). We often use* $\mathcal{R}$ *to speak about the induced relation* $\to_{\mathcal{R}}$.

The conditions $\mathcal{V}(r) \subseteq \mathcal{V}(\ell)$ and $\ell \notin \mathcal{V}$ rule out rules that trivially cause non-termination.

Since $\mathcal{R}$ (or more precisely, since $(\mathcal{T}, \to_{\mathcal{R}})$) is an ARS, we can reuse all definitions regarding ARSs, e.g., reduction sequences, normal forms, etc. Nevertheless, we want to restate the notion we are mainly interested in: *termination*.

Definition 3.2.28 – Termination.
*Let* $\mathcal{R}$ *be a TRS. We call it terminating if there is no infinite rewrite sequence*

$$t_0 \to_{\mathcal{R}} t_1 \to_{\mathcal{R}} t_2 \to_{\mathcal{R}} \dots.$$

Additionally, we extend the idea of a normal form to terms in *argument normal form* where every proper subterm is in normal form, but we may be able to perform a rewrite step at the root.

Definition 3.2.29 – Argument Normal Form.
*Let* $\mathcal{R}$ *be a TRS and* $t \in \mathcal{T}$. *We say* $t$ *is in argument normal form w.r.t.* $\mathcal{R}$ *if every proper subterm* $t' \lhd t$ *is in normal form w.r.t.* $\mathcal{R}$, *i.e.,* $t' \in \mathtt{NF}_{\mathcal{R}}$. *The set of all argument normal forms w.r.t.* $\mathcal{R}$ *is denoted by* $\mathtt{ANF}_{\mathcal{R}}$.

Example 3.2.30 (Addition of Natural Numbers). Consider the TRS $\mathcal{R}_{\mathsf{plus}}$ consisting of the two rewrite rules (3.1) and (3.2) over the signature $\Sigma_{plus}$ and a set of variables with $\{x, y\} \subseteq \mathcal{V}$.

$$\mathsf{plus}(\mathsf{0}, y) \to y \qquad (3.1) \qquad\qquad \mathsf{plus}(\mathsf{s}(x), y) \to \mathsf{s}(\mathsf{plus}(x, y)) \qquad (3.2)$$

The TRS $\mathcal{R}_{\mathsf{plus}}$ is an implementation of the addition of two natural numbers. If we want to calculate "$2 + 1$" with $\mathcal{R}_{\mathsf{plus}}$, we start with the term $\mathsf{plus}(\mathsf{s}(\mathsf{s}(\mathsf{0})), \mathsf{s}(\mathsf{0}))$ and iteratively apply rewrite rules until we reach a normal form. The left-hand side of Rule (3.2) matches the term $\mathsf{plus}(\mathsf{s}(\mathsf{s}(\mathsf{0})), \mathsf{s}(\mathsf{0}))$ using the substitution $\sigma = [x/\mathsf{s}(\mathsf{0}), y/\mathsf{s}(\mathsf{0})]$. Hence, we can rewrite at the root position $\varepsilon$ such that[3]

$$\underline{\mathsf{plus}}(\mathsf{s}(\mathsf{s}(\mathsf{0})), \mathsf{s}(\mathsf{0})) \to_{\mathcal{R}_{\mathsf{plus}}} \mathsf{s}(\mathsf{plus}(\mathsf{s}(\mathsf{0}), \mathsf{s}(\mathsf{0}))).$$

Then the left-hand side of Rule (3.2) matches the subterm at position 1 using the substitution $\sigma = [x/\mathsf{0}, y/\mathsf{s}(\mathsf{0})]$, and thus,

$$\mathsf{s}(\underline{\mathsf{plus}}(\mathsf{s}(\mathsf{0}), \mathsf{s}(\mathsf{0}))) \to_{\mathcal{R}_{\mathsf{plus}}} \mathsf{s}(\mathsf{s}(\mathsf{plus}(\mathsf{0}, \mathsf{s}(\mathsf{0})))).$$

Finally, the left-hand side of (3.1) matches the term just obtained at position 1.1. Thus, we have

$$\mathsf{s}(\mathsf{s}(\underline{\mathsf{plus}}(\mathsf{0}, \mathsf{s}(\mathsf{0})))) \to_{\mathcal{R}_{\mathsf{plus}}} \mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{0}))).$$

[3] We always mark the function symbol at the root of the redex that we reduce.

In the end, we obtain the normal form $\mathsf{s}(\mathsf{s}(\mathsf{s}(0))) \in \mathtt{NF}_{\mathcal{R}_{\mathsf{plus}}}$ representing the natural number 3, which is the correct result for "$2 + 1$".

Termination of $\mathcal{R}_{\mathsf{plus}}$ holds since in each rewrite step we remove one $\mathsf{s}(\ldots)$ occurrence in the first argument of the plus function symbol, or we remove the plus function symbol itself. We formally prove termination of $\mathcal{R}_{\mathsf{plus}}$ in Example 3.2.65, and explain how the tool AProVE is able to find such a termination proof fully automatically.

Example 3.2.31 (List Length). Consider the signature $\Sigma = \Sigma_{plus} \cup \Sigma_{list}$, a variable set with $\{x, xs\} \subseteq \mathcal{V}$, and the TRS $\mathcal{R}_{\mathsf{len}}$ consisting of the two rules (3.3) and (3.4). $\mathcal{R}_{\mathsf{len}}$ computes the length of a list.

$$\mathsf{len}(\mathsf{nil}) \to 0 \qquad (3.3) \qquad\qquad \mathsf{len}(\mathsf{cons}(x, xs)) \to \mathsf{s}(\mathsf{len}(xs)) \qquad (3.4)$$

We can compute the length of a list $[0, 0]$ containing two zeros via the rewrite sequence

$$\underline{\mathsf{len}}(\mathsf{cons}(0, \mathsf{cons}(0, \mathsf{nil}))) \to_{\mathcal{R}_{\mathsf{len}}} \mathsf{s}(\underline{\mathsf{len}}(\mathsf{cons}(0, \mathsf{nil}))) \to_{\mathcal{R}_{\mathsf{len}}} \mathsf{s}(\mathsf{s}(\underline{\mathsf{len}}(\mathsf{nil}))) \to_{\mathcal{R}_{\mathsf{len}}} \mathsf{s}(\mathsf{s}(0)).$$

The TRS $\mathcal{R}_{\mathsf{len}}$ terminates because in each rewrite step we decrease the size of the list in the argument of the len function symbol. We formally prove termination of $\mathcal{R}_{\mathsf{len}}$ in Example 3.2.66. AProVE is able to automatically prove its termination as well.

Example 3.2.32 (Quicksort). Term rewrite systems are especially good for easily expressing data structures. One of the most important problems for a computer scientist is to sort a list of numbers $[x_1, \ldots, x_n]$. One prominent algorithm to solve the sorting problem is *quicksort*. Quicksort is a divide-and-conquer algorithm that first chooses a pivot element $x = x_j$ of $[x_1, \ldots, x_n]$ and then splits the list of remaining elements into two sublists $L = [x_i \mid 1 \leq i \leq n, i \neq j, x_i < x]$ and $R = [x_i \mid 1 \leq i \leq n, i \neq j, x_i \geq x]$. Afterwards, we sort the (shorter) lists $L$ and $R$ recursively, which yields the sorted lists $L'$ and $R'$, respectively. In the end, we combine the sorted lists by placing the pivot element $x$ in between the two lists, i.e., the result is $[L', x, R']$.

The following TRS $\mathcal{R}_{\mathsf{qs}}$ is an implementation of a deterministic quicksort algorithm.

$$\begin{aligned}
\mathsf{qs}(\mathsf{nil}) &\to \mathsf{nil}\\
\mathsf{qs}(\mathsf{cons}(x, xs)) &\to \mathsf{qsHelp}(\mathsf{cons}(x, xs))\\
\mathsf{qsHelp}(\mathsf{cons}(x, xs)) &\to \mathsf{app}(\mathsf{qs}(\mathsf{low}(x, xs)), \mathsf{cons}(x, \mathsf{qs}(\mathsf{high}(x, xs))))\\
\mathsf{low}(x, \mathsf{nil}) &\to \mathsf{nil}\\
\mathsf{low}(x, \mathsf{cons}(y, ys)) &\to \mathsf{ifLow}(\mathsf{leq}(x, y), x, \mathsf{cons}(y, ys))\\
\mathsf{ifLow}(\mathsf{t}, x, \mathsf{cons}(y, ys)) &\to \mathsf{low}(x, ys)\\
\mathsf{ifLow}(\mathsf{f}, x, \mathsf{cons}(y, ys)) &\to \mathsf{cons}(y, \mathsf{low}(x, ys))\\
\mathsf{high}(x, \mathsf{nil}) &\to \mathsf{nil}\\
\mathsf{high}(x, \mathsf{cons}(y, ys)) &\to \mathsf{ifHigh}(\mathsf{leq}(x, y), x, \mathsf{cons}(y, ys))\\
\mathsf{ifHigh}(\mathsf{t}, x, \mathsf{cons}(y, ys)) &\to \mathsf{cons}(y, \mathsf{high}(x, ys))\\
\mathsf{ifHigh}(\mathsf{f}, x, \mathsf{cons}(y, ys)) &\to \mathsf{high}(x, ys)\\
\mathsf{leq}(0, x) &\to \mathsf{t}\\
\mathsf{leq}(\mathsf{s}(x), 0) &\to \mathsf{f}\\
\mathsf{leq}(\mathsf{s}(x), \mathsf{s}(y)) &\to \mathsf{leq}(x, y)\\
\mathsf{app}(\mathsf{nil}, ys) &\to ys\\
\mathsf{app}(\mathsf{cons}(x, xs), ys) &\to \mathsf{cons}(x, \mathsf{app}(xs, ys))
\end{aligned}$$

These rules represent the classical quicksort algorithm where we always choose the first element of the list as the pivot element. Here, app computes list concatenation, $\mathsf{low}(x, xs)$ returns all elements of the list $xs$ that are smaller than $x$, and $\mathsf{high}(x, xs)$ returns all elements of the list $xs$ that are greater than or equal to $x$.

Termination of $\mathcal{R}_{\mathsf{qs}}$ can be proven automatically by AProVE. We present in Example 3.4.4 a probabilistic version of this quicksort algorithm that selects its pivot elements randomly. We show in Chapter 7 how to analyze probabilistic programs like $\mathcal{P}_{\mathsf{qs}}$ that deal with data structures.

We end this subsection with several important properties of a TRS, which we always illustrate with a short example. All these properties are used within this thesis at different points, but we always explain the property again once needed.

Definition 3.2.33 – Joinable.
*Let $\mathcal{R}$ be a TRS. Two terms $s, t \in \mathcal{T}$ are joinable via $\mathcal{R}$ if there exists a $w \in \mathcal{T}$ such that $s \to_{\mathcal{R}}^* w \,{}^*\!\leftarrow_{\mathcal{R}} t$.*

Example 3.2.34 (Joinable). Let $\mathcal{R}$ contain the four rules $\mathsf{a} \to \mathsf{b}$, $\mathsf{b} \to \mathsf{a}$, $\mathsf{a} \to \mathsf{c}$, and $\mathsf{b} \to \mathsf{d}$. The two terms a and b are joinable via $\mathcal{R}$, as we have $\mathsf{a} \to_{\mathcal{R}} \mathsf{b}$. The same holds for b and c, as we have $\mathsf{b} \to_{\mathcal{R}} \mathsf{a} \to_{\mathcal{R}} \mathsf{c}$. The two terms c and d are not joinable, as both are normal forms and cannot be rewritten any further.

Definition 3.2.35 – Overlapping.
*Let $\mathcal{R}$ be a TRS. Two rules $\ell_1 \to r_1, \ell_2 \to r_2 \in \mathcal{R}$ with renamed variables such that $\mathcal{V}(\ell_1) \cap \mathcal{V}(\ell_2) = \varnothing$ are overlapping if there exists a non-variable position $\pi$ of $\ell_1$ such that $\ell_1|_\pi$ and $\ell_2$ are unifiable, i.e., there exists a substitution $\sigma$ such that $\ell_1|_\pi \sigma = \ell_2 \sigma$. If both rules are renamed copies of the same rule, then we require that $\pi \neq \varepsilon$. $\mathcal{R}$ is non-overlapping (NO) if it has no overlapping rules.*

Example 3.2.36 (Overlapping). The TRS $\mathcal{R}$ containing the two rules $\mathsf{g}(x, y) \to x$ and $\mathsf{g}(x, y) \to y$ is overlapping because the two rules overlap at the root position. The TRS $\mathcal{R}$ containing the only rule $\mathsf{g}(\mathsf{g}(x)) \to x$ is overlapping, because the only rule overlaps with itself at the non-root position 1. Finally, the TRS $\mathcal{R}_{\mathsf{plus}}$ from Example 3.2.30 is non-overlapping.

Definition 3.2.37 – Linear, Erasing, Duplicating.
*Let $\mathcal{R}$ be a TRS. By $|t|_x$ we denote the number of positions $\pi \in \mathrm{pos}(t)$ such that $t|_\pi = x \in \mathcal{V}$ for every term $t \in \mathcal{T}$.*

a) *A rule $\ell \to r \in \mathcal{R}$ is left-linear (LL) if $|\ell|_x \leq 1$ for every variable $x \in \mathcal{V}$, right-linear (RL) if $|r|_x \leq 1$ for every variable $x \in \mathcal{V}$, and linear (L) if it is both left-linear and right-linear.*

b) *A rule $\ell \to r \in \mathcal{R}$ is duplicating (or variable increasing) if $|\ell|_x < |r|_x$ for some variable $x \in \mathcal{V}$. If it is not duplicating, then we call it non-duplicating (ND).*

c) *A rule $\ell \to r \in \mathcal{R}$ is variable decreasing if $|\ell|_x > |r|_x$ for some variable $x \in \mathcal{V}$. If it is not variable decreasing, then we call it NVD.*

d) *A rule $\ell \to r \in \mathcal{R}$ is erasing if $|r|_x = 0$ for some variable $x \in \mathcal{V}(\ell)$ occurring in the left-hand side. If it is not erasing, then we call it non-erasing (NE).*

e) *The TRS $\mathcal{R}$ is LL / RL / L / ND / NVD / NE if all of its rules are.*

Example 3.2.38 (Variable Conditions). The following is a list of examples to illustrate all notions from Definition 3.2.37.

- $\mathsf{f}(x) \to \mathsf{g}(x)$ is left-linear and $\mathsf{h}(x,x) \to \mathsf{g}(x)$ is not left-linear.
- $\mathsf{f}(x) \to \mathsf{g}(x)$ is right-linear and $\mathsf{f}(x) \to \mathsf{h}(x,x)$ is not right-linear.
- $\mathsf{f}(x) \to \mathsf{g}(x)$ is linear, and $\mathsf{h}(x,x) \to \mathsf{g}(x)$ and $\mathsf{f}(x) \to \mathsf{h}(x,x)$ are not linear.
- $\mathsf{h}(x,x) \to \mathsf{c}(x,x,x)$ is duplicating and $\mathsf{h}(x,x) \to \mathsf{h}(x,x)$ is non-duplicating.
- $\mathsf{h}(x,x) \to \mathsf{g}(x)$ is variable decreasing and $\mathsf{h}(x,x) \to \mathsf{h}(x,x)$ is $\mathtt{NVD}$.
- $\mathsf{f}(x) \to \mathsf{a}$ is erasing and $\mathsf{h}(x,x) \to \mathsf{g}(x)$ is non-erasing.

### 3.2.2 Complexity

We already introduced the notion of derivational complexity in Section 3.1. The notion of "size" we use for a term is simply the number of function symbols and variables it contains, or in other words, the number of nodes in its tree representation.

Definition 3.2.39 – Term Size.
*For a term $t \in \mathcal{T}$, the size $|t|$ denotes the number of function symbols and variables in $t$, i.e.,*

- $|x| = 1$ *for every variable* $x \in \mathcal{V}$, *and*
- *if* $t = f(t_1, \dots, t_n) \in \mathcal{T}$, *then* $|t| = 1 + \sum_{i=1}^{n} |t_i|$.

Example 3.2.40. We have $|\mathsf{plus}(\mathsf{s}(\mathsf{s}(0)), \mathsf{s}(\mathsf{s}(0)))| = 7$ and $|\mathsf{plus}(x,y)| = 3$. Moreover, we have $|\mathsf{f}^n(\mathsf{a})| = n+1$ for every $n \in \mathbb{N}$ where $\mathsf{f}^n(\mathsf{a})$ abbreviates $\underbrace{\mathsf{f}(\dots\mathsf{f}(\mathsf{a})\dots)}_{n \text{ times}}$.

In addition to the derivational complexity as defined in Definition 3.1.10, there exists a second prominent notion of time complexity for term rewrite systems, namely *runtime complexity* [HL89, HM08]. While the derivational complexity considers rewrite sequences that start with an arbitrary term of size at most $n$, for runtime complexity one restricts the start terms to be *basic*. Basic terms represent an algorithm that is provided with fixed input data, i.e., the arguments should not contain any function symbol representing another algorithm. For this, we divide the signature into *defined symbols* (representing executable functions) and *constructors* (representing data).

Definition 3.2.41 – Defined Symbols, Constructor, Basic Terms.
*For a TRS $\mathcal{R}$, we decompose its signature $\Sigma = \mathcal{C} \uplus \mathcal{D}$ such that $f \in \mathcal{D}$ iff $f = \mathrm{root}(\ell)$ for some rule $\ell \to r \in \mathcal{R}$. The symbols in $\mathcal{C}$ and $\mathcal{D}$ are called constructors and defined symbols, respectively.*
*A term $t \in \mathcal{T}$ is basic if $t = f(t_1, \dots, t_k)$ such that $f \in \mathcal{D}$ and $t_i \in \mathcal{T}(\mathcal{C}, \mathcal{V})$ for all $1 \le i \le k$, and the set of all basic terms is denoted by $\mathcal{T}_{\mathcal{B}}$.*

Example 3.2.42. Reconsider the TRS $\mathcal{R}_{\mathsf{plus}}$ from Example 3.2.30. Here, the defined symbols are $\mathcal{D} = \{\mathsf{plus}\}$ and the constructors are $\mathcal{C} = \{\mathsf{s}, 0\}$. The terms $\mathsf{plus}(\mathsf{s}(\mathsf{s}(0)), \mathsf{s}(\mathsf{s}(0)))$ and $\mathsf{plus}(x,y)$ are basic, while the term $\mathsf{plus}(0, \mathsf{plus}(0,0))$ is not basic, because of the inner $\mathsf{plus}$ function symbol in the arguments of the outer $\mathsf{plus}$.

The definition of the runtime complexity of a TRS is now similar to the definition of the derivational complexity, where we additionally restrict the start terms to be basic.

Definition 3.2.43 – Runtime Complexity.

*Let $(\mathcal{T}, \to)$ be an ARS on terms with a set of basic terms $\mathcal{T}_\mathcal{B} \subseteq \mathcal{T}$. The runtime complexity* $\mathrm{rc}_\to : \mathbb{N} \to \mathbb{N} \cup \{\infty\}$ *of* $\to$ *is defined as*

$$\mathrm{rc}_\to(n) = \sup\{\mathrm{dh}_\to(t) \mid t \in \mathcal{T}_\mathcal{B}, |t| \leq n\}.$$

The runtime complexity of a TRS $\mathcal{R}$ yields a lower bound for the derivational complexity of $\mathcal{R}$. The other direction does not hold as we see in the following counterexample.

Counterexample 3.2.44 (Runtime Complexity vs. Derivational Complexity). To see the difference between the derivational complexity and the runtime complexity, consider the TRS $\mathcal{R}$ over the signature $\Sigma = \{\mathsf{f}, \mathsf{a}\}$ with the only rule $\mathsf{f}(\mathsf{a}) \to \mathsf{a}$. On the one hand, we have $\iota(\mathrm{rc}_{\to_\mathcal{R}}) = \mathbf{Pol}_0$, because the only basic terms are $\mathsf{f}(\mathsf{a})$ and $\mathsf{f}(x)$ for variables $x \in \mathcal{V}$, and both of them reach a normal form after at most one rewrite step. On the other hand, we have $\iota(\mathrm{dc}_{\to_\mathcal{R}}) = \mathbf{Pol}_1$, because we can start with the term $\mathsf{f}^n(\mathsf{a})$ and it takes $n$ rewrite steps to reach the normal form $\mathsf{a}$, i.e., $\underline{\mathsf{f}}^n(\mathsf{a}) \to_\mathcal{R} \underline{\mathsf{f}}^{n-1}(\mathsf{a}) \to_\mathcal{R} \cdots \to_\mathcal{R} \underline{\mathsf{f}}(\mathsf{a}) \to_\mathcal{R} \mathsf{a}$.

We see in Section 3.2.5 that finding upper bounds on the runtime complexity is in general easier than finding upper bounds on the derivational complexity, due to the additional restriction on the initial terms. Nevertheless, both notions of complexity are important, depending on what the TRS should model: a fixed algorithm where executable functions and data are strictly divided (runtime complexity), or a system where no clear distinction between data and functions can be made (derivational complexity).

### 3.2.3 Evaluation Strategies

While rewriting a term $t$ with a TRS $\mathcal{R}$, multiple different non-determinisms occur. First, we have to decide on a redex at a position $\pi \in \mathrm{pos}(t)$ in $t$ that is matched by some left-hand side of a rule $\ell \to r \in \mathcal{R}$. Second, if there are multiple different rules applicable at position $\pi$, because the subterm $t|_\pi$ is matched by multiple left-hand sides, then we additionally have to decide on a rewrite rule to use. In this section, we consider the first non-determinism and define *evaluation strategies* that restrict the search space for the redex position. We only consider *positional strategies* in this thesis, i.e., strategies that define the possible redexes purely based on the positions of the possible redexes. Positional strategies can be considered memoryless. However, note that the notion of a strategy does not directly translate to that of a scheduler in MDPs (see Definition 2.4.24), because an evaluation strategy does not fully resolve all occurring non-determinisms, but just reduces the number of different possibilities, since the choice for the specific rewrite rule to use remains.

Definition 3.2.45 – Evaluation Strategy.

*Let $\mathcal{R}$ be a TRS and let $s, t \in \mathcal{T}$. We denote the set of positions in $t$ at which we have a redex w.r.t. $\mathcal{R}$ by* $\mathrm{pos}^{\mathrm{red}}_\mathcal{R}(t) = \{\pi \in \mathrm{pos}(t) \mid \exists \sigma.\ \exists \ell \to r \in \mathcal{R}.\ t|_\pi = \ell\sigma\}$.
*An evaluation strategy is a function* $\mathsf{s} : \mathcal{T} \to \mathrm{Pow}(\mathbb{N}^*)$ *such that* $\mathsf{s}(t) \subseteq \mathrm{pos}^{\mathrm{red}}_\mathcal{R}(t)$.
*So an evaluation strategy limits the positions where we can rewrite.*

*The TRS $\mathcal{R}$ together with the evaluation strategy $\mathbf{s}$ induces an ARS $(\mathcal{T}, \xrightarrow{\mathbf{s}}_{\mathcal{R}})$ with the rewrite relation $\xrightarrow{\mathbf{s}}_{\mathcal{R}} \subseteq \mathcal{T} \times \mathcal{T}$ via $s \xrightarrow{\mathbf{s}}_{\mathcal{R}} t$ if $s|_\pi = \ell\sigma$ and $t = s[r\sigma]_\pi$, for a position $\pi \in \mathbf{s}(s)$, a rule $\ell \to r \in \mathcal{R}$, and a substitution $\sigma$.*

Note that a term is in normal form w.r.t. $\xrightarrow{\mathbf{s}}_{\mathcal{R}}$ iff it is in normal form w.r.t. $\to_{\mathcal{R}}$, under the assumption that $\mathbf{s}(t) \neq \varnothing$ iff $\mathrm{pos}^{\mathrm{red}}_{\mathcal{R}}(t) \neq \varnothing$. This condition holds for all evaluation strategies we are going to introduce in this section.

Definition 3.2.46 – Full Rewriting.

*Let $\mathcal{R}$ be a TRS and let $t \in \mathcal{T}$. The full evaluation strategy is defined as the function $\mathbf{f} : \mathcal{T} \to \mathrm{Pow}(\mathbb{N}^*)$ with $\mathbf{f}(t) = \mathrm{pos}^{\mathrm{red}}_{\mathcal{R}}(t)$. The full rewrite relation is denoted by $\xrightarrow{\mathbf{f}}_{\mathcal{R}} = \to_{\mathcal{R}}$.*

One of the most common evaluation strategies is *innermost rewriting*. Here, one has to rewrite at an innermost redex, i.e., if we rewrite at position $\pi$, then there cannot be another redex at a position $\pi'$ strictly below $\pi$. Innermost rewriting corresponds to a call-by-value evaluation strategy, which is a common evaluation strategy in programming languages. Moreover, it is often easier to analyze innermost rewriting than full rewriting, where one allows arbitrary evaluations. This holds for both termination and complexity analysis.

Definition 3.2.47 – Innermost Rewriting.

*Let $\mathcal{R}$ be a TRS and let $t \in \mathcal{T}$. The innermost evaluation strategy is defined as the function $\mathbf{i} : \mathcal{T} \to \mathrm{Pow}(\mathbb{N}^*)$ with $\mathbf{i}(t) = \{\pi \in \mathrm{pos}^{\mathrm{red}}_{\mathcal{R}}(t) \mid \neg\exists \pi' \in \mathrm{pos}^{\mathrm{red}}_{\mathcal{R}}(t).\ \pi < \pi'\}$. The innermost rewrite relation is denoted by $\xrightarrow{\mathbf{i}}_{\mathcal{R}}$.*

Example 3.2.48. Reconsider the TRS $\mathcal{R}_{\mathsf{plus}}$ from Example 3.2.30. We have the innermost rewrite sequence

$$\mathsf{plus}(\mathsf{plus}(0,0), \underline{\mathsf{plus}}(0,0)) \xrightarrow{\mathbf{i}}_{\mathcal{R}_{\mathsf{plus}}} \mathsf{plus}(\underline{\mathsf{plus}}(0,0), 0) \xrightarrow{\mathbf{i}}_{\mathcal{R}_{\mathsf{plus}}} \underline{\mathsf{plus}}(0,0) \xrightarrow{\mathbf{i}}_{\mathcal{R}_{\mathsf{plus}}} 0.$$

Another important evaluation strategy is *outermost rewriting*. Outermost rewriting corresponds to a call-by-name or lazy evaluation strategy, which is used in programming languages like Haskell.

Definition 3.2.49 – Outermost Rewriting.

*Let $\mathcal{R}$ be a TRS and let $t \in \mathcal{T}$. The outermost evaluation strategy is defined as the function $\mathbf{o} : \mathcal{T} \to \mathrm{Pow}(\mathbb{N}^*)$ with $\mathbf{o}(t) = \{\pi \in \mathrm{pos}^{\mathrm{red}}_{\mathcal{R}}(t) \mid \neg\exists \pi' \in \mathrm{pos}^{\mathrm{red}}_{\mathcal{R}}(t).\ \pi' < \pi\}$. The outermost rewrite relation is denoted by $\xrightarrow{\mathbf{o}}_{\mathcal{R}}$.*

Example 3.2.50. Reconsider the TRS $\mathcal{R}_{\mathsf{plus}}$ from Example 3.2.30. An outermost rewrite sequence that is not an innermost rewrite sequence is given by

$$\mathsf{plus}(\underline{\mathsf{plus}}(0,0), \mathsf{plus}(0,0)) \xrightarrow{\mathbf{o}}_{\mathcal{R}_{\mathsf{plus}}} \underline{\mathsf{plus}}(0, \mathsf{plus}(0,0)) \xrightarrow{\mathbf{o}}_{\mathcal{R}_{\mathsf{plus}}} \underline{\mathsf{plus}}(0,0) \xrightarrow{\mathbf{o}}_{\mathcal{R}_{\mathsf{plus}}} 0.$$

Note that the first rewrite step is outermost, because we have no left-hand side that matches the whole term $\mathsf{plus}(\mathsf{plus}(0,0), \mathsf{plus}(0,0))$ even though there is a $\mathsf{plus}$ function symbol at the root.

Finally, one can further restrict the rewrite relation based on the relation $\prec$ on orthogonal positions, e.g., by only allowing rewriting at the leftmost innermost redex. This is called *leftmost-innermost rewriting*.

Definition 3.2.51 – Leftmost-Innermost Rewriting.

*Let $\mathcal{R}$ be a TRS and let $t \in \mathcal{T}$. The leftmost-innermost evaluation strategy is defined as the function $\mathbf{li} : \mathcal{T} \to \mathrm{Pow}(\mathbb{N}^*)$ with $\mathbf{li}(t) = \{\pi \in \mathbf{i}(t) \mid \neg\exists \pi' \in \mathbf{i}(t).\ \pi' \prec \pi\}$. The leftmost-innermost rewrite relation is denoted by $\xrightarrow{\mathbf{li}}_{\mathcal{R}}$.*

Example 3.2.52. Reconsider the TRS $\mathcal{R}_{\mathsf{plus}}$ from Example 3.2.30. The rewrite sequences from Example 3.2.48 and Example 3.2.50 are not leftmost-innermost. Instead, a leftmost-innermost rewrite sequence is given by

$$\mathsf{plus}(\underline{\mathsf{plus}}(0,0),\mathsf{plus}(0,0)) \xrightarrow{\mathbf{li}}_{\mathcal{R}_{\mathsf{plus}}} \mathsf{plus}(0,\underline{\mathsf{plus}}(0,0)) \xrightarrow{\mathbf{li}}_{\mathcal{R}_{\mathsf{plus}}} \underline{\mathsf{plus}}(0,0) \xrightarrow{\mathbf{li}}_{\mathcal{R}_{\mathsf{plus}}} 0.$$

Note that we have $|\mathbf{li}(t)| = 1$ if $t \notin \mathtt{NF}_{\mathcal{R}}$ and $|\mathbf{li}(t)| = 0$ if $t \in \mathtt{NF}_{\mathcal{R}}$. Leftmost-innermost rewriting can therefore be seen as a memoryless scheduler for the choice of the position. Still, the choice of the rewrite rule remains.

Apart from the obvious relations between those strategies w.r.t. termination and complexity, e.g., $\mathtt{SN}(\xrightarrow{\mathbf{i}}_{\mathcal{R}}) \implies \mathtt{SN}(\xrightarrow{\mathbf{li}}_{\mathcal{R}})$, there exist further non-trivial relations that are useful for termination analysis. We recapitulate these relations in Chapter 4, where we present novel adaptations of such relations to the probabilistic setting.

### 3.2.4 Non-Termination Witnesses

Next, we show that we can restrict ourselves to specific rewrite sequences for analyzing termination. Namely, most of the time we can restrict ourselves to rewrite sequences starting with ground terms, and moreover, ensure specific structural properties of the terms in an infinite rewrite sequence. In the end, we show that the latter result can be tightened for innermost rewriting, which is one of the reasons why innermost rewriting is the easiest strategy to analyze in terms of termination.

First, it often suffices to only regard sequences of ground terms. This does not generally hold, as shown in Counterexample 3.2.53, but we can extend the signature $\Sigma$ with a *fresh* constant $\mathsf{c} \notin \Sigma \cup \mathcal{V}$ so that termination of $\mathcal{R}$ over the signature $\Sigma$ is equivalent to *ground termination* of $\mathcal{R}$ over the signature $\Sigma \uplus \{\mathsf{c}\}$.

Counterexample 3.2.53 (Ground Termination, [GM03]). Consider the TRS $\mathcal{R}$ consisting of the rewrite rule $\mathsf{f}(x) \to \mathsf{f}(x)$ over the signature $\Sigma = \Sigma_1 = \{\mathsf{f}\}$. $\mathcal{R}$ is not terminating, because of the rewrite sequence $\mathsf{f}(x) \to_{\mathcal{R}} \mathsf{f}(x) \to_{\mathcal{R}} \cdots$, but $\mathcal{R}$ is ground terminating because there are no ground terms for $\mathcal{R}$, as there exists no constant.

Theorem 3.2.54 – Ground Termination, [GM03].

Let $\mathcal{R}$ be a TRS over the signature $\Sigma$. Let $\mathsf{c}$ be a fresh constant, i.e., $\mathsf{c} \notin \Sigma \cup \mathcal{V}$. We say that a TRS is ground terminating if every rewrite sequence of ground terms is finite. Then $\mathcal{R}$ is terminating over the signature $\Sigma$ iff $\mathcal{R}$ is ground terminating over the signature $\Sigma \uplus \{\mathsf{c}\}$.

*Proof.* Replacing $\mathsf{c}$ by a fresh variable turns every ground rewrite sequence over $\Sigma \uplus \{\mathsf{c}\}$ into a rewrite sequence over $\Sigma$, so termination of $\mathcal{R}$ implies ground termination of $\mathcal{R}$. For the converse direction, let $t, t' \in \mathcal{T}(\Sigma, \mathcal{V})$. If $t \to_{\mathcal{R}} t'$, then $t\sigma \to_{\mathcal{R}} t'\sigma$, where $\sigma(x) = \mathsf{c}$ for all $x \in \mathcal{V}(t)$ (i.e., both $t\sigma$ and $t'\sigma$ are ground terms). Hence, if a term $t$ starts an infinite rewrite sequence, then so does the ground term $t\sigma$, so that ground termination of $\mathcal{R}$ implies termination of $\mathcal{R}$. *Proof* *QED*

Second, for full rewriting we have the following *minimality property*. Whenever a term $t$ starts an infinite rewrite sequence, then there exists a *minimal* infinite rewrite sequence beginning with $t$, where one only reduces redexes whose proper subterms are strongly normalizing. Moreover, there is such a sequence whose initial term is an instance $\ell\sigma$ of a left-hand side such that all proper subterms of $\ell\sigma$ are strongly normalizing as well.

Theorem 3.2.55 – Witness for Non-Termination, [AG00].

Let $\mathcal{R}$ be a TRS. If $\mathcal{R}$ is not terminating, then there exists an infinite rewrite sequence $t_0 \to_\mathcal{R} t_1 \to_\mathcal{R} t_2 \to_\mathcal{R} \cdots$ such that $t_0 = \ell\sigma$ for some rule $\ell \to r \in \mathcal{R}$ and some substitution $\sigma$, such that every proper subterm of $\ell\sigma$ is strongly normalizing w.r.t. $\mathcal{R}$, and moreover, one only reduces redexes whose proper subterms are strongly normalizing.

*Proof.* Let $t_0 \to_\mathcal{R} t_1 \to_\mathcal{R} t_2 \to_\mathcal{R} \cdots$ be an infinite rewrite sequence. Let $t_0' \trianglelefteq t_0$ such that all proper subterms of $t_0'$ are strongly normalizing and $t_0'$ itself is not strongly normalizing. If $t_0$ has multiple such subterms, choose $t_0'$ to be minimal w.r.t. $\triangleleft$. Thus, $t_0'$ starts an infinite rewrite sequence $t_0' \to_\mathcal{R} t_1' \to_\mathcal{R} t_2' \to_\mathcal{R} \cdots$ and all proper subterms of $t_0'$ are strongly normalizing.
Again, let $t_1'' \trianglelefteq t_1'$ be minimal w.r.t. $\triangleleft$ such that all proper subterms of $t_1''$ are strongly normalizing and $t_1''$ itself is not strongly normalizing. Thus, $t_1''$ starts an infinite rewrite sequence $t_1'' \to_\mathcal{R} t_2'' \to_\mathcal{R} t_3'' \to_\mathcal{R} \cdots$ and all proper subterms of $t_1''$ are strongly normalizing. We now have $t_0' \,(\to_\mathcal{R} \circ \trianglerighteq)\, t_1'' \to_\mathcal{R} t_2'' \to_\mathcal{R} t_3'' \to_\mathcal{R} \cdots$. Following the same argumentation, we can construct an infinite $(\to_\mathcal{R} \circ \trianglerighteq)$-rewrite sequence inductively. Let us denote this rewrite sequence by $s_0 \,(\to_\mathcal{R} \circ \trianglerighteq)\, s_1 \,(\to_\mathcal{R} \circ \trianglerighteq) \cdots$. Note that every proper subterm of every term in this $(\to_\mathcal{R} \circ \trianglerighteq)$-rewrite sequence is strongly normalizing. Therefore, we must eventually rewrite at the root of some $s_i$. Then, $s_i \,(\to_\mathcal{R} \circ \trianglerighteq)\, s_{i+1} \,(\to_\mathcal{R} \circ \trianglerighteq) \cdots$ gives rise to a $\to_\mathcal{R}$-rewrite sequence that satisfies our conditions. *QED*

If we consider only innermost rewriting, then the restriction to ground terms requires an additional fresh unary function symbol $\mathsf{h}$ so that innermost termination of $\mathcal{R}$ over the signature $\Sigma$ is equivalent to *ground innermost termination* of $\mathcal{R}$ over the signature $\Sigma \uplus \{\mathsf{c}, \mathsf{h}\}$. The two new fresh function symbols are needed to create arbitrarily many different ground terms in order to handle non-left-linear rewrite rules.

Counterexample 3.2.56 (Innermost Ground Termination, [GM03]). Consider the TRS $\mathcal{R}$ consisting of the two rewrite rules (3.5) and (3.6) over the signature $\Sigma = \Sigma_0 \uplus \Sigma_1$ with $\Sigma_0 = \{\mathsf{a}\}$ and $\Sigma_1 = \{\mathsf{f}\}$.

$$\mathsf{f}(\mathsf{f}(x)) \to \mathsf{f}(\mathsf{f}(x)) \qquad (3.5) \qquad\qquad \mathsf{f}(\mathsf{a}) \to \mathsf{a} \qquad (3.6)$$

Here, $\mathcal{R}$ is innermost ground terminating as the only possible ground terms have the form $\mathsf{f}^n(\mathsf{a})$ and thus, we have to rewrite the proper subterm $\mathsf{f}(\mathsf{a})$ before we can actually use rule (3.5). In comparison to that, $\mathcal{R}$ is not innermost terminating as we have the infinite innermost rewrite sequence $\mathsf{f}(\mathsf{f}(x)) \xrightarrow{\mathsf{i}}_\mathcal{R} \mathsf{f}(\mathsf{f}(x)) \xrightarrow{\mathsf{i}}_\mathcal{R} \cdots$ that will never reach a normal form.

Theorem 3.2.57 – Innermost Ground Termination, [GM03].

Let $\mathcal{R}$ be a TRS over the signature $\Sigma$. Let $\mathsf{c}$ be a fresh constant, and $\mathsf{h}$ be a fresh unary function symbol. We say that a TRS is ground innermost terminating iff every innermost rewrite sequence of ground terms is finite. Then $\mathcal{R}$ is innermost terminating over the signature $\Sigma$ iff $\mathcal{R}$ is ground innermost terminating over the

signature $\Sigma \uplus \{\mathsf{c}, \mathsf{h}\}$.

*Proof.* Let $t, t' \in \mathcal{T}(\Sigma, \mathcal{V})$ where the term $t$ has the variables $x_1, \ldots, x_m$. Then we have $t \xrightarrow{\mathtt{i}}_{\mathcal{R}} t'$ iff $t\sigma \xrightarrow{\mathtt{i}}_{\mathcal{R}} t'\sigma$, where $\sigma(x_i) = \mathsf{h}^i(\mathsf{c})$ for all $1 \leq i \leq m$ (i.e., both $t\sigma$ and $t'\sigma$ are ground terms). By induction, we can then prove that a term $t$ with variables $x_1, \ldots, x_m$ starts an infinite innermost rewrite sequence iff the ground term $t\sigma$ does. Hence, ground innermost termination over $\Sigma \uplus \{\mathsf{c}, \mathsf{h}\}$ implies innermost termination over $\Sigma$. For the converse direction, let $t$ be a ground term over $\Sigma \uplus \{\mathsf{c}, \mathsf{h}\}$ that starts an infinite innermost rewrite sequence $t = t_0 \xrightarrow{\mathtt{i}}_{\mathcal{R}} t_1 \xrightarrow{\mathtt{i}}_{\mathcal{R}} \cdots$ such that all proper subterms of $t$ are innermost terminating (i.e., $t$ is minimal w.r.t. $\lhd$). Since $\mathsf{c}$ and $\mathsf{h}$ do not occur in the rules of $\mathcal{R}$, we have $t \neq \mathsf{c}$, and $t \neq \mathsf{h}(u)$. So the root of $t$ is in $\Sigma$, and the sequence contains a rewrite step at the root, say $t_k = \ell\sigma \xrightarrow{\mathtt{i}}_{\mathcal{R}} t_{k+1}$, as otherwise one of the finitely many arguments of $t$ would start an infinite innermost rewrite sequence. Since all proper subterms of $\ell\sigma$ are normal forms, all subterms of $t_k$ with root $\mathsf{c}$ or $\mathsf{h}$ are normal forms. As $\mathsf{c}$ and $\mathsf{h}$ occur neither in left- nor in right-hand sides, every subterm of $t_{j+1}$ with root $\mathsf{c}$ or $\mathsf{h}$ is then a subterm of $t_j$ for all $j \geq k$, so these subterms remain normal forms and are never rewritten. Now we replace every maximal subterm with root $\mathsf{c}$ or $\mathsf{h}$ in all $t_j$ with $j \geq k$ by a variable, using the same variable for equal subterms and different variables for different subterms. This turns $t_k \xrightarrow{\mathtt{i}}_{\mathcal{R}} t_{k+1} \xrightarrow{\mathtt{i}}_{\mathcal{R}} \cdots$ into an infinite innermost rewrite sequence over $\Sigma$, where the steps remain innermost since a term is a normal form if one of its instances is a normal form. Hence, innermost termination over $\Sigma$ implies ground innermost termination over $\Sigma \uplus \{\mathsf{c}, \mathsf{h}\}$. *QED*

Finally, the witness theorem for innermost termination can be strengthened such that the first term $t_0$ in a sequence is the left-hand side of a rewrite rule with possibly instantiated variables and such that $t_0$ is an argument normal form. Termination analysis benefits from this normal form requirement immensely (see Chapter 7).

Theorem 3.2.58 – Witness for Innermost Non-Termination, [AG00].

Let $\mathcal{R}$ be a TRS. If $\mathcal{R}$ is not innermost terminating, then there exists an infinite innermost rewrite sequence $t_0 \xrightarrow{\mathtt{i}}_{\mathcal{R}} t_1 \xrightarrow{\mathtt{i}}_{\mathcal{R}} t_2 \xrightarrow{\mathtt{i}}_{\mathcal{R}} \cdots$ such that $t_0 = \ell\sigma \in \mathtt{ANF}_{\mathcal{R}}$ for some rule $\ell \to r \in \mathcal{R}$ and some substitution $\sigma$.

*Proof.* We prove the contrapositive, namely that if every innermost sequence $t_0 \xrightarrow{\mathtt{i}}_{\mathcal{R}} t_1 \xrightarrow{\mathtt{i}}_{\mathcal{R}} t_2 \xrightarrow{\mathtt{i}}_{\mathcal{R}} \cdots$ is finite where $t_0 = \ell\sigma \in \mathtt{ANF}_{\mathcal{R}}$ for some rule $\ell \to r \in \mathcal{R}$ and some substitution $\sigma$, then $\mathcal{R}$ is innermost terminating. We prove this by structural induction over the start term $t_0$.
If $t_0 \in \mathtt{NF}_{\mathcal{R}}$, then obviously, every innermost sequence that starts with this term is finite. If $t_0 = \ell\sigma \in \mathtt{ANF}_{\mathcal{R}}$ for some rule $\ell \to r \in \mathcal{R}$ and some substitution $\sigma$, then we know by our assumption that every innermost sequence starting with $t_0$ is finite.
Now we regard the induction step, and assume that $t_0 = f(q_1, \ldots, q_k)$. Due to the innermost evaluation strategy, we can only rewrite at the root position if every proper subterm is in normal form. Hence, we first prove that every sequence $t_0 \xrightarrow{\mathtt{i}\neg\varepsilon}_{\mathcal{R}} t_1 \xrightarrow{\mathtt{i}\neg\varepsilon}_{\mathcal{R}} t_2 \xrightarrow{\mathtt{i}\neg\varepsilon}_{\mathcal{R}} \cdots$ is finite, where $\xrightarrow{\mathtt{i}\neg\varepsilon}_{\mathcal{R}}$ denotes the restriction of $\xrightarrow{\mathtt{i}}_{\mathcal{R}}$ that prohibits rewrite steps at the root position. If there exists an infinite sequence $t_0 \xrightarrow{\mathtt{i}\neg\varepsilon}_{\mathcal{R}} t_1 \xrightarrow{\mathtt{i}\neg\varepsilon}_{\mathcal{R}} t_2 \xrightarrow{\mathtt{i}\neg\varepsilon}_{\mathcal{R}} \cdots$, then we have to rewrite infinitely often below a position $1 \leq j \leq k$. But this would mean that we can find an infinite innermost sequence starting with $q_j$, which is not possible by the induction hypothesis. Therefore, every sequence $t_0 \xrightarrow{\mathtt{i}\neg\varepsilon}_{\mathcal{R}} t_1 \xrightarrow{\mathtt{i}\neg\varepsilon}_{\mathcal{R}} t_2 \xrightarrow{\mathtt{i}\neg\varepsilon}_{\mathcal{R}} \cdots$ must be finite.

Next, we show that every innermost sequence $t_0 \xrightarrow{\mathtt{i}}_{\mathcal{R}} t_1 \xrightarrow{\mathtt{i}}_{\mathcal{R}} t_2 \xrightarrow{\mathtt{i}}_{\mathcal{R}} \cdots$ is finite. By our argumentation in the previous paragraph, we know that there is some $N \in \mathbb{N}$ such that $t_0 \xrightarrow{\mathtt{i}}_{\mathcal{R}} \cdots \xrightarrow{\mathtt{i}}_{\mathcal{R}} t_N$ and $t_N$ is in normal form w.r.t. $\xrightarrow{\mathtt{i}\neg\varepsilon}_{\mathcal{R}}$. Now, $t_N$ is either also in normal form w.r.t. $\mathcal{R}$, in which case the whole innermost sequence is finite, or we have $t_N = \ell\sigma \in \mathtt{ANF}_{\mathcal{R}}$ for some rule $\ell \to r \in \mathcal{R}$ and some substitution $\sigma$. By our assumption, we know that an innermost sequence that starts with $t_N$ is finite, and this means that our whole sequence $t_0 \xrightarrow{\mathtt{i}}_{\mathcal{R}} \cdots \xrightarrow{\mathtt{i}}_{\mathcal{R}} t_N \xrightarrow{\mathtt{i}}_{\mathcal{R}} \cdots$ must be finite. QED

### 3.2.5 Proving Termination and Complexity via Interpretations

We end this section with a first glimpse of what an automatic technique to prove termination of TRSs looks like. Up to now, we have only been working with syntactical objects (terms), and a rewrite system gives a syntactical meaning of a reduction step. Via $\Sigma$-*algebras* and $\Sigma$-*interpretations* we can give terms semantics in a specific *carrier*.

Definition 3.2.59 – $\Sigma$-Interpretation, $\Sigma$-Algebra.

*Let $\Sigma$ be a signature, and $\mathcal{V}$ be a set of variables. A $\Sigma$-interpretation is a triple $\mathfrak{I} = (I, \alpha, \beta)$. Here, $I$ is a non-empty set, called the carrier of $\mathfrak{I}$. Moreover, we have $\alpha = (\alpha_f)_{f\in\Sigma}$ with $\alpha_f : I^n \to I$ for $f \in \Sigma_n$. The function $\alpha_f$ is the meaning of the function symbol $f$ under the interpretation $\mathfrak{I}$. Lastly, $\beta : \mathcal{V} \to I$ is the variable assignment. Every $\Sigma$-interpretation $\mathfrak{I}$ induces a function $\mathfrak{I} : \mathcal{T}(\Sigma, \mathcal{V}) \to I$ by*

$$\begin{aligned} \mathfrak{I}(x) &= \beta(x) \text{ for all } x \in \mathcal{V}, \text{ and} \\ \mathfrak{I}(f(t_1, \ldots, t_n)) &= \alpha_f(\mathfrak{I}(t_1), \ldots, \mathfrak{I}(t_n)) \text{ for all } f \in \Sigma_n, t_1, \ldots, t_n \in \mathcal{T}(\Sigma, \mathcal{V}) \,. \end{aligned}$$

*We call $\mathfrak{I}(t)$ the interpretation of the term $t$ under $\mathfrak{I}$. A $\Sigma$-interpretation without variable assignment $(I, \alpha)$ is called a $\Sigma$-algebra, which maps every variable to itself, i.e., $\mathfrak{I}(x) = x$. Instead of $\alpha_f$ and $\beta(x)$ we simply write $f_{\mathfrak{I}}$ and $\mathfrak{I}(x)$, respectively.*

Termination only regards the syntactical rewrite relation of a TRS $\mathcal{R}$. By mapping terms into an *ordered* $\Sigma$-*algebra* $\mathfrak{I} = (I, \alpha)$ where the set $I$ has an additional well-founded order $\succ \subseteq I \times I$ such that a rewrite step $s \to_{\mathcal{R}} t$ implies a decrease in the order under the interpretation $\mathfrak{I}(s) \succ \mathfrak{I}(t)$, we can prove termination of the rewrite relation or even infer upper complexity bounds.

Definition 3.2.60 – Well-founded, Ordered $\Sigma$-Algebra.

*A relation $\succ \subseteq I \times I$ on some set $I$ is well founded if there exists no infinite sequence $a_0 \succ a_1 \succ \cdots$. An ordered $\Sigma$-algebra is a pair $(\mathfrak{I}, \succ)$ where $\mathfrak{I}$ is a $\Sigma$-algebra with carrier $I$ and $\succ$ is a well-founded order on $I$, i.e., $\succ$ is well founded and transitive. If the order is clear from the context, we simply write $\mathfrak{I}$ for $(\mathfrak{I}, \succ)$ again.*

We obtain a well-founded order on ground terms by considering $t_1$ to be greater than $t_2$ if $\mathfrak{I}(t_1) \succ \mathfrak{I}(t_2)$. We extend this to a well-founded order on terms with variables by defining $\mathfrak{I}(t_1) \succ \mathfrak{I}(t_2)$ if $\mathfrak{I}(t_1)\sigma \succ \mathfrak{I}(t_2)\sigma$ holds for all instantiations $\sigma : \mathcal{V} \to I$ that replace the variables of $t_1$ and $t_2$ by elements of the carrier $I$. By $\succcurlyeq \subseteq I \times I$ we denote the reflexive closure of $\succ$, and we extend it to terms with variables in the same way, i.e., $\mathfrak{I}(t_1) \succcurlyeq \mathfrak{I}(t_2)$ if $\mathfrak{I}(t_1)\sigma \succcurlyeq \mathfrak{I}(t_2)\sigma$ for all such instantiations $\sigma$.

Suppose that we can find an ordered $\Sigma$-algebra, and we can show that whenever we perform a rewrite step $s \to_{\mathcal{R}} t$, then we have a strict decrease in the interpretation

of those terms $\mathfrak{I}(s) \succ \mathfrak{I}(t)$. Then, we have proven that $\mathcal{R}$ is terminating, i.e., $\to_{\mathcal{R}}$ is well founded as well. The function $t \mapsto \mathfrak{I}(t)$ is then called a *ranking function*. However, this requires considering all the (possibly infinitely many) rewrite steps. Instead, we would like to only consider the (often finitely many) rewrite rules. To do so, we have to additionally enforce that the algebra is *monotonic* (*closed under contexts*).

Definition 3.2.61 – Monotonicity, Closed Under Contexts.

*We say that* $(\mathfrak{I}, \succ)$ *is monotonic or closed under contexts if* $\mathfrak{I}(t_1) \succ \mathfrak{I}(t_2)$ *implies* $\mathfrak{I}(C[t_1]) \succ \mathfrak{I}(C[t_2])$*, and weakly monotonic if* $\mathfrak{I}(t_1) \succcurlyeq \mathfrak{I}(t_2)$ *implies* $\mathfrak{I}(C[t_1]) \succcurlyeq \mathfrak{I}(C[t_2])$ *for every context* $C$*.*

Using a monotonic $\Sigma$-algebra yields the following result.

Theorem 3.2.62 – Proving Termination Via Interpretations.

Let $\mathcal{R}$ be a TRS, and let $(\mathfrak{I}, \succ)$ be a monotonic, ordered $\Sigma$-algebra. If (1) is satisfied, then $\mathcal{R}$ is terminating.

(1) $\quad \forall \ell \to r \in \mathcal{R}.\ \mathfrak{I}(\ell) \succ \mathfrak{I}(r)$.

*Proof.* We start by showing that the condition (1) of the theorem extends to rewrite steps instead of just rules: If $s, t \in \mathcal{T}$ with $s \to_{\mathcal{R}} t$, then $\mathfrak{I}(s) \succ \mathfrak{I}(t)$.
Let $s, t \in \mathcal{T}$ with $s \to_{\mathcal{R}} t$. Then there exist a rule $\ell \to r \in \mathcal{R}$ with $\mathfrak{I}(\ell) \succ \mathfrak{I}(r)$ (by (1)), a substitution $\sigma$, and a position $\pi$ such that $s|_\pi = \ell\sigma$, $t = s[r\sigma]_\pi$.
We perform induction on $\pi$. So in the induction base, let $\pi = \varepsilon$. Hence, we have $s = \ell\sigma$ and $t = r\sigma$. By assumption we have $\mathfrak{I}(\ell) \succ \mathfrak{I}(r)$. As these inequations hold for all instantiations of the occurring variables, we have $\mathfrak{I}(s) = \mathfrak{I}(\ell\sigma) \succ \mathfrak{I}(r\sigma) = \mathfrak{I}(t)$.
In the induction step, we have $\pi = i.\pi'$, $s = f(s_1, \dots, s_i, \dots, s_n)$ and $s_i \to_{\mathcal{R}} t_i$. Then by the induction hypothesis we have $\mathfrak{I}(s_i) \succ \mathfrak{I}(t_i)$. For $t = f(s_1, \dots, t_i, \dots, s_n)$ we obtain

$$\begin{aligned}
\mathfrak{I}(s) &= \mathfrak{I}(f(s_1, \dots, s_i, \dots, s_n)) \\
&= f_{\mathfrak{I}}(\mathfrak{I}(s_1), \dots, \mathfrak{I}(s_i), \dots, \mathfrak{I}(s_n)) \\
&\qquad \downarrow \quad \text{by monotonicity and } \mathfrak{I}(s_i) \succ \mathfrak{I}(t_i) \\
&\succ f_{\mathfrak{I}}(\mathfrak{I}(s_1), \dots, \mathfrak{I}(t_i), \dots, \mathfrak{I}(s_n)) \\
&= \mathfrak{I}(f(s_1, \dots, t_i, \dots, s_n)) \\
&= \mathfrak{I}(t).
\end{aligned}$$

Now, assume that $\mathcal{R}$ is not terminating. Then there exists an infinite rewrite sequence $t_0 \to_{\mathcal{R}} t_1 \to_{\mathcal{R}} t_2 \to_{\mathcal{R}} \cdots$. But then for any instantiation $\sigma : \mathcal{V} \to I$, we obtain the infinite descending sequence $\mathfrak{I}(t_0)\sigma \succ \mathfrak{I}(t_1)\sigma \succ \mathfrak{I}(t_2)\sigma \succ \cdots$ in $I$, which is a contradiction to $\succ$ being well founded. Hence, $\mathcal{R}$ must be terminating. *QED*

Examples of $\Sigma$-algebras used for termination analysis are variants of *polynomial interpretations* [Lan75, Lan79] or variants of *matrix interpretations* [EWZ06, HW06, CGP10]. We mainly use polynomial interpretations in this thesis, but matrix interpretations are important algebras in practice as well. Additionally, for simplicity we only consider natural polynomial interpretations, i.e., the coefficients in our polynomials range over the natural numbers. For techniques on other types of polynomials, e.g., ranging over the rationals or reals, see [Luc05, Luc07, Luc10]. However, note that while our interpretations range over the natural numbers, due to the occurring probabilities in later chapters, we still have to consider orders on real numbers, e.g., multiplying the natural polynomial $x$ with the probability $1/2$ yields the polynomial $1/2 \cdot x$ with real coefficients.

**Definition 3.2.63** – Polynomial Interpretation.

*A polynomial interpretation* Pol *is a* $\Sigma$*-algebra with carrier set* $\mathbb{N}$ *which maps every function symbol* $f \in \Sigma_n$ *to a polynomial* $f_{\mathrm{Pol}} \in \mathbb{N}[x_1, \ldots, x_n]$*. The order* $> \subseteq \mathbb{N}[\mathcal{V}] \times \mathbb{N}[\mathcal{V}]$ *is defined by* $p(x_1, \ldots, x_n) > p'(x_1, \ldots, x_n)$ *if* $p(a_1, \ldots, a_n) > p'(a_1, \ldots, a_n)$ *for all* $a_1, \ldots, a_n \in \mathbb{N}$*.*

To prove termination automatically, one has to find a mapping to monotonic polynomials with natural numbers as coefficients such that for every rewrite rule the left-hand side is strictly greater than the right-hand side. An easy condition on the interpretation that ensures monotonicity is mapping each function symbol $f \in \Sigma_n$ to a polynomial $f_{\mathrm{Pol}}(x_1, \ldots, x_n) = x_1 + \cdots + x_n + p(x_1, \ldots, x_n)$ with $p(x_1, \ldots, x_n) \in \mathbb{N}[x_1, \ldots, x_n]$, i.e., we ensure monotonicity by requiring that the coefficient of the monomial $x_i$ is greater than or equal to 1 for every $1 \leq i \leq n$.

**Corollary 3.2.64** – Proving Termination Via Polynomial Interpretations.

*Let* $\mathcal{R}$ *be a TRS, and let* $\mathrm{Pol} : \mathcal{T} \to \mathbb{N}[\mathcal{V}]$ *be a monotonic polynomial interpretation. If (1) holds, then* $\mathcal{R}$ *is terminating.*

(1) $\forall \ell \to r \in \mathcal{R}.\ \mathrm{Pol}(\ell) > \mathrm{Pol}(r)$.

*Proof.* Follows from Theorem 3.2.62. *Proof QED*

Corollary 3.2.64 can then be automated by searching for the coefficients using an SMT solver.

Example 3.2.65. Consider the TRS $\mathcal{R}_{\mathsf{plus}}$ from Example 3.2.30. A polynomial interpretation that satisfies the condition of Corollary 3.2.64 for every rule of $\mathcal{R}_{\mathsf{plus}}$ is Pol with $\mathsf{s}_{\mathrm{Pol}}(x) = x + 1$, $\mathsf{plus}_{\mathrm{Pol}}(x, y) = 2x + y + 1$, and $\mathsf{0}_{\mathrm{Pol}} = 0$. Now, we have

$$\begin{array}{rrcl} & \mathrm{Pol}(\mathsf{plus}(\mathsf{0}, y)) & > & \mathrm{Pol}(y) \\ \Leftrightarrow & 2 \cdot 0 + y + 1 & > & y \\ \Leftrightarrow & y + 1 & > & y \end{array} \qquad \begin{array}{rrcl} & \mathrm{Pol}(\mathsf{plus}(\mathsf{s}(x), y)) & > & \mathrm{Pol}(\mathsf{s}(\mathsf{plus}(x, y))) \\ \Leftrightarrow & 2 \cdot (x + 1) + y + 1 & > & (2x + y + 1) + 1 \\ \Leftrightarrow & 2x + y + 3 & > & 2x + y + 2 \end{array}$$

Example 3.2.66 (Termination with Data Structures). Consider the TRS $\mathcal{R}_{\mathsf{len}}$ from Example 3.2.31. We can prove termination with the polynomial interpretation Pol with $\mathsf{len}_{\mathrm{Pol}}(x) = x + 1$, $\mathsf{s}_{\mathrm{Pol}}(x) = x$, $\mathsf{cons}_{\mathrm{Pol}}(x, y) = x + y + 1$, $\mathsf{nil}_{\mathrm{Pol}} = 0$, $\mathsf{0}_{\mathrm{Pol}} = 0$, and $\mathsf{plus}_{\mathrm{Pol}}(x, y) = x + y$.

Given a termination proof as in Corollary 3.2.64, we can additionally infer upper bounds on the derivational complexity and on the runtime complexity of the TRS $\mathcal{R}$. Note that the polynomial interpretation $\mathrm{Pol}_0(t)$[4] of a term $t$ yields an upper bound on the derivation height of $t$ where $\mathrm{Pol}_0$ denotes the interpretation that behaves like Pol, and in addition, maps every variable to 0. So we have $\mathrm{dh}_{\to_{\mathcal{R}}}(t) \leq \mathrm{Pol}_0(t)$. It remains to analyze the size of $\mathrm{Pol}_0(t)$ for terms of size $\leq n$.

While arbitrary monotonic polynomial interpretations can be used to prove termination of TRSs [Lan79], a special form, namely *CPIs*, suffices to infer a polynomial runtime bound from such a termination proof [BCMT01, HL89].

[4] Polynomial interpretations are denoted by Pol, while polynomial complexities are denoted $\mathbf{Pol}_i$.

Definition 3.2.67 – Complexity Polynomial Interpretation.

*Let* Pol *be a polynomial interpretation. We say* Pol *is a complexity polynomial interpretation (CPI) if for all constructors* $f \in \mathcal{C}$ *we have* $f_{\mathrm{Pol}}(x_1, \ldots, x_k) = a_1x_1 + \ldots + a_kx_k + b$*, where* $b \in \mathbb{N}$ *and* $a_j \in \{0, 1\}$ *for all* $1 \leq j \leq k$*.*

For monotonic CPIs we always have $a_j = 1$, but in Chapter 7 we will consider weakly monotonic CPIs, where $a_j \in \{0, 1\}$ is possible.

In the following, we analyze the size of $\mathrm{Pol}_0(t)$ for every term $t$ and every possible polynomial interpretation Pol [BCMT01, HL89]. We fix a signature $\Sigma = \mathcal{C} \uplus \mathcal{D}$ with defined symbols and constructors w.r.t. a TRS $\mathcal{R}$. Here, a statement like $\mathrm{Pol}_0(s) \in \mathcal{O}(f(|s|))$ for all terms $s$ of a set $M \subseteq \mathcal{T}$ means that the function $n \mapsto \sup\{\mathrm{Pol}_0(s) \mid s \in M, |s| \leq n\}$ is in $\mathcal{O}(f(n))$.

**Size of Constructor Terms** Let Pol be a CPI. Recall that a CPI guarantees that $\mathrm{Pol}(f(x_1, \ldots, x_k)) = a_1x_1 + \ldots + a_kx_k + b$, with $a_j \in \{0, 1\}$ and $b \in \mathbb{N}$ for all constructors $f \in \mathcal{C}$. Let $b_{max}$ be the maximum of all $b$ for all constructors $f \in \mathcal{C}$. We show that $\mathrm{Pol}_0(s) \leq b_{max} \cdot |s|$ holds for every constructor term $s \in \mathcal{T}(\mathcal{C}, \mathcal{V})$ by induction on the structure of $s$. If $s = x \in \mathcal{V}$ is a variable, then $\mathrm{Pol}_0(s) = 0 \leq b_{max} \cdot |s|$, since $\mathrm{Pol}_0$ maps every variable to 0. Otherwise, we have $s = f(s_1, \ldots, s_k)$ for some constructor $f \in \mathcal{C}$, where $k = 0$ is possible (i.e., $s$ is a constant), and thus $|s| = 1 + \sum_{1 \leq j \leq k} |s_j|$. Using $a_j \in \{0, 1\}$ and the induction hypothesis, we obtain

$$\begin{aligned}\mathrm{Pol}_0(s) \;&=\; \sum_{1 \leq j \leq k} a_j \cdot \mathrm{Pol}_0(s_j) \;+\; b \;\leq\; \sum_{1 \leq j \leq k} \mathrm{Pol}_0(s_j) \;+\; b_{max} \\ &\leq\; b_{max} \cdot \sum_{1 \leq j \leq k} |s_j| \;+\; b_{max} \;=\; b_{max} \cdot |s|.\end{aligned}$$

Next, let Pol not be a CPI but linear on constructors, i.e., $\mathrm{Pol}(f(x_1, \ldots, x_k)) = a_1x_1 + \ldots + a_kx_k + b$, with $a_1, \ldots, a_k, b \in \mathbb{N}$ for all constructors $f \in \mathcal{C}$. Let $b_{max}$ be the maximum of all $b$, let $a_{max}$ be the maximum of all $a$, and let $\alpha = \max\{a_{max} + 1, 2\}$. Again by induction on the structure of $s$, we obtain $\mathrm{Pol}_0(s) \leq b_{max} \cdot (\alpha^{|s|} - 1)$. For $|s| = 1$, i.e., if $s$ is a variable or a constant, we have $\mathrm{Pol}_0(s) \leq b_{max} \leq b_{max} \cdot (\alpha - 1) = b_{max} \cdot (\alpha^{|s|} - 1)$ since $\alpha \geq 2$. For $s = f(s_1, \ldots, s_k)$ with $k \geq 1$, and hence $|s| \geq 2$, we get

$$\begin{aligned}\mathrm{Pol}_0(s) \;&\leq\; a_{max} \cdot \sum_{1 \leq j \leq k} \mathrm{Pol}_0(s_j) \;+\; b_{max} \\ &\overset{\text{IH}}{\leq}\; a_{max} \cdot \sum_{1 \leq j \leq k} \big(b_{max} \cdot (\alpha^{|s_j|} - 1)\big) \;+\; b_{max} \\ &\leq\; a_{max} \cdot b_{max} \cdot \sum_{1 \leq j \leq k} (\alpha^{|s_j|} - 1) \;+\; b_{max} \\ &\leq\; a_{max} \cdot b_{max} \cdot (\alpha^{\sum_{1 \leq j \leq k} |s_j|} - 1) \;+\; b_{max} \\ &\leq\; a_{max} \cdot b_{max} \cdot (\alpha^{|s|-1} - 1) \;+\; b_{max} \;\leq\; b_{max} \cdot (\alpha^{|s|} - 1),\end{aligned}$$

where the step $\sum_{1\leq j\leq k}(\alpha^{|s_j|}-1) \leq \alpha^{\sum_{1\leq j\leq k}|s_j|}-1$ holds as

$$\sum_{1\leq j\leq k}(\alpha^{n_j}-1) \;\leq\; \alpha^{\sum_{1\leq j\leq k} n_j}-1 \qquad \text{for all } n_j \geq 1 \text{ and } \alpha \geq 2.$$

It can be shown by induction on $k$ that this reduces to the case $(\alpha^m-1)+(\alpha^n-1) \leq \alpha^{m+n}-1$, which is equivalent to $\alpha^{m+n}-\alpha^m-\alpha^n+1 = (\alpha^m-1)\cdot(\alpha^n-1) \geq 0$, as $\alpha \geq 2$ and $m,n \geq 1$. The final inequality of the derivation additionally uses $a_{max} \leq \alpha-1$ (as $\alpha = \max\{a_{max}+1, 2\}$) and $\alpha^{|s|-1} \geq \alpha \geq 2$ (as $|s| \geq 2$). Hence, we have $\mathrm{Pol}_0(s) \in \mathcal{O}(2^{\mathrm{pol}(|s|)})$ for a linear polynomial $\mathrm{pol}(n)$ in $n$.

Finally, if Pol is not a CPI and not even linear on constructors, e.g., $\mathrm{Pol}(f(x_1,x_2)) = x_1\cdot x_2$ or $\mathrm{Pol}(f(x_1,x_2)) = x_1^2+x_2^2$, then we can only guarantee that $\mathrm{Pol}_0(s) \leq 2^{2^{\mathrm{pol}(|s|)}}$, for some polynomial $\mathrm{pol}(n)$ in $n$. To be precise, let $a_{max}$ be the maximum coefficient of all non-constant monomials in all interpretations $f_{\mathrm{Pol}}$ of constructors $f$, and let $b_{max}$ be the maximum constant monomial in all these interpretations. Moreover, let $c_{max}$ be the maximum degree of all $\mathrm{Pol}(f(x_1,\ldots,x_k))$, and let $N$ be the maximum number of monomials of all these interpretations. We bound $T(n) = \sup\{\mathrm{Pol}_0(s) \mid s \in \mathcal{T}(\mathcal{C},\mathcal{V}), |s| \leq n\}$. For $|s| \leq 1$ we again have $\mathrm{Pol}_0(s) \leq b_{max}$, and for $s = f(s_1,\ldots,s_k)$ with $|s| = n > 1$ we have $|s_j| \leq n-1$ for all $j$, so bounding every argument by $T(n-1)$ yields $T(n) \leq N\cdot a_{max}\cdot T(n-1)^{c_{max}} + b_{max}$. After conversion to homogeneous form (using $T(n-1) \geq 1$, which we can assume w.l.o.g. by replacing $T(n)$ with $\max\{T(n),1\}$), and taking the log on both sides, we obtain a first-order linear recurrence relation. Hence, we have $\log(\mathrm{Pol}_0(s)) \leq 2^{\mathrm{pol}(|s|)}$, for some polynomial $\mathrm{pol}(n)$, i.e., we have $\mathrm{Pol}_0(s) \in \mathcal{O}(2^{2^{\mathrm{pol}'(|s|)}})$, for some polynomial $\mathrm{pol}'(n)$.

**Size of Basic Terms** First, let Pol be a CPI. Again, let $b_{max}$ be the maximum of all $b$ for all constructors $f \in \mathcal{C}$. Moreover, let $t = f(t_1,\ldots,t_k) \in \mathcal{T}_{\mathcal{B}}$. Here, we have

$$\begin{aligned}\mathrm{Pol}_0(t) &= f_{\mathrm{Pol}}(\mathrm{Pol}_0(t_1),\ldots,\mathrm{Pol}_0(t_k))\\ &\leq f_{\mathrm{Pol}}(b_{max}\cdot|t_1|,\ldots,b_{max}\cdot|t_k|)\\ &\leq f_{\mathrm{Pol}}(b_{max}\cdot|t|,\ldots,b_{max}\cdot|t|).\end{aligned}$$

The right-hand side is a polynomial in $|t|$ of degree at most $a$, where $a$ is the degree of $f_{\mathrm{Pol}}$. Hence, we have $\mathrm{Pol}_0(t) \in \mathcal{O}(|t|^a)$.

If Pol is only linear on constructors, we get $\mathrm{Pol}_0(t) \in \mathcal{O}(2^{\mathrm{pol}(|t|)})$, and if Pol is not linear, we obtain $\mathrm{Pol}_0(t) \in \mathcal{O}(2^{2^{\mathrm{pol}(|t|)}})$ in a similar fashion.

**Size of Arbitrary Terms** If we have $\mathrm{Pol}(f(x_1,\ldots,x_k)) = a_1x_1+\ldots+a_kx_k+b$ with $a_j \in \{0,1\}, b \in \mathbb{N}$ for all $f \in \Sigma$ (and not only for all constructors $f \in \mathcal{C}$), then $\mathrm{Pol}_0(t) \in \mathcal{O}(|t|)$. Such interpretations are called *hard CPIs*. If Pol is linear for all $f \in \Sigma$, then $\mathrm{Pol}_0(t) \in \mathcal{O}(2^{\mathrm{pol}(|t|)})$, and if Pol is not linear, we obtain $\mathrm{Pol}_0(t) \in \mathcal{O}(2^{2^{\mathrm{pol}(|t|)}})$.

**Bounds on Complexity via Polynomials** All in all, we get the following corollary.

Corollary 3.2.68 – Upper Complexity Bounds Via Polynomial Interpretations, [BCMT01, HL89].

*Let $\mathcal{R}$ be a finite TRS, and let* $\mathrm{Pol} : \mathcal{T} \to \mathbb{N}[\mathcal{V}]$ *be a monotonic polynomial interpretation. If we have* $\forall \ell \to r \in \mathcal{R}.\ \mathrm{Pol}(\ell) > \mathrm{Pol}(r)$*, then we can infer the following upper bounds on the derivational complexity and the runtime complexity of* $\mathcal{R}$*.*

- *If* Pol *is a hard CPI, then* $\iota(\mathrm{dc}_\mathcal{R}) \sqsubseteq \mathbf{Pol}_1$ *and* $\iota(\mathrm{rc}_\mathcal{R}) \sqsubseteq \mathbf{Pol}_1$*.*
- *If* Pol *is a CPI, then* $\iota(\mathrm{rc}_\mathcal{R}) \sqsubseteq \mathbf{Pol}_a$*, where $a$ is the maximum degree of a polynomial $f_{\mathrm{Pol}}$ for some $f \in \mathcal{D}$.*
- *If* Pol *is linear on constructors, then* $\iota(\mathrm{rc}_\mathcal{R}) \sqsubseteq \mathbf{Exp}$*.*
- *If* Pol *is linear, then* $\iota(\mathrm{dc}_\mathcal{R}) \sqsubseteq \mathbf{Exp}$*.*
- *If* Pol *is arbitrary, then* $\iota(\mathrm{dc}_\mathcal{R}) \sqsubseteq \mathbf{2\text{-}Exp}$ *and* $\iota(\mathrm{rc}_\mathcal{R}) \sqsubseteq \mathbf{2\text{-}Exp}$*.*

SECTION 3.3

## Probabilistic Abstract Reduction Systems

In the following two sections, we introduce probabilistic rewriting—the probabilistic model of computation we want to analyze in this thesis. In this section, we consider the basic form of a *probabilistic abstract reduction system*, and in the next section, we introduce *probabilistic term rewrite systems*. We extend non-probabilistic rewriting, where the reduct is chosen non-deterministically based on the possible rules, by probabilistic choices. We do not simply replace the non-deterministic choice by a probabilistic one, as for Markov chains that are based on transition systems, see Section 2.4. Instead, we extend the non-deterministic rule selection by an additional probabilistic choice afterwards. So rule selection remains non-deterministic but after selecting a rule one has multiple probabilistic choices for the reduct. In this way, non-deterministic aspects and probabilistic aspects are intertwined, as for Markov decision processes, see Section 2.4. This idea of probabilistic rewriting was introduced in [BK02, BG05] and is widely used, e.g., [Gna07, DCM18, Fag22, ADLY20, KG25, KG26]. There also exists a notion of probabilistic abstract reduction systems that replaces the non-determinism of the rule selection completely by probabilities [KC18], which is not directly possible in our term rewriting setting.

We begin by recapitulating some basic definitions from [BK02, BG05, DCM18, Fag22, ADLY20, KG25]. In order to extend the rule selection by an additional probabilistic choice, we replace the right-hand sides (which were simple objects $a \in A$ for ARSs) by *multi-distributions* over $A$. A multi-distribution is a more fine-grained distribution, where objects $a \in A$ not only have a single probability, but instead may have multiple probabilities that can be treated separately. We need this more fine-grained distinction to be able to deal with the same object resulting from two different execution paths differently, see Example 3.3.9.

Definition 3.3.1 – Multiset.

*A multiset on a set $A$ is a function $c : A \to \mathbb{N}$. The multiset is finite if $\sum_{a \in A} c(a) < \infty$. We write $\{a_1, \dots, a_{1_{n_1}}, \dots, a_{k_1}, \dots, a_{k_{n_k}}\}$ for the multiset $c$ with $c(a_i) = n_i$ for all $1 \leq i \leq k$ and $c(a) = 0$ for all $a \notin \{a_1, \dots, a_k\}$. The union $c \uplus c'$ of two multisets $c$ and $c'$ is the multiset with $(c \uplus c')(a) = c(a) + c'(a)$ for all*

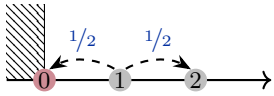
$a \in A$.

Definition 3.3.2 – Multi-Distribution.
*A (finite) multi-distribution* $\mu$ *on a set* $A \neq \varnothing$ *is a (finite) multiset of pairs* $(p, a)$ *denoted by* ${}^{p}a$*, where* $0 < p \leq 1$ *is a probability and* $a \in A$*, such that* $\sum_{{}^{p}a \in \mu} p \leq 1$*. Let* $\mathrm{Dist}(A)$ *denote the set of all multi-distributions on* $A$*, and let* $\mathrm{FDist}(A)$ *denote the set of all finite multi-distributions on* $A$*. For* $\mu \in \mathrm{Dist}(A)$*, its support is the set* $\mathrm{Supp}(\mu) = \{a \mid {}^{p}a \in \mu \text{ for some } p\}$*. Moreover, for every* $a \in A$*, we write* $\mu(a) = \sum_{{}^{p}a \in \mu} p$ *for the total probability of the object* $a$ *in* $\mu$*.*

Example 3.3.3. Examples of finite multi-distributions on $\mathbb{N}$ are $\{{}^{1/2}0, {}^{1/2}1\}$, $\{{}^{1/4}0, {}^{1/4}0, {}^{1/2}0\}$, and an example of an infinite multi-distribution is $\bigcup_{i=1}^{\infty}\{{}^{(1/2)^i}i-1\} = \{{}^{1/2}0, {}^{1/4}1, {}^{1/8}2, \ldots\}$. The latter is called a *geometric distribution*.

Definition 3.3.4 – Probabilistic Abstract Reduction System (PARS).
*A probabilistic abstract reduction system (PARS)* $(A, \to)$ *is a countable set* $A$ *(called the universe) together with a probabilistic relation* $\to \subseteq A \times \mathrm{FDist}(A)$ *such that* $\sum_{{}^{p}a \in \mu} p = 1$ *holds for every* $b \to \mu$*. A PARS* $(A, \to)$ *is non-probabilistic if* $\to \subseteq A \times \{\{{}^{1}a\} \mid a \in A\}$*. If* $A$ *is clear from the context, we write* $\to$ *for the PARS. Additionally, if the relation has an index as in* $\to_\alpha$*, then we sometimes refer to* $\to_\alpha$ *simply as* $\alpha$*.*

Example 3.3.5. We use the following two PARSs as running examples for this section. The first one is $\to_{\mathsf{rw}} = \{n+1 \to \{{}^{1/2}n, {}^{1/2}n+2\} \mid n \in \mathbb{N}\}$ on $\mathbb{Z}$, and models a symmetric random walk program. See Figure 3.2 for an illustration of the random walk modeled by $\to_{\mathsf{rw}}$. As usual, the random walk program terminates once it hits a number below or equal to zero. The second models a coin flip that terminates once we flip heads, and keeps on going if we flip tails. We model it via the PARS $(\{\mathsf{head}, \mathsf{tail}\}, \to_{\mathsf{coin}})$ with $\to_{\mathsf{coin}} = \{\mathsf{tail} \to \{{}^{1/2}\mathsf{tail}, {}^{1/2}\mathsf{head}\}\}$, see Figure 3.3 for an illustration of $\to_{\mathsf{coin}}$ via a finite Markov chain.




$\mu_{\mathsf{rw}}(-1) = {}^{1/2}$
$\mu_{\mathsf{rw}}(1) = {}^{1/2}$

Figure 3.2: The PARS $\to_{\mathsf{rw}}$ illustrated as a random walk, corresponding to the symmetric random walk program $\mu_{\mathsf{rw}} = \{{}^{1/2}-1, {}^{1/2}1\}$.

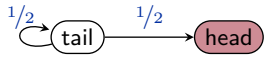



Figure 3.3: The PARS $\to_{\mathsf{coin}}$ illustrated as a Markov chain.

While Markov chains and Markov decision processes evolve infinitely, we always have a fixed set of terminal states in mind. To highlight these terminal states we color them in red and omit the trivial self loops for those states. In illustrations via random walks, it is important to denote the measure $\|.\| : A \to \mathbb{Z}$ that maps each object to its position in the random walk. In the case of Example 3.3.5 the measure $\|.\| : \mathbb{Z} \to \mathbb{Z}$ is simply the identity $n \mapsto n$.

Normal forms are defined as for ARSs. These are elements $a \in A$ such that there is no $\mu \in \mathrm{FDist}(A)$ with $a \to \mu$, i.e., we cannot perform any reduction steps anymore. In general, we can lift every definition regarding ARSs to PARSs, if they only consider the left-hand sides of rules, because left-hand sides remain single objects.

Definition 3.3.6 – Normal Form.
*Let $(A, \to)$ be a PARS. An element $a \in A$ such that there is no $\mu \in \mathrm{FDist}(A)$ with $a \to \mu$ is called a normal form w.r.t. $\to$. The set of all normal forms w.r.t. $\to$ is denoted by $\mathtt{NF}_{\to}$.*

Example 3.3.7. The PARS $\to_{\mathsf{rw}}$ has the normal forms $\mathtt{NF}_{\to_{\mathsf{rw}}} = \{x \in \mathbb{Z} \mid x \leq 0\}$, and the normal forms of $\to_{\mathsf{coin}}$ are $\mathtt{NF}_{\to_{\mathsf{coin}}} = \{\mathsf{head}\}$.

If we want to lift the notion of a reduction sequence from the non-probabilistic to the probabilistic setting, then we encounter a crucial problem. While the relation of an ARS relates objects to other objects, the relation of a PARS relates objects not to objects, but to multi-distributions. To solve this problem, one can follow three different approaches. The first one is to *lift* the relation $\to$ to a relation $\rightrightarrows \subseteq \mathrm{FDist}(A) \times \mathrm{FDist}(A)$ between multi-distributions [ADLY20]. In this way, one obtains a relation $\rightrightarrows$ on $\mathrm{FDist}(A)$. Second, one can use trees and use the relation $\to$ to define the edge relation in such a tree [KG25]. Then, instead of reduction sequences one considers reduction trees. Finally, one can follow a more mathematical approach and define a stochastic process for $\to$, completely omitting the necessity of defining a notion of reduction and paths [BK02, BG05]. We introduce all three options as all of them have their own advantages and disadvantages.

We start by lifting the relation $\to \subseteq A \times \mathrm{FDist}(A)$ to a relation $\rightrightarrows \subseteq \mathrm{FDist}(A) \times \mathrm{FDist}(A)$ between multi-distributions in order to track all probabilistic reduction sequences (up to non-determinism) at once. This idea was already used in, e.g., [DCM18, ADLY20, Fag22, KG23]. For any $0 < p \leq 1$ and any $\mu \in \mathrm{FDist}(A)$, let $p \cdot \mu = \{p \cdot q\, a \mid {}^{q}a \in \mu\}$.

Definition 3.3.8 – Lifting.
*The lifting $\rightrightarrows \subseteq \mathrm{FDist}(A) \times \mathrm{FDist}(A)$ of a relation $\to \subseteq A \times \mathrm{FDist}(A)$ is the smallest relation such that*

- *if $a \in \mathtt{NF}_{\to}$, then $\{{}^{1}a\} \rightrightarrows \{{}^{1}a\}$,*
- *if $a \to \mu$, then $\{{}^{1}a\} \rightrightarrows \mu$, and*
- *if for all $1 \leq j \leq k$ there are $\mu_j, \nu_j \in \mathrm{FDist}(A)$ with $\mu_j \rightrightarrows \nu_j$ and $0 < p_j \leq 1$ with $\sum_{1 \leq j \leq k} p_j = 1$, then $\biguplus_{1 \leq j \leq k} p_j \cdot \mu_j \rightrightarrows \biguplus_{1 \leq j \leq k} p_j \cdot \nu_j$.*

*By $\rightrightarrows_{\alpha}$ we denote the lifting of an indexed relation $\to_{\alpha}$.*

The main advantage of using the lifting is that we obtain a non-probabilistic relation $\rightrightarrows$ on $\mathrm{FDist}(A)$, which can be used to lift all definitions from ARSs to PARSs by simply considering the lift $\rightrightarrows$ of $\to$ (formally, $(\mathrm{FDist}(A), \rightrightarrows)$ is not an ARS, as $\mathrm{FDist}(A)$ is uncountable, but none of these definitions relies on the countability of the universe). We say that a reduction $\{{}^{1}a\} \rightrightarrows \cdots$ *starts* with $a$.

Example 3.3.9 (Lifting). Consider the PARS $(\{\mathsf{head}, \mathsf{tail}\}, \to_{\mathsf{coin}})$ from Example 3.3.5. The only possible reduction $\mu_0 \rightrightarrows_{\mathsf{coin}} \mu_1 \rightrightarrows_{\mathsf{coin}} \mu_2 \rightrightarrows_{\mathsf{coin}} \cdots$ starting with $\mathsf{tail}$ is

$$\begin{array}{lll} & \{{}^{1}\underline{\mathsf{tail}}\} & = \mu_0 \\ \rightrightarrows_{\mathsf{coin}} & \{{}^{1/2}\mathsf{head}, {}^{1/2}\underline{\mathsf{tail}}\} & = \mu_1 \\ \rightrightarrows_{\mathsf{coin}} & \{{}^{1/2}\mathsf{head}, {}^{1/4}\mathsf{head}, {}^{1/4}\mathsf{tail}\} & = \mu_2 \\ \rightrightarrows_{\mathsf{coin}} & \cdots & \end{array}$$

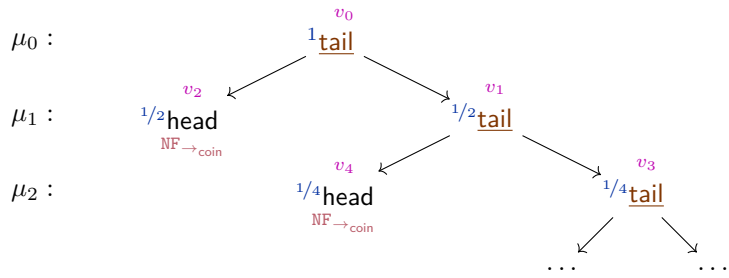



Figure 3.4: Corresponding $\to_{\mathsf{coin}}$-RT $\mathfrak{T}$ for the $\rightrightarrows_{\mathsf{coin}}$-reduction in Example 3.3.9.

Example 3.3.9 shows why we need to consider multi-distributions instead of distributions. Both of the heads in $\mu_2$ have been generated by different computations. Afterwards, if we had multiple rules to rewrite head, we could use different rules to rewrite both heads into different objects again. If we "merged" the entries within the distribution and only considered the distribution $\mu_2' = \{^{1/2+1/4}\mathsf{head}, ^{1/4}\mathsf{tail}\} = \{^{3/4}\mathsf{head}, ^{1/4}\mathsf{tail}\}$, then it would not be possible to rewrite them differently anymore.

The second way to track all possible reductions (up to non-determinism) with their corresponding probabilities is to lift $\to$ to *reduction trees (RTs)* as we did in [KDG24, KG25, KG26]. Compared to the lifting relation $\rightrightarrows$, reduction trees lead to a more fine-grained definition of probabilistic reductions. Single rewrite steps on an object level are modeled by a single node together with its successors.

Definition 3.3.10 – Reduction Tree.
*Let $(A, \to)$ be a PARS. A labeled tree $\mathfrak{T} = (N, E)$ is a $\to$-reduction tree ($\to$-RT) or simply reduction tree (RT) if the nodes $v \in N$ are labeled by pairs $^{p_v}a_v$ of a non-zero probability $0 < p_v \leq 1$ and an object $a_v \in A$, where the root $\mathfrak{r}$ is always labeled with the probability $p_{\mathfrak{r}} = 1$. For each non-leaf node $v \in N$ with the successors $w_1, \ldots, w_k \in N$, the edge relation represents a reduction step with the relation $\to$, i.e., $a_v \to \{^{p_{w_1}/p_v}a_{w_1}, \ldots, ^{p_{w_k}/p_v}a_{w_k}\}$.*

a) *By* Leaf$(\mathfrak{T})$ *we denote the set of leaves.*
b) *By* root$(\mathfrak{T})$ *we denote the object $a \in A$ at the root, i.e.,* root$(\mathfrak{T}) = a_{\mathfrak{r}}$. *We say that $\mathfrak{T}$ starts with $a$ if* root$(\mathfrak{T}) = a$.
c) *We say that $\mathfrak{T}$ is fully evaluated if for every $v \in$* Leaf$(\mathfrak{T})$ *the corresponding object $a_v$ is a normal form, i.e., $a_v \in \mathtt{NF}_{\to}$.*

*Whenever the referenced reduction tree is ambiguous, we will explicitly indicate the corresponding tree. For example, for the probability $p_v$ of the node $v \in N$ of some $\to$-RT $\mathfrak{T} = (N, E)$, we may also write $p_v^{\mathfrak{T}}$, and $N^{\mathfrak{T}}$ for its set of nodes.*

Example 3.3.11 (Reduction Tree). In Figure 3.4 one can see the $\to_{\mathsf{coin}}$-RT $\mathfrak{T}$ for the $\rightrightarrows_{\mathsf{coin}}$-reduction from Example 3.3.9. Note that the normal forms remain in each multi-distribution of a $\rightrightarrows_{\mathsf{coin}}$-reduction, but they are leaves of the corresponding $\to_{\mathsf{coin}}$-RT. Here, we have $N^{\mathfrak{T}} = \{v_0, v_1, v_2, v_3, \ldots\}$, where $v_0 = \mathfrak{r}^{\mathfrak{T}}$ is the root with root$(\mathfrak{T}) = \mathsf{tail}$, and we have Leaf$(\mathfrak{T}) = \{v_2, v_4, \ldots\}$. Additionally, the node $v_2$ has depth 1.

We can again see the reason for multi-distributions instead of distributions. While different nodes can be labeled with the same object and the same probability, they

originated from different program runs and should be treated separately.

For a connected set of nodes $W$ in a reduction tree $\mathfrak{T}$, the *induced subtree* $\mathfrak{T}[W]$ is the subtree that consists of the nodes $W$ with the inherited edges and labels, where the probabilities are rescaled such that the root of $\mathfrak{T}[W]$ has probability 1.

Definition 3.3.12 – Subtree.
*Let $(A, \to)$ be a PARS and let $\mathfrak{T} = (N, E)$ be a $\to$-RT. Let $W \subseteq N$ be non-empty, weakly connected (i.e., connected when ignoring the direction of the edges), and for all $v \in W$ we have $vE \cap W = \varnothing$ or $vE \cap W = vE$. Then, we define the subtree $\mathfrak{T}[W]$ by $\mathfrak{T}[W] = (W, E \cap (W \times W))$. Let $w \in W$ be the root of $\mathfrak{T}[W]$. To ensure that the root of our subtree has the probability 1 again, we use the labeling $L(v) = ({}^{p_v^{\mathfrak{T}}/p_w^{\mathfrak{T}}}a_v^{\mathfrak{T}})$ for all nodes $v \in W$. If $W$ contains the root of $(N, E)$, then we call the subtree grounded.*

The property of being non-empty and weakly connected ensures that the resulting graph $\mathfrak{T}[W]$ is a tree again. The property that we have $vE \cap W = \varnothing$ or $vE \cap W = vE$ ensures that the sum of probabilities for the successors of a node $v$ is equal to the probability for the node $v$ itself.

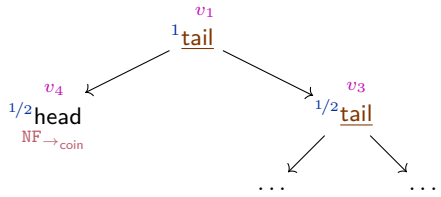



Figure 3.5: Induced subtree of the $\to_{\mathsf{coin}}$-RT $\mathfrak{T}$ from Figure 3.4 with node set $W = N^{\mathfrak{T}} \setminus \{v_0, v_2\}$.




Figure 3.6: Another induced subtree of the $\to_{\mathsf{coin}}$-RT $\mathfrak{T}$ from Figure 3.4 with node set $W = \{v_2\}$.

Example 3.3.13. In Figures 3.5 and 3.6 we have two different subtrees of $\mathfrak{T}$ from Figure 3.4.

Finally, the original work on probabilistic rewriting [BK02, BG05] considered probabilistic rewriting as a stochastic process. Although all the formality can become tedious, it gives nice insights into the mathematical structure hidden underneath the structural behavior, which is just rewriting on pairs of numbers and objects.

First we get rid of the occurring non-determinism by fixing a *scheduler* that resolves all occurring non-determinisms from rule selection. These schedulers are analogous to the ones for MDPs in Definition 2.4.24.

Definition 3.3.14 – Scheduler for PARS.
*Let $(A, \to)$ be a PARS and $n \in \mathbb{N}$. An $n$-step history is a sequence $a_{0..n} = (a_0, \ldots, a_n) \in A^+$. We call $a_{0..n}$ terminating if $a_n \in \mathtt{NF}_\to$.*
*A scheduler is a mapping $\mathfrak{S} : A^+ \to \mathrm{FDist}(A)$ that satisfies $a_n \to \mathfrak{S}(a_{0..n})$ whenever $a_n \notin \mathtt{NF}_\to$, and $\mathfrak{S}(a_{0..n}) = \{{}^1a_n\}$ whenever $a_n \in \mathtt{NF}_\to$. We call $a_{0..n}$ realizable under $\mathfrak{S}$ if $\mathfrak{S}(a_{0..i})(a_{i+1}) > 0$ for every $0 \leq i < n$.*

Given a specific start object, a scheduler completely resolves any non-determinism and

defines for each history the rule to use, and hence, the distribution over all possible reducts. Realizability of histories ensures that the chosen reduct can be reached with a certain probability after a single reduction step. With schedulers, we can define probabilistic rewriting as a stochastic process. Note that the initial distribution is given by $X_0^{\mathfrak{S}}$, and as usual, we start with a single object $a \in A$.

Definition 3.3.15 – Stochastic Reduction.
*Let $(A, \to)$ be a PARS. A stochastic $\to$-reduction under a given scheduler $\mathfrak{S}$ is a stochastic process $\vec{X}^{\mathfrak{S}} = (X_n^{\mathfrak{S}})_{n\in\mathbb{N}}$ over $A$ such that $X_0^{\mathfrak{S}} = \mathbb{1}_a$ for some $a \in A$ and*

$$\begin{aligned} \mathbb{P}(X_{n+1}^{\mathfrak{S}} = b \mid X_n^{\mathfrak{S}} = b) &= 1, && \text{if } b \in \mathtt{NF}_{\to}, \text{ and} \\ \mathbb{P}(X_{n+1}^{\mathfrak{S}} = b \mid X_n^{\mathfrak{S}} = a_n, \dots, X_0^{\mathfrak{S}} = a_0) &= \mathfrak{S}(a_{0..n})(b), && \text{if } a_{0..n} \text{ is realizable under } \mathfrak{S} \\ & && \text{and not terminating.} \end{aligned}$$

Remark 3.3.16 – Entries of Multi-Distributions in Histories.
A multi-distribution may contain the same object several times, e.g., $a \to \{{}^{1/2}b, {}^{1/2}b\}$. To treat such copies separately, as the lifting does, we implicitly assume a fixed order on the entries of every multi-distribution $\mu = \{{}^{p_1}b_1, \dots, {}^{p_k}b_k\}$ with $a \to \mu$, and a history $a_{0..n}$ records in each step which entry was taken, not just its object. Accordingly, $\mathfrak{S}(a_{0..n})(a_{n+1})$ denotes the probability $p_j$ of the chosen entry ${}^{p_j}b_j$ with $b_j = a_{n+1}$, and an event $X_0 = a_0, \dots, X_n = a_n$ refers to such a history. Hence, two copies of the same object yield different histories, and a scheduler may treat them differently. Such a stochastic reduction exists by Theorem 2.3.21, applied to the countable set of histories, where every history $a_{0..n}$ moves to $a_{0..n+1}$ with probability $\mathfrak{S}(a_{0..n})(a_{n+1})$, and $X_n$ is the last object of the $n$-step history.

If we want to compute the probability of having an element $a_n$ in the $n$-th step, we can compute $\mathbb{P}(X_n = a_n) = \sum_{a_{0..n-1}\in A^n} \mathbb{P}(X_0 = a_0, \dots, X_{n-1} = a_{n-1}, X_n = a_n)$ by the law of total probability.

Example 3.3.17 (Stochastic Reduction). The PARS $(\{\mathsf{head}, \mathsf{tail}\}, \to_{\mathsf{coin}})$ from Example 3.3.5 is completely deterministic, i.e., there exists a unique scheduler $\mathfrak{S}_{\mathsf{coin}}$, which selects the only possible rule for $\mathsf{tail}$. Let $\vec{X}_{\mathsf{coin}} = (X_n)_{n\in\mathbb{N}}$ be a stochastic $\to_{\mathsf{coin}}$-reduction under $\mathfrak{S}_{\mathsf{coin}}$ with $X_0 = \mathbb{1}_{\mathsf{tail}}$. The probability of having $\mathsf{head}$ in the second stochastic reduction step is $\mathbb{P}(X_2 = \mathsf{head}) = {}^3/_4$, because

$$\begin{aligned} & \mathbb{P}(X_2 = \mathsf{head}) \\ = \; & \mathbb{P}(X_2 = \mathsf{head}, X_1 = \mathsf{head}, X_0 = \mathsf{tail}) + \mathbb{P}(X_2 = \mathsf{head}, X_1 = \mathsf{tail}, X_0 = \mathsf{tail}) \\ = \; & \mathbb{P}(X_2 = \mathsf{head} \mid X_1 = \mathsf{head}, X_0 = \mathsf{tail}) \cdot \mathbb{P}(X_1 = \mathsf{head} \mid X_0 = \mathsf{tail}) \cdot \mathbb{P}(X_0 = \mathsf{tail}) \\ & + \mathbb{P}(X_2 = \mathsf{head} \mid X_1 = \mathsf{tail}, X_0 = \mathsf{tail}) \cdot \mathbb{P}(X_1 = \mathsf{tail} \mid X_0 = \mathsf{tail}) \cdot \mathbb{P}(X_0 = \mathsf{tail}) \\ = \; & 1 \cdot {}^1/_2 \cdot 1 + {}^1/_2 \cdot {}^1/_2 \cdot 1 \; = \; {}^3/_4. \end{aligned}$$

### 3.3.1 Notions of Probabilistic Termination

While the notion of "termination" was straightforward in the non-probabilistic setting, there are several notions of "termination" in the probabilistic setting as we have already seen in Section 2.3 and Section 2.4. All of them can be defined via the rewrite sequences of the lifting, reduction trees, or stochastic reductions. We now introduce and illustrate all the different notions of "termination", namely *certain termination*, *almost-sure*

*termination*, *positive almost-sure termination*, and *strong almost-sure termination*, using all three notions of probabilistic reductions and show their equivalence.

> Remark 3.3.18 – Distinguishing the Different Termination Definitions.
> In the following, we clearly distinguish the different notions of termination depending on the notion of probabilistic reduction used for the definition. This allows us to formally prove their equivalence. Starting with Section 3.4, we use the different notions of termination without specifically referring to any notion of probabilistic reduction, since all of them are equivalent.

**Definitions via the Lifting** We start by considering $\rightrightarrows$-reductions. In this case, *certain termination* means that we eventually only have normal forms inside the distributions of every possible $\rightrightarrows$-reduction.

> Definition 3.3.19 – Certain Termination (Lifting).
> *Let $(A, \to)$ be a PARS. The relation $\to$ is certainly terminating if for every infinite $\rightrightarrows$-sequence $\vec{\mu} = (\mu_n)_{n\in\mathbb{N}}$ there exists an $n \in \mathbb{N}$ such that $a \in \mathtt{NF}_\to$ for all $a \in \operatorname{Supp}(\mu_n)$.*

Certain termination is a hard constraint in the probabilistic setting, where one is often interested in infinite reductions. In fact, there may exist runs that are non-terminating, but the probability of the event that an infinite run occurs remains 0. To express the concept of the termination probability, one has to determine the probability for normal forms in a multi-distribution.

> Definition 3.3.20 – Probability of Normal Forms.
> *Let $(A, \to)$ be a PARS and let $\mu \in \operatorname{FDist}(A)$. By $|\mu|_\to = \sum_{p_a \in \mu, a\in\mathtt{NF}_\to} p$ we denote the probability of normal forms within $\mu$.*

Example 3.3.21. Consider $\mu_2 = \{{}^{1/2}\mathsf{head}, {}^{1/4}\mathsf{head}, {}^{1/4}\mathsf{tail}\}$ from Example 3.3.9. Then $|\mu_2|_{\to_{\mathsf{coin}}} = 1/2 + 1/4 = 3/4$.

A PARS where non-terminating reductions have a zero probability is called *almost surely terminating*. Similar to weak termination from Definition 3.1.7, we can also define a notion of *weak* almost-sure termination where it is possible to create a $\rightrightarrows$-sequence that converges with probability 1 from every possible start object $a \in A$, but there may exist $\rightrightarrows$-sequences that converge with probability less than 1.

> Definition 3.3.22 – Almost-Sure Termination (Lifting).
> *Let $(A, \to)$ be a PARS.*
>
> a) *We say that an infinite $\rightrightarrows$-reduction $\vec{\mu} = (\mu_n)_{n\in\mathbb{N}}$ converges (or terminates) with probability $|\vec{\mu}| = \lim_{n\to\infty} |\mu_n|_\to$.*
>
> b) *An object $a \in A$ is almost surely terminating w.r.t. $\to$ if $|\vec{\mu}| = 1$ holds for every infinite $\rightrightarrows$-reduction $\vec{\mu} = (\mu_n)_{n\in\mathbb{N}}$ with $\mu_0 = \{{}^1a\}$. The PARS $\to$ is almost surely terminating (denoted $\mathtt{AST}(\to)$) if all $a \in A$ are almost surely terminating w.r.t. $\to$.*
>
> c) *An object $a \in A$ is weakly $\mathtt{AST}$ w.r.t. $\to$ if there exists an infinite $\rightrightarrows$-reduction $\vec{\mu} = (\mu_n)_{n\in\mathbb{N}}$ with $\mu_0 = \{{}^1a\}$ and $|\vec{\mu}| = 1$. The PARS $\to$ is weakly $\mathtt{AST}$*

*(denoted* $\mathtt{wAST}(\to)$*) if all* $a \in A$ *are weakly* $\mathtt{AST}$ *w.r.t.* $\to$*.*

For the definition of $\mathtt{wAST}$ recall that by Definition 3.3.8 every object can start infinite $\rightrightarrows$-sequences, as we keep normal forms in $\rightrightarrows$-steps, similar to the looping terminal states in a Markov chain. Note that $\mathtt{AST}(\to) \implies \mathtt{wAST}(\to)$ for every PARS $(A, \to)$.

Example 3.3.23. For the (unique) infinite $\rightrightarrows_{\mathsf{coin}}$-reduction $\vec{\mu}$ from Example 3.3.9, we have $|\vec{\mu}| = 1$. Indeed, we have $\mathtt{AST}(\to_{\mathsf{coin}})$, and thus, $\mathtt{wAST}(\to_{\mathsf{coin}})$ as well.

Analogously, every infinite $\rightrightarrows_{\mathsf{rw}}$-reduction $\vec{\mu}$ that starts with a single object converges with probability 1, as $\to_{\mathsf{rw}}$ models a symmetric random walk, see Section 2.3. Therefore, we have both $\mathtt{AST}(\to_{\mathsf{rw}})$ and $\mathtt{wAST}(\to_{\mathsf{rw}})$.

Next, we define *positive* almost-sure termination, which considers the *expected derivation length* $\mathrm{edl}(\vec{\mu})$ of a reduction $\vec{\mu}$, i.e., the expected number of steps until one reaches a normal form. For positive almost-sure termination, we require that the expected derivation length of every possible reduction starting with some $a \in A$ is finite. In the following definition, $1 - |\mu_n|_{\to}$ is the probability of objects that are *not* in normal form w.r.t. $\to$ after the $n$-th step. Taking the sum of all $1 - |\mu_n|_{\to}$ for all $n \in \mathbb{N}$ yields the expected number of steps we have to perform before terminating.

Definition 3.3.24 – Positive Almost-Sure Termination (Lifting).

*Let* $(A, \to)$ *be a PARS.*

a) *By* $\mathrm{edl}(\vec{\mu}) = \sum_{n=0}^{\infty}(1 - |\mu_n|_{\to})$ *we denote the expected derivation length of an infinite* $\rightrightarrows$*-reduction* $\vec{\mu} = (\mu_n)_{n\in\mathbb{N}}$*.*

b) *An object* $a \in A$ *is positively almost surely terminating w.r.t.* $\to$ *if* $\mathrm{edl}(\vec{\mu})$ *is finite for every infinite* $\rightrightarrows$*-reduction* $\vec{\mu} = (\mu_n)_{n\in\mathbb{N}}$ *with* $\mu_0 = \{^1a\}$*. The PARS* $\to$ *is positively almost surely terminating (denoted* $\mathtt{PAST}(\to)$*) if all* $a \in A$ *are positively almost surely terminating w.r.t.* $\to$*.*

c) *An object* $a \in A$ *is weakly* $\mathtt{PAST}$ *w.r.t.* $\to$ *if there exists an infinite* $\rightrightarrows$*-reduction* $\vec{\mu} = (\mu_n)_{n\in\mathbb{N}}$ *with* $\mu_0 = \{^1a\}$ *such that* $\mathrm{edl}(\vec{\mu})$ *is finite. The PARS* $\to$ *is weakly* $\mathtt{PAST}$ *(denoted* $\mathtt{wPAST}(\to)$*) if all* $a \in A$ *are weakly* $\mathtt{PAST}$ *w.r.t.* $\to$*.*

Remark 3.3.25 – Alternative Representation of the Expected Derivation Length.

For every $\rightrightarrows$-reduction $\vec{\mu}$ that converges with probability 1, we have $\mathrm{edl}(\vec{\mu}) = \sum_{n=0}^{\infty}(1 - |\mu_n|_{\to}) = \sum_{n=1}^{\infty} n \cdot (|\mu_n|_{\to} - |\mu_{n-1}|_{\to})$, where $|\mu_n|_{\to} - |\mu_{n-1}|_{\to}$ denotes the probability that we reach a normal form in the $n$-th step. This is due to the correspondence between a probability mass function $f : \mathbb{N} \to [0, 1]$ and its cumulative distribution function $F : \mathbb{N} \to [0, 1]$ w.r.t. the expected value, namely $\mathbb{E}(f) = \sum_{n=1}^{\infty} n \cdot f(n) = \sum_{n=0}^{\infty}(1 - F(n))$. With $f_{\vec{\mu}}(0) = |\mu_0|_{\to}$ and $f_{\vec{\mu}}(n) = |\mu_n|_{\to} - |\mu_{n-1}|_{\to}$ for all $n > 0$, we get $F_{\vec{\mu}}(n) = |\mu_n|_{\to}$, and hence we obtain the above equation for $\mathrm{edl}(\vec{\mu})$. Note that $f_{\vec{\mu}}$ is only a probability mass function if $\lim_{n\to\infty} |\mu_n|_{\to} = 1$, because then $\sum_{n=0}^{\infty} f_{\vec{\mu}}(n) = |\mu_0|_{\to} + \sum_{n=1}^{\infty}(|\mu_n|_{\to} - |\mu_{n-1}|_{\to}) = \lim_{n\to\infty} |\mu_n|_{\to} = 1$.

It is well known that $\mathtt{PAST}(\to)$ implies $\mathtt{AST}(\to)$, but not vice versa, see [ADLY20].

Example 3.3.26. For every infinite $\rightrightarrows_{\mathsf{rw}}$-reduction $\vec{\mu}$ that starts with an object $n \geq 1$, the expected derivation length $\mathrm{edl}(\vec{\mu})$ is infinite, because $\to_{\mathsf{rw}}$ models a symmetric

random walk, which is not PAST by Corollary 2.3.33. Therefore, $\neg\mathtt{wPAST}(\to_{\mathsf{rw}})$ and $\neg\mathtt{PAST}(\to_{\mathsf{rw}})$.

On the other hand, for the (unique) infinite $\rightrightarrows_{\mathsf{coin}}$-reduction $\vec{\mu}$ from Example 3.3.9, the expected derivation length $\mathrm{edl}(\vec{\mu})$ is $1 + 1/2 + 1/4 + \ldots = 2$, hence $\mathtt{wPAST}(\to_{\mathsf{coin}})$ and $\mathtt{PAST}(\to_{\mathsf{coin}})$.

Next, we define *strong almost-sure termination* (SAST) [FC19, ADLY20], which is even stricter than PAST in case of non-determinism, and was defined in Section 2.4 neither for MCs nor for MDPs. Because of the results regarding memoryless schedulers that completely resolve all non-determinism, there would be no difference between PAST and SAST for MCs and MDPs. SAST requires a finite bound on the expected derivation lengths of all rewrite sequences with the same start object. For an object $a \in A$, the *expected derivation height* $\mathrm{edh}_{\to}(a)$ considers all $\rightrightarrows$-sequences that start with $\{^1a\}$.

Definition 3.3.27 – Strong Almost-Sure Termination (Lifting).

*Let $(A, \to)$ be a PARS.*

a) *We define the expected derivation height of an object $a \in A$ by $\mathrm{edh}_{\to}(a) = \sup\{\mathrm{edl}(\vec{\mu}) \mid \vec{\mu} = (\mu_n)_{n\in\mathbb{N}}$ is an infinite $\rightrightarrows$-reduction with $\mu_0 = \{^1a\}\}$.*

b) *An object $a \in A$ is strongly almost surely terminating w.r.t. $\to$ if $\mathrm{edh}_{\to}(a)$ is finite. The PARS $\to$ is strongly almost surely terminating (denoted $\mathtt{SAST}(\to)$) if all $a \in A$ are strongly almost surely terminating w.r.t. $\to$.*

Note that in contrast to wAST and wPAST, we did not define any notion of *weak* SAST. The reason is that the definition of weak forms of termination always only requires the *existence* of some suitable reduction. But the definition of strong almost-sure termination imposes a requirement on *all* sequences. Thus, it is not clear how to obtain a useful definition for "weak SAST" that differs from wPAST.

Example 3.3.28. The PARS $\to_{\mathsf{rw}}$ cannot be SAST, as the start object 1 has an infinite expected derivation height, i.e., $\mathrm{edh}_{\to_{\mathsf{rw}}}(1) = \infty$, see Example 3.3.26.

The PARS $\to_{\mathsf{coin}}$ is SAST, because the expected derivation heights $\mathrm{edh}_{\to_{\mathsf{coin}}}(\mathsf{head}) = 0$ and $\mathrm{edh}_{\to_{\mathsf{coin}}}(\mathsf{tail}) = 2$ of the two possible start objects are both finite. Therefore, $\mathtt{SAST}(\to_{\mathsf{coin}})$ holds.

Counterexample 3.3.29. The difference between PAST and SAST is purely because of possible non-determinism. An example of a non-probabilistic PARS $(\mathbb{N} \cup \{\mathsf{start}\}, \to)$ that is PAST but not SAST is $\to = \{\mathsf{start} \to \{^1n\} \mid n \in \mathbb{N}\} \cup \{n+1 \to \{^1n\} \mid n \in \mathbb{N}\}$. The (expected) derivation length of every possible reduction is finite, and thus, we have $\mathtt{PAST}(\to)$. However, due to the infinitely many choices for reducing $\mathsf{start}$ that all lead to reductions of different (expected) derivation lengths, we have $\mathrm{edh}_{\to}(\mathsf{start}) = \infty$. Thus, $\neg\mathtt{SAST}(\to)$.

Again, one can show that $\mathtt{SAST}(\to)$ implies $\mathtt{PAST}(\to)$, but not vice versa [ADLY20].

Theorem 3.3.30 – Relation between AST, PAST, and SAST, [ADLY20].

Let $(A, \to)$ be a PARS. Then, $\mathtt{SAST}(\to) \implies \mathtt{PAST}(\to) \implies \mathtt{AST}(\to)$, but the other directions do not hold in general.

*Proof.* A proof of this theorem can be found in [ADLY20]. *Proof QED*

$\mathtt{PAST}(\to)$ and $\mathtt{SAST}(\to)$ are already defined in terms of the expected derivation length and height, but they consider arbitrary start objects of arbitrary size. As in the non-probabilistic setting, one can also regard a function that maps the size of the start object to the expected derivation length, leading to the *expected derivational complexity*.

Definition 3.3.31 – Expected Derivational Complexity, $\mathrm{edc}_\to$.

*Let $(A, \to)$ be a PARS and $|\,.\,| : A \to \mathbb{N}$ be a size function. The expected derivational complexity $\mathrm{edc}_\to : \mathbb{N} \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ is defined as*

$$\mathrm{edc}_\to(n) = \sup\{\mathrm{edh}_\to(a) \mid a \in A, |a| \leq n\}.$$

Again, we use the function $\iota$ from Definition 3.1.12 to relate the expected derivational complexity to asymptotic complexities. Note that as in the non-probabilistic setting, this definition uses the expected derivation height of an object, and not the expected derivation length of a reduction. Hence, the expected derivational complexity $\mathrm{edc}_\to$ relates to $\mathtt{SAST}(\to)$ more than to $\mathtt{PAST}(\to)$. Indeed, we may have both $\mathtt{PAST}(\to)$ and $\iota(\mathrm{edc}_\to) = \mathbf{Inf}$, see Counterexample 3.4.18. Furthermore, we have the following easy observation regarding $\mathrm{edc}_\to$ and $\mathtt{SAST}(\to)$.

Lemma 3.3.32 – Relation between $\mathrm{edc}_\to$ and $\mathtt{SAST}(\to)$.

*Let $(A, \to)$ be a PARS and let $|\,.\,| : A \to \mathbb{N}$ be a size function such that the set $\{a \in A \mid |a| \leq n\}$ is finite for every $n \in \mathbb{N}$. Then $\iota(\mathrm{edc}_\to) \sqsubseteq \mathbf{Fin}$ iff $\mathtt{SAST}(\to)$.*

*Proof.* If we do not have $\mathtt{SAST}(\to)$, then $\iota(\mathrm{edc}_\to) = \mathbf{Inf}$ follows directly from the definition. On the other hand, if $\{a \in A \mid |a| \leq n\}$ is finite, then $\mathrm{edc}_\to(n) = \sup\{\mathrm{edh}_\to(a) \mid a \in A, |a| \leq n\} = \max\{\mathrm{edh}_\to(a) \mid a \in A, |a| \leq n\}$, i.e., $\mathrm{edc}_\to(n)$ is equal to $\mathrm{edh}_\to(a)$ for an object $a \in A$ with $|a| \leq n$, for every $n \in \mathbb{N}$. Hence, if $\iota(\mathrm{edc}_\to) = \mathbf{Inf}$, then there exists some $n \in \mathbb{N}$ and an object $a \in A$ such that $|a| \leq n$ and $\mathrm{edh}_\to(a) = \infty$, and therefore, $\mathtt{SAST}(\to)$ does not hold. *Proof* *QED*

**Definitions via Reduction Trees** Next, we focus on $\to$-RTs, and we reintroduce all necessary notions regarding the termination probability and expected runtimes w.r.t. RTs instead of $\rightrightarrows$-reductions. We start by stating all the definitions in terms of RTs. Note that RTs start by definition only with a single object $a \in A$. In the following, we omit a formal definition of *certain termination* as we are mostly interested in probabilistic notions of termination.

Definition 3.3.33 – Almost-Sure Termination (Reduction Tree).

*Let $(A, \to)$ be a PARS.*

a) *For any $\to$-RT $\mathfrak{T}$ we define $|\mathfrak{T}| = \sum_{v \in \mathrm{Leaf}(\mathfrak{T})} p_v$ and say that the $\to$-RT $\mathfrak{T}$ converges (or terminates) with probability $|\mathfrak{T}|$.*

b) *An object $a \in A$ is $\mathtt{AST}$ w.r.t. $\to$ if $|\mathfrak{T}| = 1$ holds for every $\to$-RT $\mathfrak{T}$ starting with $a$. The PARS $\to$ is $\mathtt{AST}$ if all $a \in A$ are $\mathtt{AST}$ w.r.t. $\to$.*

c) *An object $a \in A$ is weakly $\mathtt{AST}$ w.r.t. $\to$ if there exists a fully evaluated $\to$-RT $\mathfrak{T}$ starting with $a$ with $|\mathfrak{T}| = 1$. The PARS $\to$ is weakly $\mathtt{AST}$ if all $a \in A$ are weakly $\mathtt{AST}$ w.r.t. $\to$.*

**Definition 3.3.34** – Positive Almost-Sure Termination (Reduction Tree).
*Let $(A, \to)$ be a PARS.*

a) *By* $\mathrm{edl}(\mathfrak{T}) = \sum_{x \in N^{\mathfrak{T}} \setminus \mathrm{Leaf}(\mathfrak{T})} p_x = \lim_{n\to\infty} \sum_{x \in N^{\mathfrak{T}} \setminus \mathrm{Leaf}(\mathfrak{T}), \mathrm{d}(x) \leq n} p_x = \sum_{n=0}^{\infty} \sum_{x \in N^{\mathfrak{T}} \setminus \mathrm{Leaf}(\mathfrak{T}), \mathrm{d}(x) = n} p_x$ *we denote the expected derivation length of a $\to$-RT $\mathfrak{T}$.*

b) *An object $a \in A$ is* PAST *w.r.t. $\to$ if* $\mathrm{edl}(\mathfrak{T})$ *is finite for every $\to$-RT $\mathfrak{T}$ starting with $a$. The PARS $\to$ is* PAST *if all $a \in A$ are* PAST *w.r.t. $\to$.*

c) *An object $a \in A$ is weakly* PAST *w.r.t. $\to$ if there exists a fully evaluated $\to$-RT $\mathfrak{T}$ starting with $a$ such that* $\mathrm{edl}(\mathfrak{T})$ *is finite. The PARS $\to$ is weakly* PAST *if all $a \in A$ are weakly* PAST *w.r.t. $\to$.*

**Definition 3.3.35** – Strong Almost-Sure Termination (Reduction Tree).
*Let $(A, \to)$ be a PARS.*

a) *We define the expected derivation height of an object $a \in A$ by* $\mathrm{edh}_{\to}(a) = \sup\{\mathrm{edl}(\mathfrak{T}) \mid \mathfrak{T}$ *is a $\to$-RT starting with $a\}$.*

b) *An object $a \in A$ is* SAST *w.r.t. $\to$ if* $\mathrm{edh}_{\to}(a)$ *is finite. The PARS $\to$ is* SAST *if all $a \in A$ are* SAST *w.r.t. $\to$.*

By Lemmas 3.3.36 to 3.3.38 below, the expected derivation height $\mathrm{edh}_{\to}(a)$ defined via RTs coincides with the one from Definition 3.3.27.

It is easy to observe that every infinite $\rightrightarrows$-reduction $\vec{\mu} = (\mu_n)_{n\in\mathbb{N}}$ that begins with a single start object (i.e., $\mu_0 = \{{}^1a\}$) can be represented by a $\to$-RT $\mathfrak{T}$ that is fully evaluated such that $\lim_{n\to\infty} |\mu_n|_{\to} = |\mathfrak{T}|$ and vice versa. We formalize this observation in the following three lemmas (Lemmas 3.3.36 to 3.3.38).

**Lemma 3.3.36** – From Lifting to Reduction Trees.
*Let $(A, \to)$ be a PARS. For every infinite $\rightrightarrows$-reduction $\vec{\mu} = (\mu_n)_{n\in\mathbb{N}}$ with $\mu_0 = \{{}^1a\}$ for some $a \in A$ there exists a fully evaluated $\to$-RT $\mathfrak{T}$ starting with $a$ such that $|\mathfrak{T}| = |\vec{\mu}|$, and* $\mathrm{edl}(\mathfrak{T}) = \mathrm{edl}(\vec{\mu})$.

*Proof.* We construct the tree $\mathfrak{T}$ recursively such that for every $i \in \mathbb{N}$ and every ${}^pa \in \mu_i$ there is a (unique) corresponding leaf $x_{(p_a)}$ in $\mathfrak{T}$ labeled ${}^pa$ during the $i$-th iteration of the construction. Then, in the $(i+1)$-th iteration, we extend the tree by rewriting the objects that are not in normal form, while the leaves that contain normal forms remain.
We start with a single root labeled by ${}^1a$. Suppose we have already defined the first $i$ layers of the tree $\mathfrak{T}$ for some $i \in \mathbb{N}$, i.e., the tree $\mathfrak{T}_i$ that consists of $\mathfrak{T}$ up to (including) depth $i$. For each ${}^pa \in \mu_i$, we have a (unique) leaf $x_{(p_a)}$ in $\mathfrak{T}_i$ labeled ${}^pa$, by our induction hypothesis. We partition $\mu_i$ into the multisets $\mu_i = Z_{\mathtt{NF}} \uplus Z_{\neg\mathtt{NF}} = \{{}^pa \mid {}^pa \in \mu_i, a \in \mathtt{NF}_{\to}\} \uplus \{{}^pa \mid {}^pa \in \mu_i, a \notin \mathtt{NF}_{\to}\}$.
First, consider the case ${}^pa \in Z_{\mathtt{NF}}$. Since $a$ is in normal form w.r.t. $\to$, we get ${}^pa \in \mu_{i+1}$. For our leaf in $\mathfrak{T}_i$, we do nothing as normal forms remain leaves in $\to$-RTs.
Second, let ${}^pa \in Z_{\neg\mathtt{NF}}$. Since $a \notin \mathtt{NF}_{\to}$, we can find a $\{{}^{p_1}a_1, \ldots, {}^{p_k}a_k\} \in \mathrm{FDist}(A)$ such that $\{{}^{p\cdot p_1}a_1, \ldots, {}^{p\cdot p_k}a_k\} \subseteq \mu_{i+1}$ and $a \to \{{}^{p_1}a_1, \ldots, {}^{p_k}a_k\}$. We extend $\mathfrak{T}$ in the node $x_{(p_a)}$ by performing this rewrite step, i.e., we add $k$ new children for the node $x_{(p_a)}$ labeled ${}^{p\cdot p_j}a_j$ for all $1 \leq j \leq k$.

*Proof*

In the end, we have to show that $|\mathfrak{T}| = |\vec{\mu}|$ and $\mathrm{edl}(\mathfrak{T}) = \mathrm{edl}(\vec{\mu})$. Let $i \in \mathbb{N}$. By construction, for every ${}^{p}a \in \mu_i$ with $a \in \mathtt{NF}_{\to}$ we can find a (unique) leaf in the tree $\mathfrak{T}_i$ that is labeled ${}^{p}a$. Therefore, we have

$$|\mu_i|_{\to} = \sum_{{}^{p}a \in \mu_i, a \in \mathtt{NF}_{\to}} p = \sum_{x \in N^{\mathfrak{T}}, \mathrm{d}(x) \leq i, a_x \in \mathtt{NF}_{\to}} p_x = \sum_{x \in \mathrm{Leaf}(\mathfrak{T}_i), a_x \in \mathtt{NF}_{\to}} p_x, \tag{3.7}$$

and thus

$$|\vec{\mu}| = \lim_{i \to \infty} |\mu_i|_{\to} \overset{(3.7)}{=} \lim_{i \to \infty} \sum_{x \in \mathrm{Leaf}(\mathfrak{T}_i), a_x \in \mathtt{NF}_{\to}} p_x = \sum_{x \in \mathrm{Leaf}(\mathfrak{T})} p_x = |\mathfrak{T}|.$$

Moreover, for every ${}^{p}a \in \mu_i$ with $a \notin \mathtt{NF}_{\to}$ we can find a (unique) leaf at depth $i$ in the tree $\mathfrak{T}_i$ that is labeled ${}^{p}a$, which is an inner node in $\mathfrak{T}$ because $a \notin \mathtt{NF}_{\to}$. Therefore, we have

$$\begin{array}{lllll} & \mathrm{edl}(\vec{\mu}) & = & \sum_{n=0}^{\infty} & (1 - |\mu_n|_{\to}) \\ = & \sum_{n=0}^{\infty} \sum_{{}^{p}a \in \mu_n, a \notin \mathtt{NF}_{\to}} p & = & \sum_{n=0}^{\infty} & \sum_{x \in N^{\mathfrak{T}} \setminus \mathrm{Leaf}(\mathfrak{T}), \mathrm{d}(x) = n} p_x \\ = & \lim_{n \to \infty} \sum_{x \in N^{\mathfrak{T}} \setminus \mathrm{Leaf}(\mathfrak{T}), \mathrm{d}(x) \leq n} p_x & = & \mathrm{edl}(\mathfrak{T}). & \end{array}$$

*QED*

**Lemma 3.3.37** – From Fully Evaluated Reduction Trees to Lifting.

*Let $(A, \to)$ be a PARS. For every fully evaluated $\to$-RT $\mathfrak{T}$ starting with $a \in A$ there exists an infinite $\rightrightarrows$-reduction $\vec{\mu} = (\mu_n)_{n \in \mathbb{N}}$ with $\mu_0 = \{{}^{1}a\}$ and such that $|\mathfrak{T}| = |\vec{\mu}|$, and $\mathrm{edl}(\mathfrak{T}) = \mathrm{edl}(\vec{\mu})$.*

*Proof.* We construct the $\rightrightarrows$-reduction recursively such that for every $i \in \mathbb{N}$ and every node $x \in N^{\mathfrak{T}}$ with $\mathrm{d}(x) = i$, as well as every leaf $x \in \mathrm{Leaf}(\mathfrak{T})$ with $\mathrm{d}(x) < i$, there is a (unique) corresponding ${}^{p}a \in \mu_i$. Then, in the $(i+1)$-th iteration, we extend the $\rightrightarrows$-reduction by rewriting to $\mu_{i+1}$.
We start the $\rightrightarrows$-reduction with $\mu_0 = \{{}^{1}a\}$. Suppose we have already defined $\mu_i$ for some $i \in \mathbb{N}$. We now define the $(i+1)$-th step for the reduction, i.e., we define $\mu_{i+1}$. If $x \in N^{\mathfrak{T}}$ at depth $\mathrm{d}(x) \leq i$ is a leaf, then $a_x$ is in normal form w.r.t. $\to$ and we keep the (unique) corresponding pair ${}^{p_x}a_x \in \mu_i$ in the next multi-distribution of our $\rightrightarrows$-reduction, i.e., ${}^{p_x}a_x \in \mu_{i+1}$. If $x \in N^{\mathfrak{T}}$ is an inner node at depth $\mathrm{d}(x) = i$, then $a_x$ is not in normal form w.r.t. $\to$, as $\mathfrak{T}$ is fully evaluated. Since the edges of the tree, together with the labeling, represent valid $\to$ steps, we have $a_x \to \{{}^{p_{y_1}/p_x}a_{y_1}, \ldots, {}^{p_{y_k}/p_x}a_{y_k}\}$ with $xE = \{y_1, \ldots, y_k\}$. For all $1 \leq j \leq k$, we set ${}^{p_{y_j}}a_{y_j} \in \mu_{i+1}$ as the (unique) corresponding pair for node $y_j$.
In the end, we have to show that $|\mathfrak{T}| = |\vec{\mu}|$ and $\mathrm{edl}(\mathfrak{T}) = \mathrm{edl}(\vec{\mu})$. This can be shown as in the proof of Lemma 3.3.36.

*Proof* *QED*

Furthermore, note that for every $\to$-RT $\mathfrak{T}$, there exists a fully evaluated $\to$-RT $\mathfrak{T}'$ such that $|\mathfrak{T}| \geq |\mathfrak{T}'|$. To get from $\mathfrak{T}$ to $\mathfrak{T}'$ we can simply perform arbitrary rewrite steps at the leaves that are not in normal form to fully evaluate the tree.

**Lemma 3.3.38** – From Reduction Trees to Fully Evaluated Reduction Trees.

*Let $(A, \to)$ be a PARS. For every $\to$-RT $\mathfrak{T}$ starting with $a \in A$ there exists a fully evaluated $\to$-RT $\mathfrak{T}'$ starting with $a$ such that $|\mathfrak{T}| \geq |\mathfrak{T}'|$, and $\mathrm{edl}(\mathfrak{T}) \leq \mathrm{edl}(\mathfrak{T}')$.*

*Proof.* To get from $\mathfrak{T}$ to $\mathfrak{T}'$ we pick every leaf $x \in \mathrm{Leaf}(\mathfrak{T})$ where the corresponding object $a_x$ is not in normal form w.r.t. $\to$ and use arbitrary rewrite steps to completely evaluate it. Obviously, we cannot increase the convergence probability of our tree $\mathfrak{T}$ with this construction nor decrease the expected derivation length. Hence, we have $|\mathfrak{T}'| \leq |\mathfrak{T}|$ and $\mathrm{edl}(\mathfrak{T}) \leq \mathrm{edl}(\mathfrak{T}')$. *Proof* *QED*

We obtain the following corollaries regarding the equivalence of both definitions.

Corollary 3.3.39 – Equivalence of AST: Lifting vs. Reduction Trees.

*Let $(A, \to)$ be a PARS. The notions of almost-sure termination as in Definition 3.3.22 and as in Definition 3.3.33 coincide.*

*Proof.* We show both directions individually. *Proof*

▶ *Proof*($\Longleftarrow$): Assume that $\mathtt{AST}(\to)$ as in Definition 3.3.22 does not hold. Then, there exists an infinite $\rightrightarrows$-reduction $\vec{\mu} = (\mu_n)_{n\in\mathbb{N}}$ with $\mu_0 = \{^1a\}$ and $a \in A$ such that $|\vec{\mu}| < 1$. By Lemma 3.3.36 there exists a $\to$-RT $\mathfrak{T}$ that starts with $a$ such that $|\mathfrak{T}| = |\vec{\mu}| < 1$. Thus, $\mathtt{AST}(\to)$ as in Definition 3.3.33 does not hold either.

▶ *Proof*($\Longrightarrow$): Assume that $\mathtt{AST}(\to)$ as in Definition 3.3.33 does not hold. Then, there exists a $\to$-RT $\mathfrak{T}$ that starts with $a \in A$ such that $|\mathfrak{T}| < 1$. By Lemma 3.3.38, there exists a fully evaluated $\to$-RT $\mathfrak{T}'$ that starts with $a$ such that $|\mathfrak{T}'| \leq |\mathfrak{T}| < 1$, and by Lemma 3.3.37 there exists an infinite $\rightrightarrows$-reduction $\vec{\mu} = (\mu_n)_{n\in\mathbb{N}}$ with $\mu_0 = \{^1a\}$ such that $|\vec{\mu}| = |\mathfrak{T}'| < 1$. Thus, $\mathtt{AST}(\to)$ as in Definition 3.3.22 does not hold either. *QED*

Corollary 3.3.40 – Equivalence of PAST: Lifting vs. Reduction Trees.

*Let $(A, \to)$ be a PARS. The notions of positive almost-sure termination as in Definition 3.3.24 and as in Definition 3.3.34 coincide.*

*Proof.* We show both directions individually. *Proof*

▶ *Proof*($\Longleftarrow$): Assume that $\mathtt{PAST}(\to)$ as in Definition 3.3.24 does not hold. Then, there exists an infinite $\rightrightarrows$-reduction $\vec{\mu} = (\mu_n)_{n\in\mathbb{N}}$ with $\mu_0 = \{^1a\}$ and $a \in A$ such that $\mathrm{edl}(\vec{\mu}) = \infty$. By Lemma 3.3.36 there exists a $\to$-RT $\mathfrak{T}$ starting with $a$ such that $\mathrm{edl}(\mathfrak{T}) = \mathrm{edl}(\vec{\mu}) = \infty$. Thus, $\mathtt{PAST}(\to)$ as in Definition 3.3.34 does not hold either.

▶ *Proof*($\Longrightarrow$): Assume that $\mathtt{PAST}(\to)$ as in Definition 3.3.34 does not hold. Then, there exists a $\to$-RT $\mathfrak{T}$ that starts with $a \in A$ such that $\mathrm{edl}(\mathfrak{T}) = \infty$. By Lemma 3.3.38, there exists a fully evaluated $\to$-RT $\mathfrak{T}'$ starting with $a$ such that $\mathrm{edl}(\mathfrak{T}') \geq \mathrm{edl}(\mathfrak{T}) = \infty$, and by Lemma 3.3.37 there exists an infinite $\rightrightarrows$-reduction $\vec{\mu} = (\mu_n)_{n\in\mathbb{N}}$ with $\mu_0 = \{^1a\}$ such that $\mathrm{edl}(\vec{\mu}) = \mathrm{edl}(\mathfrak{T}') = \infty$. Thus, $\mathtt{PAST}(\to)$ as in Definition 3.3.24 does not hold either. *QED*

Corollary 3.3.41 – Equivalence of SAST: Lifting vs. Reduction Trees.

*Let $(A, \to)$ be a PARS. The notions of strong almost-sure termination as in Definition 3.3.27 and as in Definition 3.3.35 coincide.*

*Proof.* We show both directions individually. *Proof*

▶ *Proof*($\Longleftarrow$): Assume that $\mathtt{SAST}(\to)$ as in Definition 3.3.27 does not hold. Then, there exist an $a \in A$ and a sequence $(\vec{\mu}_m)_{m\in\mathbb{N}}$ of infinite $\rightrightarrows$-reductions $\vec{\mu}_m = (\mu_{m,n})_{n\in\mathbb{N}}$ with $\mu_{m,0} = \{^1a\}$ for all $m \in \mathbb{N}$ such that $\sup_{m\in\mathbb{N}} \mathrm{edl}(\vec{\mu}_m) = \infty$. By applying Lemma 3.3.36 to all individual $\rightrightarrows$-reductions, we obtain a sequence of $\to$-RTs $(\mathfrak{T}_m)_{m\in\mathbb{N}}$ all starting with $a$ such that $\sup_{m\in\mathbb{N}} \mathrm{edl}(\mathfrak{T}_m) = \sup_{m\in\mathbb{N}} \mathrm{edl}(\vec{\mu}_m) = \infty$. Thus, $\mathtt{SAST}(\to)$ as in Definition 3.3.35 does not hold either.

► *Proof* ($\Longrightarrow$): Assume that $\mathtt{SAST}(\to)$ as in Definition 3.3.35 does not hold. Then, there exist an $a \in A$ and a sequence of $\to$-RTs $(\mathfrak{T}_m)_{m\in\mathbb{N}}$ all starting with $a$ such that $\sup_{m\in\mathbb{N}} \operatorname{edl}(\mathfrak{T}_m) = \infty$. By applying Lemma 3.3.38 to all individual $\to$-RTs, there exists a sequence of fully evaluated $\to$-RTs $(\mathfrak{T}'_m)_{m\in\mathbb{N}}$ all starting with $a$ with $\sup_{m\in\mathbb{N}} \operatorname{edl}(\mathfrak{T}'_m) \geq \sup_{m\in\mathbb{N}} \operatorname{edl}(\mathfrak{T}_m) = \infty$, and by applying Lemma 3.3.37 to each fully evaluated $\to$-RT there exists a sequence of infinite $\rightrightarrows$-reductions $\vec{\mu}_m = (\mu_{m,n})_{n\in\mathbb{N}}$ with $\mu_{m,0} = \{^1 a\}$ for all $m \in \mathbb{N}$ such that $\sup_{m\in\mathbb{N}} \operatorname{edl}(\vec{\mu}_m) = \sup_{m\in\mathbb{N}} \operatorname{edl}(\mathfrak{T}'_m) = \infty$. Thus, $\mathtt{SAST}(\to)$ as in Definition 3.3.27 does not hold either. *QED*

Equivalence between the weak notions can be shown analogously.

**Definitions via Stochastic Processes** Finally, we consider stochastic $\to$-reductions. Again, we reintroduce all necessary notions regarding the termination probability and expected runtimes, and we present all the different notions of termination in terms of stochastic $\to$-reductions. In this case, we view probabilistic rewriting as a stochastic process $\vec{X}$ and the termination probability is defined via the *hitting time* $T_{\vec{X}}$ of $\mathtt{NF}_\to$. The following lemmas, corollaries, and proofs are based on [ADLY20, BG05].

Definition 3.3.42 – Almost-Sure Termination (Stochastic Process).

*Let $(A, \to)$ be a PARS. For a stochastic $\to$-reduction $\vec{X} = (X_n)_{n\in\mathbb{N}}$ we define the random variable $T_{\vec{X}} = \inf\{n \in \mathbb{N} \mid X_n \in \mathtt{NF}_\to\}$, where $\inf \varnothing = \infty$. The random variable $T_{\vec{X}}$ is called the hitting time of $\mathtt{NF}_\to$ for $\vec{X}$.*

a) *An object $a \in A$ is $\mathtt{AST}$ w.r.t. $\to$ if $\mathbb{P}(T_{\vec{X}} = \infty) = 0$ holds for every stochastic $\to$-reduction $\vec{X} = (X_n)_{n\in\mathbb{N}}$ with $X_0 = \mathbb{1}_a$. The PARS $\to$ is $\mathtt{AST}$ if all $a \in A$ are $\mathtt{AST}$ w.r.t. $\to$.*

b) *An object $a \in A$ is weakly $\mathtt{AST}$ w.r.t. $\to$ if there exists a stochastic $\to$-reduction $\vec{X} = (X_n)_{n\in\mathbb{N}}$ with $X_0 = \mathbb{1}_a$ and $\mathbb{P}(T_{\vec{X}} = \infty) = 0$. The PARS $\to$ is weakly $\mathtt{AST}$ if all $a \in A$ are weakly $\mathtt{AST}$ w.r.t. $\to$.*

Definition 3.3.43 – Positive Almost-Sure Termination (Stochastic Process).

*Let $(A, \to)$ be a PARS.*

a) *An object $a \in A$ is $\mathtt{PAST}$ w.r.t. $\to$ if $\mathbb{E}(T_{\vec{X}}) < \infty$ holds for every stochastic $\to$-reduction $\vec{X} = (X_n)_{n\in\mathbb{N}}$ with $X_0 = \mathbb{1}_a$. The PARS $\to$ is $\mathtt{PAST}$ if all $a \in A$ are $\mathtt{PAST}$ w.r.t. $\to$.*

b) *An object $a \in A$ is weakly $\mathtt{PAST}$ w.r.t. $\to$ if there exists a stochastic $\to$-reduction $\vec{X} = (X_n)_{n\in\mathbb{N}}$ with $X_0 = \mathbb{1}_a$ and $\mathbb{E}(T_{\vec{X}}) < \infty$. The PARS $\to$ is weakly $\mathtt{PAST}$ if all $a \in A$ are weakly $\mathtt{PAST}$ w.r.t. $\to$.*

Definition 3.3.44 – Strong Almost-Sure Termination (Stochastic Process).

*Let $(A, \to)$ be a PARS.*
*An object $a \in A$ is $\mathtt{SAST}$ w.r.t. $\to$ if there is a bound $C_a \in \mathbb{R}$ such that for every stochastic $\to$-reduction $\vec{X} = (X_n)_{n\in\mathbb{N}}$ with $X_0 = \mathbb{1}_a$ we have $\mathbb{E}(T_{\vec{X}}) < C_a$. The PARS $\to$ is $\mathtt{SAST}$ if all $a \in A$ are $\mathtt{SAST}$ w.r.t. $\to$.*

To prove equivalence of our notions of termination, we relate stochastic $\to$-reductions to $\rightrightarrows$-reductions again. For a stochastic $\to$-reduction $\vec{X} = (X_n)_{n\in\mathbb{N}}$, we define the multi-distributions $X_n^M = \{^p a \mid \mathbb{P}(X_0 = a_0, \ldots, X_n = a) = p > 0\}$. Note that $\mathbb{P}(T_{\vec{X}} < \infty) =$

$\lim_{n\to\infty} \mathbb{P}(T_{\vec{X}} \leq n) = \lim_{n\to\infty} |X_n^M|_{\to}$ and $\mathbb{P}(T_{\vec{X}} = n+1) = |X_{n+1}^M|_{\to} - |X_n^M|_{\to}$ for all $n \in \mathbb{N}$. The following lemmas show that a stochastic $\to$-reduction yields a $\rightrightarrows$-reduction on the multi-distributions $X_0^M \rightrightarrows X_1^M \rightrightarrows \cdots$, and vice versa.

Lemma 3.3.45 – From Lifting to Stochastic Processes.

*Let $(A, \to)$ be a PARS. For every infinite $\rightrightarrows$-reduction $\vec{\mu} = (\mu_n)_{n\in\mathbb{N}}$ with $\mu_0 = \{{}^1a\}$ for some $a \in A$ there exists a scheduler $\mathfrak{S}$ and a stochastic $\to$-reduction $\vec{X} = (X_n)_{n\in\mathbb{N}}$ under $\mathfrak{S}$ with $X_0 = \mathbb{1}_a$ such that $\mathbb{P}(T_{\vec{X}} < \infty) = |\vec{\mu}|$ and $\mathbb{E}(T_{\vec{X}}) = \mathrm{edl}(\vec{\mu})$.*

*Proof.* We define a scheduler $\mathfrak{S}$ such that $\mu_n = X_n^M$ for the stochastic $\to$-reduction $\vec{X} = (X_n)_{n\in\mathbb{N}}$ under $\mathfrak{S}$. To do so, we define functions $\mathfrak{S}_n : A^n \to (A \to \mathbb{R}_{\geq 0})$ inductively for every $n \in \mathbb{N}$ such that

$$\mu_n = \{{}^{\mathfrak{S}_n(a_{0..n-1})(a_n)}a_n \mid \mathfrak{S}_n(a_{0..n-1})(a_n) > 0\}. \tag{3.8}$$

Then the desired scheduler is $\mathfrak{S}(a_{0..n}) = {}^1\!/_{\mathfrak{S}_n(a_{0..n-1})(a_n)} \cdot \mathfrak{S}_{n+1}(a_{0..n})$. Note that $\mathfrak{S}_{n+1}(a_{0..n})(a_{n+1})$ is the probability of the object $a_{n+1}$ in $\mu_{n+1}$ and $\mathfrak{S}_n(a_{0..n-1})(a_n)$ is the probability of its predecessor $a_n$ in $\mu_n$. Therefore, $a_n \to {}^1\!/_{\mathfrak{S}_n(a_{0..n-1})(a_n)} \cdot \mathfrak{S}_{n+1}(a_{0..n})$, and $\mathfrak{S}$ is a valid scheduler. Next, we present the construction.
In the beginning we define $\mathfrak{S}_0(\epsilon)(a) = 1$ for the start object $a$, and $\mathfrak{S}_0(\epsilon)(b) = 0$ for all $b \neq a$. Next, assume that (3.8) holds for some $n \in \mathbb{N}$. Then for $n+1$ we define $I = \{a_{0..n} \mid \mathfrak{S}_n(a_{0..n-1})(a_n) > 0, a_n \notin \mathtt{NF}_{\to}\}$. Since $\mu_n \rightrightarrows \mu_{n+1}$, for each $a_{0..n} \in I$ there are multi-distributions $\nu_{a_{0..n}}$ such that $a_n \to \nu_{a_{0..n}}$. All in all, we get

$$\mu_{n+1} = \{{}^p a \mid {}^p a \in \mu_n, a \in \mathtt{NF}_{\to}\} \uplus \biguplus_{a_{0..n}\in I} \mathfrak{S}_n(a_{0..n-1})(a_n) \cdot \nu_{a_{0..n}}.$$

Hence, defining

$$\mathfrak{S}_{n+1}(a_{0..n}) = \begin{cases} \mathfrak{S}_n(a_{0..n-1})(a_n) \cdot \nu_{a_{0..n}}, & \text{if } a_{0..n} \in I, \\ \mathfrak{S}_n(a_{0..n-1})(a_n) \cdot \{{}^1a_n\}, & \text{if } \mathfrak{S}_n(a_{0..n-1})(a_n) > 0 \text{ and } a_n \in \mathtt{NF}_{\to}, \\ \varnothing, & \text{otherwise,} \end{cases}$$

yields the desired function $\mathfrak{S}_{n+1}$. Here, all elements that are not explicitly mapped to a probability are mapped to 0 instead.
For the stochastic $\to$-reduction $\vec{X}$ we obtain $\mathbb{P}(T_{\vec{X}} < \infty) = |\vec{\mu}|$ and $\mathbb{E}(T_{\vec{X}}) = \mathrm{edl}(\vec{\mu})$. Concerning the first property, we have $\mathbb{P}(T_{\vec{X}} < \infty) = \lim_{n\to\infty} \mathbb{P}(T_{\vec{X}} \leq n) = \lim_{n\to\infty} |X_n^M|_{\to} = \lim_{n\to\infty} |\mu_n|_{\to} = |\vec{\mu}|$. For the second property, we have $\mathbb{E}(T_{\vec{X}}) = \sum_{n=1}^{\infty} n \cdot \mathbb{P}(T_{\vec{X}} = n) = \sum_{n=1}^{\infty} n \cdot (|X_n^M|_{\to} - |X_{n-1}^M|_{\to}) = \sum_{n=1}^{\infty} n \cdot (|\mu_n|_{\to} - |\mu_{n-1}|_{\to}) = \mathrm{edl}(\vec{\mu})$ assuming that $\mathbb{P}(T_{\vec{X}} < \infty) = |\vec{\mu}| = 1$. Otherwise, we would have $\mathbb{E}(T_{\vec{X}}) = \mathrm{edl}(\vec{\mu}) = \infty$. *QED*

Lemma 3.3.46 – From Stochastic Processes to Lifting.

*Let $(A, \to)$ be a PARS. For every stochastic $\to$-reduction $\vec{X} = (X_n)_{n\in\mathbb{N}}$ with $X_0 = \mathbb{1}_a$ for some $a \in A$ there exists an infinite $\rightrightarrows$-reduction $\vec{\mu} = (\mu_n)_{n\in\mathbb{N}}$ with $\mu_0 = \{{}^1a\}$ such that $|\vec{\mu}| = \mathbb{P}(T_{\vec{X}} < \infty)$ and $\mathrm{edl}(\vec{\mu}) = \mathbb{E}(T_{\vec{X}})$.*

*Proof.* We show that $X_0^M \rightrightarrows X_1^M \rightrightarrows \cdots$ is a valid $\rightrightarrows$-reduction sequence. Then $\vec{\mu} = (X_n^M)_{n\in\mathbb{N}}$ is the desired $\rightrightarrows$-reduction with $|\vec{\mu}| = \mathbb{P}(T_{\vec{X}} < \infty)$ and $\mathrm{edl}(\vec{\mu}) = \mathbb{E}(T_{\vec{X}})$.

The latter two can be proven as in the proof of Lemma 3.3.45.
Let $n \in \mathbb{N}$. We have

$$\begin{aligned}
X_n^M &= \{{}^p a_n \mid \mathbb{P}(X_0 = a_0, \dots, X_n = a_n) = p > 0, a_n \notin \mathtt{NF}_\to\} \\
&\quad \uplus \{{}^p a_n \mid \mathbb{P}(X_0 = a_0, \dots, X_n = a_n) = p > 0, a_n \in \mathtt{NF}_\to\} \\
&\rightrightarrows \biguplus \{p \cdot \mathfrak{S}(a_{0..n}) \mid \mathbb{P}(X_0 = a_0, \dots, X_n = a_n) = p > 0, a_n \notin \mathtt{NF}_\to\} \\
&\quad \uplus \{{}^p a_n \mid \mathbb{P}(X_0 = a_0, \dots, X_n = a_n) = p > 0, a_n \in \mathtt{NF}_\to\} \\
&= \{{}^q a_{n+1} \mid \mathbb{P}(X_0 = a_0, \dots, X_n = a_n) \cdot \mathfrak{S}(a_{0..n})(a_{n+1}) = q > 0, a_n \notin \mathtt{NF}_\to\} \\
&\quad \uplus \{{}^p a_n \mid \mathbb{P}(X_0 = a_0, \dots, X_n = a_n) = p > 0, a_n \in \mathtt{NF}_\to\} \\
&= \{{}^q a_{n+1} \mid \mathbb{P}(X_0 = a_0, \dots, X_n = a_n, X_{n+1} = a_{n+1}) = q > 0, a_n \notin \mathtt{NF}_\to\} \\
&\quad \uplus \{{}^p a_n \mid \mathbb{P}(X_0 = a_0, \dots, X_n = a_n, X_{n+1} = a_n) = p > 0, a_n \in \mathtt{NF}_\to\} \\
&= X_{n+1}^M.
\end{aligned}$$

*QED*

**Corollary 3.3.47** – Equivalence of `AST`: Lifting vs. Stochastic Process.

*Let $(A, \to)$ be a PARS. The notions of almost-sure termination as in Definition 3.3.22 and as in Definition 3.3.42 coincide.*

*Proof.* Similar to Corollary 3.3.39 using Lemma 3.3.45 and Lemma 3.3.46. *Proof QED*

**Corollary 3.3.48** – Equivalence of `PAST`: Lifting vs. Stochastic Process.

*Let $(A, \to)$ be a PARS. The notions of positive almost-sure termination as in Definition 3.3.24 and as in Definition 3.3.43 coincide.*

*Proof.* Similar to Corollary 3.3.40 using Lemma 3.3.45 and Lemma 3.3.46. *Proof QED*

**Corollary 3.3.49** – Equivalence of `SAST`: Lifting vs. Stochastic Process.

*Let $(A, \to)$ be a PARS. The notions of strong almost-sure termination as in Definition 3.3.27 and as in Definition 3.3.44 coincide.*

*Proof.* Similar to Corollary 3.3.41 using Lemma 3.3.45 and Lemma 3.3.46. *Proof QED*

Once again, the results hold for weak `AST` and weak `PAST` as well.

SECTION 3.4

## Probabilistic Term Rewriting

Finally, we introduce the main notion of interest: *probabilistic term rewrite systems (PTRSs)*. PTRSs are based on TRSs, just as PARSs are based on ARSs. All important notations for dealing with probabilities in the rewriting setting were introduced in the previous section, see Section 3.3. We follow the same structure for this section as we used to introduce TRSs in Section 3.2. We start in Section 3.4.1 by defining the syntax and semantics of a PTRS. In addition to the expected derivational complexity defined in Section 3.3, we can lift runtime complexity to *expected runtime complexity* as well, and this is done in Section 3.4.2. The different positional evaluation strategies of interest for PTRSs are introduced in Section 3.4.3. In Section 3.4.4 we present the first technique of this thesis to analyze PTRSs. It shows an interesting relation between `PAST` and `SAST` for PTRSs. More precisely, we show that `PAST` and `SAST` are equivalent for most practically relevant PTRSs. Hence, we focus on proving `AST` and `SAST` and finding upper bounds on the expected derivational complexity and expected

runtime complexity in the remainder of this thesis. Finally, in Section 3.4.5, we present a technique to prove SAST from [ADLY20], and moreover, the technique to prove AST from [KG23, KG26], which is based on [MMKK18]. Both are based on the idea to find a single ranking function to orient all the rules of a PTRS. However, ranking functions are replaced by supermartingales, which provide a quantitative measure of expected decrease. Additionally, we explain how to infer upper complexity bounds on the expected derivational complexity and the expected runtime complexity from such termination proofs based on supermartingales. A section similar to Section 3.2.4 looking into witnesses for non-AST, non-PAST, and non-SAST requires a more in-depth investigation of the problem in the probabilistic setting. Therefore, we postpone such results to Chapter 5. These results form the backbone of many proofs regarding results for proving and disproving AST, PAST, or SAST.

### 3.4.1 Syntax and Semantics

The definition of a probabilistic term rewrite system is nearly the same as that of a non-probabilistic term rewrite system. We only have to change the right-hand sides from terms to distributions over terms. To ease the presentation, we restrict PTRSs to use only finitely supported multi-distributions on the right-hand sides of the rewrite rules. While we conjecture that most results in this thesis can be lifted to infinitely supported multi-distributions, this sometimes requires additional properties (e.g., some kind of bound on the expected value of the infinite distribution), and increases the proof complexity dramatically. Since the proofs of many results are already quite complex compared to the proofs of their non-probabilistic counterparts, we decided to omit this additional overhead in proof complexity. Additionally, remember that one only needs fair coin flips to model every computable distribution [DLZ12], see Example 2.4.3.

Definition 3.4.1 – Probabilistic Term Rewrite System.

*A probabilistic term rewrite rule is a pair* $\ell \to \{{}^{p_1}r_1, \dots, {}^{p_k}r_k\} \in \mathcal{T} \times \mathrm{FDist}(\mathcal{T})$ *such that* $\sum_{1 \le i \le k} p_i = 1$, $\mathcal{V}(r_i) \subseteq \mathcal{V}(\ell)$ *for all* $1 \le i \le k$, *and* $\ell \notin \mathcal{V}$. *A probabilistic term rewrite system (PTRS) is a set* $\mathcal{P}$ *of probabilistic term rewrite rules.*
*The PTRS* $\mathcal{P}$ *induces a probabilistic rewrite relation* $\to_{\mathcal{P}} \subseteq \mathcal{T} \times \mathrm{FDist}(\mathcal{T})$ *via*

$$s \to_{\mathcal{P}} \{{}^{p_1}t_1, \dots, {}^{p_k}t_k\}, \quad \text{if } s|_\pi = \ell\sigma \text{ and } t_j = s[r_j\sigma]_\pi \text{ for all } 1 \le j \le k,$$

*for a position* $\pi \in \mathrm{pos}(s)$, *a rule* $\ell \to \{{}^{p_1}r_1, \dots, {}^{p_k}r_k\} \in \mathcal{P}$, *and a substitution* $\sigma$. *A subterm* $s|_\pi$ *that is matched by the left-hand side of a rule is called a redex.*

Example 3.4.2 (PTRS). We consider two PTRSs that model the two PARSs from Example 3.3.5. First, the PTRS $\mathcal{P}_{\mathsf{coin}}$ with the only rule $\mathsf{g}(x) \to \{{}^{1/2}\mathsf{g}(x), {}^{1/2}x\}$ models the coin flip $\to_{\mathsf{coin}}$ if we start with the term $\mathsf{g}(x)$. Since the function symbol $\mathsf{g}$ takes an argument, we can also "stack coin flips", which results in the infinite MC depicted in Figure 3.7. The start term decides at which state we start in the MC, or more specifically, the initial state corresponds to the number of $\mathsf{g}$s in the start term.

Second, the PTRS $\mathcal{P}_{\mathsf{rw}}$ with the only rule $\mathsf{g}(x) \to \{{}^{1/2}x, {}^{1/2}\mathsf{g}(\mathsf{b}(\mathsf{g}(x)))\}$ models the random walk $\to_{\mathsf{rw}}$, see Figure 3.8. The measure $\|.\| : \mathcal{T} \to \mathbb{Z}$ that maps terms to numbers for the random walk is $\|t\| = |t|_{\mathsf{g}}$, i.e., the number of $\mathsf{g}$ symbols in $t$.

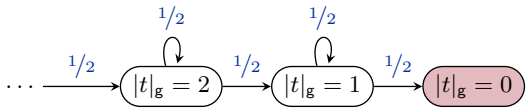
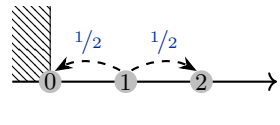



Figure 3.7: $\mathcal{P}_{\mathsf{coin}}$ illustrated as an MC.




Count $\|t\| = |t|_{\mathsf{g}}$

Figure 3.8: $\mathcal{P}_{\mathsf{rw}}$ illustrated as an RWP.

Two examples of possible $\rightrightarrows$-rewrite sequences are

$$\{^{1}\underline{\mathsf{g}}(x)\} \rightrightarrows_{\mathcal{P}_{\mathsf{coin}}} \{^{1/2}\underline{\mathsf{g}}(x), {}^{1/2}x\} \rightrightarrows_{\mathcal{P}_{\mathsf{coin}}} \{^{1/4}\mathsf{g}(x), {}^{1/4}x, {}^{1/2}x\} \rightrightarrows_{\mathcal{P}_{\mathsf{coin}}} \cdots$$

and

$$\{^{1}\underline{\mathsf{g}}(x)\} \rightrightarrows_{\mathcal{P}_{\mathsf{rw}}} \{^{1/2}x, {}^{1/2}\mathsf{g}(\mathsf{b}(\underline{\mathsf{g}}(x)))\} \rightrightarrows_{\mathcal{P}_{\mathsf{rw}}} \{^{1/2}x, {}^{1/4}\mathsf{g}(\mathsf{b}(x)), {}^{1/4}\mathsf{g}(\mathsf{b}(\mathsf{g}(\mathsf{b}(\mathsf{g}(x)))))\} \rightrightarrows_{\mathcal{P}_{\mathsf{rw}}} \cdots$$

Example 3.4.3 (Geometric Distribution). While we prohibit PTRSs with infinitely supported distributions on the right-hand side of their rewrite rules, we are still able to express algorithms that generate an infinitely supported distribution. For example, the PTRS $\mathcal{P}_{\mathsf{geo}}$ with the only rule $\mathsf{geo}(x) \to \{^{1/2}\mathsf{geo}(\mathsf{s}(x)),\ {}^{1/2}x\}$ models the generation of a geometric distribution. When starting with the term $\mathsf{geo}(0)$, it computes the representation $\mathsf{s}^k(0)$ of the number $k \in \mathbb{N}$ with a probability of $(1/2)^{k+1}$. A finite $\to_{\mathcal{P}_{\mathsf{geo}}}$-RT is shown in Figure 3.9, but of course this RT could be extended further to an infinite one.

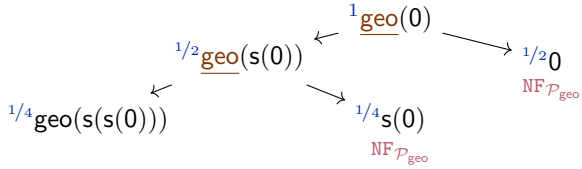



Figure 3.9: $\to_{\mathcal{P}_{\mathsf{geo}}}$-RT starting with $\mathsf{geo}(0)$.

Example 3.4.4 (Probabilistic Quicksort). We continue the example from Example 3.2.32 and integrate a random pivot element selection. The following PTRS $\mathcal{P}_{\mathsf{qs}}$ implements a probabilistic quicksort algorithm.

$$\begin{aligned}
\mathsf{rotate}(\mathsf{nil}) &\to \{^{1}\mathsf{nil}\}\\
\mathsf{rotate}(\mathsf{cons}(x, xs)) &\to \{^{1/2}\mathsf{cons}(x, xs), {}^{1/2}\mathsf{rotate}(\mathsf{app}(xs, \mathsf{cons}(x, \mathsf{nil})))\}\\
\mathsf{qs}(\mathsf{nil}) &\to \{^{1}\mathsf{nil}\}\\
\mathsf{qs}(\mathsf{cons}(x, xs)) &\to \{^{1}\mathsf{qsHelp}(\mathsf{rotate}(\mathsf{cons}(x, xs)))\}\\
\mathsf{qsHelp}(\mathsf{cons}(x, xs)) &\to \{^{1}\mathsf{app}(\mathsf{qs}(\mathsf{low}(x, xs)), \mathsf{cons}(x, \mathsf{qs}(\mathsf{high}(x, xs))))\}\\
\mathsf{low}(x, \mathsf{nil}) &\to \{^{1}\mathsf{nil}\}\\
\mathsf{low}(x, \mathsf{cons}(y, ys)) &\to \{^{1}\mathsf{ifLow}(\mathsf{leq}(x, y), x, \mathsf{cons}(y, ys))\}\\
\mathsf{ifLow}(\mathsf{t}, x, \mathsf{cons}(y, ys)) &\to \{^{1}\mathsf{low}(x, ys)\}\\
\mathsf{ifLow}(\mathsf{f}, x, \mathsf{cons}(y, ys)) &\to \{^{1}\mathsf{cons}(y, \mathsf{low}(x, ys))\}\\
\mathsf{high}(x, \mathsf{nil}) &\to \{^{1}\mathsf{nil}\}
\end{aligned}$$

$$\begin{aligned}
\mathsf{high}(x, \mathsf{cons}(y, ys)) &\to \{^1\mathsf{ifHigh}(\mathsf{leq}(x, y), x, \mathsf{cons}(y, ys))\} \\
\mathsf{ifHigh}(\mathsf{t}, x, \mathsf{cons}(y, ys)) &\to \{^1\mathsf{cons}(y, \mathsf{high}(x, ys))\} \\
\mathsf{ifHigh}(\mathsf{f}, x, \mathsf{cons}(y, ys)) &\to \{^1\mathsf{high}(x, ys)\} \\
\mathsf{leq}(0, x) &\to \{^1\mathsf{t}\} \\
\mathsf{leq}(\mathsf{s}(x), 0) &\to \{^1\mathsf{f}\} \\
\mathsf{leq}(\mathsf{s}(x), \mathsf{s}(y)) &\to \{^1\mathsf{leq}(x, y)\} \\
\mathsf{app}(\mathsf{nil}, ys) &\to \{^1 ys\} \\
\mathsf{app}(\mathsf{cons}(x, xs), ys) &\to \{^1\mathsf{cons}(x, \mathsf{app}(xs, ys))\}
\end{aligned}$$

The first two rotate rules are used to determine a random pivot element. This is achieved by rotating the list and always moving the head element to the end of the list. With a chance of $1/2$ we stop this iteration and use the current head element as the next pivot element. The rest of the rules represent the classical quicksort algorithm without any probabilities.

Almost-sure termination of $\mathcal{P}_{\mathsf{qs}}$ can be proven automatically by AProVE, and we explain in Chapter 7 how this analysis works. More specifically, we show how to analyze probabilistic algorithms like $\mathcal{P}_{\mathsf{qs}}$ that deal with data structures.

Next, we lift all the important properties of TRSs to PTRSs. Note that all properties that only take the left-hand side of the rewrite rules into account can be directly used for PTRSs as well, because the left-hand side remains a single term. Therefore, notions like *overlapping* or *left-linear* can directly be used for PTRSs. For most other properties, we simply have to consider all the terms in the support of the right-hand side. For example, a rule is *right-linear* if all the terms in the support of the right-hand side are linear.

Definition 3.4.5 – Linear, Erasing, Duplicating.

*Let $\mathcal{P}$ be a PTRS.*

a) *A rule $\ell \to \mu \in \mathcal{P}$ is left-linear (*LL*) if $|\ell|_x \leq 1$ for every variable $x \in \mathcal{V}$, right-linear (*RL*) if $|r|_x \leq 1$ for every term $r \in \mathrm{Supp}(\mu)$ and variable $x \in \mathcal{V}$, and linear (*L*) if it is both left-linear and right-linear.*

b) *A rule $\ell \to \mu \in \mathcal{P}$ is duplicating (or variable increasing) if $|\ell|_x < |r|_x$ for some term $r \in \mathrm{Supp}(\mu)$ and variable $x \in \mathcal{V}$. If it is not duplicating, then we call it non-duplicating (*ND*).*

c) *A rule $\ell \to \mu \in \mathcal{P}$ is variable decreasing if $|\ell|_x > |r|_x$ for some term $r \in \mathrm{Supp}(\mu)$ and variable $x \in \mathcal{V}$. If it is not variable decreasing, then we call it NVD.*

d) *A rule $\ell \to \mu \in \mathcal{P}$ is erasing if $|r|_x = 0$ for some term $r \in \mathrm{Supp}(\mu)$ and variable $x \in \mathcal{V}(\ell)$ occurring in the left-hand side. If it is not erasing, then we call it non-erasing (*NE*).*

e) *The PTRS $\mathcal{P}$ is* LL / RL / L / ND / NVD / NE *if all of its rules are.*

Example 3.4.6 (Variable Conditions). The following is a list of examples to illustrate all notions from Definition 3.4.5.

- $\mathsf{f}(x) \to \{^{1/2}\mathsf{g}(x), ^{1/2}\mathsf{h}(x, x)\}$ is left-linear and $\mathsf{h}(x, x) \to \{^{1/2}\mathsf{g}(x), ^{1/2}\mathsf{h}(x, x)\}$ is not.

- $\mathsf{f}(x) \to \{^{1}\mathsf{g}(x)\}$ is right-linear and $\mathsf{f}(x) \to \{^{1/2}\mathsf{g}(x), ^{1/2}\mathsf{h}(x,x)\}$ is not.
- $\mathsf{f}(x) \to \{^{1/2}\mathsf{g}(x), ^{1/2}\mathsf{f}(x)\}$ is linear.
- $\mathsf{h}(x,x) \to \{^{1/2}\mathsf{g}(x), ^{1/2}\mathsf{c}(x,x,x)\}$ is duplicating.
- $\mathsf{h}(x,x) \to \{^{1/2}\mathsf{g}(x), ^{1/2}\mathsf{h}(x,x)\}$ is non-duplicating.
- $\mathsf{h}(x,x) \to \{^{1/2}\mathsf{g}(x), ^{1/2}\mathsf{c}(x,x,x)\}$ is variable decreasing.
- $\mathsf{h}(x,x) \to \{^{1/2}\mathsf{h}(x,x), ^{1/2}\mathsf{c}(x,x,x)\}$ is $\mathtt{NVD}$.
- $\mathsf{f}(x) \to \{^{1/2}\mathsf{a}, ^{1/2}\mathsf{h}(x,x)\}$ is erasing.
- $\mathsf{h}(x,x) \to \{^{1/2}\mathsf{g}(x), ^{1/2}\mathsf{h}(x,x)\}$ is non-erasing.

Note that the rule $\mathsf{h}(x,x) \to \{^{1/2}\mathsf{g}(x), ^{1/2}\mathsf{c}(x,x,x)\}$ is both duplicating and variable decreasing due to the two different terms in the right-hand side.

### 3.4.2 Expected Complexity

Next, we lift runtime complexity to the *expected runtime complexity*, where we again require basic start terms. Since *defined symbols* and *constructor symbols* only consider the left-hand sides of the rules, their definition directly transfers to PTRSs again. Thus, expected runtime complexity is the same as the expected derivational complexity where we additionally require basic start terms.

Definition 3.4.7 – Expected Runtime Complexity, $\mathrm{erc}_{\to}$.
*Let $(\mathcal{T}, \to)$ be a PARS on terms with a set of basic terms $\mathcal{T}_{\mathcal{B}} \subseteq \mathcal{T}$. The expected runtime complexity* $\mathrm{erc}_{\to} : \mathbb{N} \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ *of $\to$ is*

$$\mathrm{erc}_{\to}(n) = \sup\{\mathrm{edh}_{\to}(t) \mid t \in \mathcal{T}_{\mathcal{B}}, |t| \leq n\}.$$

Example 3.4.8. Reconsider the PTRS $\mathcal{P}_{\mathsf{coin}}$ from Example 3.4.2, which models a coin flip. The defined symbols for $\mathcal{P}_{\mathsf{coin}}$ are $\mathcal{D} = \{\mathsf{g}\}$ and the constructors are $\mathcal{C} = \varnothing$. The set of basic terms $\mathcal{T}_{\mathcal{B}}$ is $\{\mathsf{g}(x) \mid x \in \mathcal{V}\}$. Since $\mathrm{edh}_{\mathcal{P}_{\mathsf{coin}}}(\mathsf{g}(x)) = 2$, we have $\iota(\mathrm{erc}_{\mathcal{P}_{\mathsf{coin}}}) = \mathbf{Pol}_0$ as the expected runtime complexity. If we consider arbitrary start terms, then we can start with a term like $\mathsf{g}^n(x)$ with $\mathrm{edh}_{\mathcal{P}_{\mathsf{coin}}}(\mathsf{g}^n(x)) = 2n$. Therefore, the expected derivational complexity of $\mathcal{P}_{\mathsf{coin}}$ is linear, i.e., $\iota(\mathrm{edc}_{\mathcal{P}_{\mathsf{coin}}}) = \mathbf{Pol}_1$.

Example 3.4.9. For the PTRS $\mathcal{P}_{\mathsf{geo}}$ from Example 3.4.3 we have a similar result. The defined symbols for $\mathcal{P}_{\mathsf{geo}}$ are $\mathcal{D} = \{\mathsf{geo}\}$ and the constructors are $\mathcal{C} = \{\mathsf{s}, \mathsf{0}\}$, where we additionally assume the existence of the constant constructor $\mathsf{0}$ that does not occur in the rules of $\mathcal{P}_{\mathsf{geo}}$. The set of basic terms $\mathcal{T}_{\mathcal{B}}$ is $\{\mathsf{geo}(\mathsf{s}^n(\mathsf{0})), \mathsf{geo}(\mathsf{s}^n(x)) \mid n \in \mathbb{N}, x \in \mathcal{V}\}$. We have $\mathrm{edh}_{\mathcal{P}_{\mathsf{geo}}}(\mathsf{geo}(t)) = 2$ for every constructor term $t \in \mathcal{T}(\mathcal{C}, \mathcal{V})$. Therefore, $\iota(\mathrm{erc}_{\mathcal{P}_{\mathsf{geo}}}) = \mathbf{Pol}_0$. If we consider arbitrary start terms, then we can start with a term like $\mathsf{geo}^n(x)$ with $\mathrm{edh}_{\mathcal{P}_{\mathsf{geo}}}(\mathsf{geo}^n(x)) = 2n$, and we have $\iota(\mathrm{edc}_{\mathcal{P}_{\mathsf{geo}}}) = \mathbf{Pol}_1$.

### 3.4.3 Evaluation Strategies

Since rule selection remains as for TRSs and we only incorporate probabilities after the selection, (positional) evaluation strategies for PTRSs can be defined analogously to those for TRSs. In this subsection, we restate all important evaluation strategies for PTRSs and give some easy examples.

Definition 3.4.10 – Evaluation Strategy.

*Let $\mathcal{P}$ be a PTRS and $t \in \mathcal{T}$. We denote the set of positions in $t$ at which we have a redex w.r.t. $\mathcal{P}$ by*

$$\mathrm{pos}^{\mathrm{red}}_{\mathcal{P}}(t) = \{\pi \in \mathrm{pos}(t) \mid \exists \sigma.\ \exists \ell \to \mu \in \mathcal{P}.\ t|_\pi = \ell\sigma\}.$$

*An evaluation strategy is a function $\mathtt{s} : \mathcal{T} \to \mathrm{Pow}(\mathbb{N}^*)$ such that $\mathtt{s}(t) \subseteq \mathrm{pos}^{\mathrm{red}}_{\mathcal{P}}(t)$. The PTRS $\mathcal{P}$ together with the evaluation strategy $\mathtt{s}$ induces a PARS $(\mathcal{T}, \xrightarrow{\mathtt{s}}_{\mathcal{P}})$ with the probabilistic relation $\xrightarrow{\mathtt{s}}_{\mathcal{P}} \subseteq \mathcal{T} \times \mathrm{FDist}(\mathcal{T})$ via $s \xrightarrow{\mathtt{s}}_{\mathcal{P}} \{{}^{p_1}t_1, \ldots, {}^{p_k}t_k\}$ if $\pi \in \mathtt{s}(s)$, $s|_\pi = \ell\sigma$, and $t_j = s[r_j\sigma]_\pi$ for all $1 \leq j \leq k$, for a rule $\ell \to \{{}^{p_1}r_1, \ldots, {}^{p_k}r_k\} \in \mathcal{P}$ and a substitution $\sigma$.*

Definition 3.4.11 – Full Rewriting.

*Let $\mathcal{P}$ be a PTRS and $t \in \mathcal{T}$. The full evaluation strategy is defined as the function $\mathbf{f} : \mathcal{T} \to \mathrm{Pow}(\mathbb{N}^*)$ with $\mathbf{f}(t) = \mathrm{pos}^{\mathrm{red}}_{\mathcal{P}}(t)$. The full rewrite relation is denoted by $\xrightarrow{\mathbf{f}}_{\mathcal{P}} = \to_{\mathcal{P}}$.*

Definition 3.4.12 – Innermost Rewriting.

*Let $\mathcal{P}$ be a PTRS and $t \in \mathcal{T}$. The innermost evaluation strategy is defined as the function $\mathbf{i} : \mathcal{T} \to \mathrm{Pow}(\mathbb{N}^*)$ with $\mathbf{i}(t) = \{\pi \in \mathrm{pos}^{\mathrm{red}}_{\mathcal{P}}(t) \mid \neg\exists \pi' \in \mathrm{pos}^{\mathrm{red}}_{\mathcal{P}}(t).\ \pi < \pi'\}$. The innermost rewrite relation is denoted by $\xrightarrow{\mathbf{i}}_{\mathcal{P}}$.*

Example 3.4.13. Consider the PTRS $\mathcal{P}_{\mathsf{rw}}$ from Example 3.4.2. An innermost $\rightrightarrows_{\mathcal{P}_{\mathsf{rw}}}$-reduction is $\{{}^1\mathsf{g}(\mathsf{c}(\underline{\mathsf{g}}(x)))\} \xrightarrow{\mathbf{i}}_{\mathcal{P}_{\mathsf{rw}}} \{{}^{1/2}\mathsf{g}(\mathsf{c}(x)), {}^{1/2}\mathsf{g}(\mathsf{c}(\mathsf{g}(\mathsf{b}(\mathsf{g}(x))))))\}$.

Definition 3.4.14 – Outermost Rewriting.

*Let $\mathcal{P}$ be a PTRS and $t \in \mathcal{T}$. The outermost evaluation strategy is defined as the function $\mathbf{o} : \mathcal{T} \to \mathrm{Pow}(\mathbb{N}^*)$ with $\mathbf{o}(t) = \{\pi \in \mathrm{pos}^{\mathrm{red}}_{\mathcal{P}}(t) \mid \neg\exists \pi' \in \mathrm{pos}^{\mathrm{red}}_{\mathcal{P}}(t).\ \pi' < \pi\}$. The outermost rewrite relation is denoted by $\xrightarrow{\mathbf{o}}_{\mathcal{P}}$.*

Example 3.4.15. Consider the PTRS $\mathcal{P}_{\mathsf{rw}}$ from Example 3.4.2. The $\rightrightarrows_{\mathcal{P}_{\mathsf{rw}}}$-reduction from Example 3.4.13 is not an outermost reduction. Instead, an outermost $\rightrightarrows_{\mathcal{P}_{\mathsf{rw}}}$-reduction is $\{{}^1\underline{\mathsf{g}}(\mathsf{c}(\mathsf{g}(x)))\} \xrightarrow{\mathbf{o}}_{\mathcal{P}_{\mathsf{rw}}} \{{}^{1/2}\mathsf{c}(\mathsf{g}(x)), {}^{1/2}\mathsf{g}(\mathsf{b}(\mathsf{g}(\mathsf{c}(\mathsf{g}(x))))))\}$.

Definition 3.4.16 – Leftmost-Innermost Rewriting.

*Let $\mathcal{P}$ be a PTRS and $t \in \mathcal{T}$. The leftmost-innermost evaluation strategy is defined as the function $\mathbf{li} : \mathcal{T} \to \mathrm{Pow}(\mathbb{N}^*)$ with $\mathbf{li}(t) = \{\pi \in \mathbf{i}(t) \mid \neg\exists \pi' \in \mathbf{i}(t).\ \pi' \prec \pi\}$. The leftmost-innermost rewrite relation is denoted by $\xrightarrow{\mathbf{li}}_{\mathcal{P}}$.*

Example 3.4.17. Consider the PTRS $\mathcal{P}_{\mathsf{rw}}$ from Example 3.4.2. A leftmost-innermost $\rightrightarrows_{\mathcal{P}_{\mathsf{rw}}}$-reduction is $\{{}^1\mathsf{d}(\underline{\mathsf{g}}(x), \mathsf{g}(x))\} \xrightarrow{\mathbf{li}}_{\mathcal{P}_{\mathsf{rw}}} \{{}^{1/2}\mathsf{d}(x, \mathsf{g}(x)), {}^{1/2}\mathsf{d}(\mathsf{g}(\mathsf{b}(\mathsf{g}(x))), \mathsf{g}(x))\}$.

We present novel results regarding the relations between these positional evaluation strategies with respect to `AST`, `PAST`, `SAST`, and expected complexity in Chapter 4. In the following chapters, we also use the abbreviations `iAST`, `iPAST`, and `iSAST` for $\mathtt{AST}(\xrightarrow{\mathbf{i}}_{\mathcal{P}})$, $\mathtt{PAST}(\xrightarrow{\mathbf{i}}_{\mathcal{P}})$, and $\mathtt{SAST}(\xrightarrow{\mathbf{i}}_{\mathcal{P}})$, i.e., for the respective property w.r.t. innermost evaluation, and we often say *full* `AST` instead of $\mathtt{AST}(\xrightarrow{\mathbf{f}}_{\mathcal{P}})$ to distinguish it from `iAST`.

### 3.4.4 PAST vs. SAST

While there is a clear difference between PAST and SAST when it comes to programs that allow non-deterministic choices, we have not given any example of a PTRS $\mathcal{P}$ such that $\mathtt{PAST}(\mathcal{P})$ and $\neg\mathtt{SAST}(\mathcal{P})$. In fact, coming up with an example of a PTRS $\mathcal{P}$ that uses only finitely many rules turns out to be rather difficult (compared to the PARS from Counterexample 3.3.29 that uses infinitely many rules). We show in this subsection that PTRSs have so much inherent non-determinism that the difference between PAST and SAST is very thin. In fact, PAST and SAST are equal for nearly all PTRSs that arise in practice. To be precise, we have $\mathtt{PAST}(\mathcal{P}) \iff \mathtt{SAST}(\mathcal{P})$ for all finite PTRSs $\mathcal{P}$ over a signature that contains at least one function symbol of arity at least 2. Yet, there is a difference between PAST and SAST even for *finite* PTRSs.

Counterexample 3.4.18. Consider the PTRS $\mathcal{P}_{\mathsf{unary}}$ with the rules

$$\begin{aligned}
\mathsf{f}(x) &\to \{{}^{1/2}\mathsf{f}(\mathsf{s}(x)), {}^{1/2}\mathsf{b}\} & \mathsf{h}(\mathsf{s}(x)) &\to \{{}^{1}\mathsf{q}(\mathsf{h}(x))\}\\
\mathsf{f}(x) &\to \{{}^{1}\mathsf{g}(x)\} & \mathsf{h}(\mathsf{0}) &\to \{{}^{1}\mathsf{a}(\mathsf{a}(\mathsf{a}(\mathsf{a}(\mathsf{0}))))\}\\
\mathsf{g}(\mathsf{0}) &\to \{{}^{1}\mathsf{0}\} & \mathsf{q}(\mathsf{a}(x)) &\to \{{}^{1}\mathsf{q}_1(x)\}\\
\mathsf{g}(\mathsf{s}(x)) &\to \{{}^{1}\mathsf{g}_1(\mathsf{s}(x))\} & \mathsf{q}_1(x) &\to \{{}^{1}\mathsf{q}_2(x)\}\\
\mathsf{g}_1(\mathsf{s}(x)) &\to \{{}^{1}\mathsf{h}(\mathsf{s}(x))\} & \mathsf{q}_2(x) &\to \{{}^{1}\mathsf{a}(\mathsf{a}(\mathsf{a}(\mathsf{q}(x)))))\}
\end{aligned}$$

and the following rewrite sequence:

$$\begin{aligned}
\{{}^{1}\underline{\mathsf{g}}(\mathsf{s}^n(\mathsf{0}))\} &\Rightarrow^{2}_{\mathcal{P}_{\mathsf{unary}}} \{{}^{1}\underline{\mathsf{h}}(\mathsf{s}^n(\mathsf{0}))\} &&\Rightarrow^{n}_{\mathcal{P}_{\mathsf{unary}}} \{{}^{1}\mathsf{q}^n(\underline{\mathsf{h}}(\mathsf{0}))\}\\
&\Rightarrow_{\mathcal{P}_{\mathsf{unary}}} \{{}^{1}\underline{\mathsf{q}}^n(\mathsf{a}^4(\mathsf{0}))\} &&\Rightarrow^{3\cdot 4}_{\mathcal{P}_{\mathsf{unary}}} \{{}^{1}\underline{\mathsf{q}}^{n-1}(\mathsf{a}^{4^2}(\mathsf{q}(\mathsf{0})))\}\\
&\Rightarrow^{3\cdot 4^2}_{\mathcal{P}_{\mathsf{unary}}} \{{}^{1}\underline{\mathsf{q}}^{n-2}(\mathsf{a}^{4^3}(\mathsf{q}^2(\mathsf{0})))\} &&\Rightarrow^{3\cdot 4^3}_{\mathcal{P}_{\mathsf{unary}}} \cdots\\
&\Rightarrow^{3\cdot 4^{n-1}}_{\mathcal{P}_{\mathsf{unary}}} \{{}^{1}\underline{\mathsf{q}}(\mathsf{a}^{4^n}(\mathsf{q}^{n-1}(\mathsf{0})))\} &&\Rightarrow^{3\cdot 4^n}_{\mathcal{P}_{\mathsf{unary}}} \{{}^{1}\mathsf{a}^{4^{n+1}}(\mathsf{q}^n(\mathsf{0}))\}
\end{aligned}$$

Thus, for every $n \in \mathbb{N}$ with $n > 0$, the term $\mathsf{g}(\mathsf{s}^n(\mathsf{0}))$ has an expected derivation height of $2 + n + 1 + 3 \cdot \sum_{i=1}^{n} 4^i = n + 3 \cdot \sum_{i=0}^{n} 4^i = n + 3 \cdot \frac{4^{n+1}-1}{3} = 4^{n+1} - 1 + n$.

Consider the possible derivations of $\mathsf{f}(\mathsf{0})$. If we never use the rule from $\mathsf{f}$ to $\mathsf{g}$, then the expected derivation length is $1/2 \cdot 1 + 1/4 \cdot 2 + 1/8 \cdot 3 + \cdots = \sum_{i=1}^{\infty} i/2^i = 2$ and if we never use the probabilistic $\mathsf{f}$-rule, then the derivation length is 2 as well. Otherwise, if we use the probabilistic $\mathsf{f}$-rule in the first $k \in \mathbb{N}$ steps with $k > 0$ and the rule from $\mathsf{f}$ to $\mathsf{g}$ in the $(k+1)$-th step, then the expected derivation length is $(\sum_{i=1}^{k} i/2^i) + \frac{k}{2^k} + (1/2)^k + \frac{4^{k+1}-1+k}{2^k} = (\sum_{i=1}^{k} i/2^i) + 2^{k+2} + \frac{2\cdot k}{2^k}$. Thus, for every rewrite sequence starting with $\mathsf{f}(\mathsf{0})$, the expected derivation length is finite. However, for any $k > 0$, there is a rewrite sequence starting with $\mathsf{f}(\mathsf{0})$ whose expected derivation length is $(\sum_{i=1}^{k} i/2^i) + 2^{k+2} + \frac{2\cdot k}{2^k} \geq 2^{k+2}$. In other words, although every rewrite sequence starting with $\mathsf{f}(\mathsf{0})$ has finite expected derivation length, the supremum over the expected derivation lengths of all these rewrite sequences is infinite. Thus, we do not have $\mathtt{SAST}(\mathcal{P}_{\mathsf{unary}})$, as $\mathsf{f}(\mathsf{0})$ has infinite expected derivation height, but $\mathtt{PAST}(\mathcal{P}_{\mathsf{unary}})$ holds, as every rewrite sequence starting with $\mathsf{f}(\mathsf{0})$ has finite expected derivation length and a similar argument holds for every other start term.

The PTRS in Counterexample 3.4.18 is of a very specific form: Its signature contains only unary symbols and constants, and the probabilistic rule $\mathsf{f}(x) \to \{{}^{1/2}\mathsf{f}(\mathsf{s}(x)), {}^{1/2}\mathsf{b}\}$ is erasing. If we remove one of these properties, then PAST no longer holds, see Counterexample 3.4.19.

Counterexample 3.4.19. Reconsider $\mathcal{P}_{\mathsf{unary}}$ from Counterexample 3.4.18. If we extend the signature by a binary symbol $\mathsf{c}$, then $\mathtt{PAST}(\mathcal{P}_{\mathsf{unary}})$ does not hold anymore, as we can start with the term $\mathsf{c}(\mathsf{f}(0),\mathsf{f}(0))$ and consider the following rewrite sequence.

$$\begin{aligned}
&\{{}^{1}\mathsf{c}(\underline{\mathsf{f}}(0),\mathsf{f}(0))\}\\
\Rightarrow_{\mathcal{P}_{\mathsf{unary}}}\ &\{{}^{1/2}\mathsf{c}(\mathsf{b},\mathsf{f}(0)),\ {}^{1/2}\mathsf{c}(\underline{\mathsf{f}}(\mathsf{s}(0)),\mathsf{f}(0))\}\\
\Rightarrow_{\mathcal{P}_{\mathsf{unary}}}\ &\{\quad\dots\quad,\ {}^{1/4}\mathsf{c}(\mathsf{b},\mathsf{f}(0)),\ {}^{1/4}\mathsf{c}(\underline{\mathsf{f}}(\mathsf{s}^2(0)),\mathsf{f}(0))\}\\
\Rightarrow_{\mathcal{P}_{\mathsf{unary}}}\ &\{\quad\dots\quad,\quad\dots\quad,\ {}^{1/8}\mathsf{c}(\mathsf{b},\mathsf{f}(0)),\ {}^{1/8}\mathsf{c}(\underline{\mathsf{f}}(\mathsf{s}^3(0)),\mathsf{f}(0))\}
\end{aligned}$$

Here, we use the term $\mathsf{f}(0)$ in $\mathsf{c}$'s first argument to create infinitely many copies of the term $\mathsf{f}(0)$ in $\mathsf{c}$'s second argument using the probabilistic $\mathsf{f}$-rule. The green terms are not yet normal forms, as they contain the subterm $\mathsf{f}(0)$, and will be evaluated further. As in Counterexample 3.4.18, for each $k > 0$, there is a rewrite sequence starting with $\mathsf{f}(0)$ whose expected derivation length is at least $2^{k+2}$. We can choose $k = 1$ for the first green term, $k = 2$ for the second, etc., leading to a final expected derivation length of at least $\sum_{i=1}^{\infty} (1/2)^i \cdot 2^{i+2} = \sum_{i=1}^{\infty} 4$, which diverges to infinity. Hence, $\mathtt{PAST}(\mathcal{P}_{\mathsf{unary}})$ does not hold over the extended signature. In particular, this shows that in contrast to termination for non-probabilistic TRSs, PAST is not preserved under signature extensions.

Similarly, if we consider $\mathcal{P}'_{\mathsf{unary}}$ with the non-erasing rule $\mathsf{f}(x) \to \{{}^{1/2}\mathsf{f}(\mathsf{s}(x)),\ {}^{1/2}\mathsf{b}(x)\}$ using a unary function symbol $\mathsf{b}$ instead of $\mathsf{f}(x) \to \{{}^{1/2}\mathsf{f}(\mathsf{s}(x)),\ {}^{1/2}\mathsf{b}\}$, then we obtain the following rewrite sequence

$$\begin{aligned}
&\{{}^{1}\underline{\mathsf{f}}(\mathsf{f}(0))\}\\
\Rightarrow_{\mathcal{P}'_{\mathsf{unary}}}\ &\{{}^{1/2}\mathsf{b}(\mathsf{f}(0)),\ {}^{1/2}\underline{\mathsf{f}}(\mathsf{s}(\mathsf{f}(0)))\}\\
\Rightarrow_{\mathcal{P}'_{\mathsf{unary}}}\ &\{\quad\dots\quad,\ {}^{1/4}\mathsf{b}(\mathsf{s}(\mathsf{f}(0))),\ {}^{1/4}\underline{\mathsf{f}}(\mathsf{s}^2(\mathsf{f}(0)))\}\\
\Rightarrow_{\mathcal{P}'_{\mathsf{unary}}}\ &\{\quad\dots\quad,\quad\dots\quad,\ {}^{1/8}\mathsf{b}(\mathsf{s}^2(\mathsf{f}(0))),\ {}^{1/8}\underline{\mathsf{f}}(\mathsf{s}^3(\mathsf{f}(0)))\}
\end{aligned}$$

Again, we can extend this sequence to an infinite rewrite sequence with an infinite expected derivation length. Hence, $\mathtt{PAST}(\mathcal{P}'_{\mathsf{unary}})$ does not hold either.

One might wonder whether the following rewrite sequence already works for $\mathcal{P}_{\mathsf{unary}}$ (with the rule $\mathsf{f}(x) \to \{{}^{1/2}\mathsf{f}(\mathsf{s}(x)),\ {}^{1/2}\mathsf{b}\}$) to disprove $\mathtt{PAST}(\mathcal{P}_{\mathsf{unary}})$.

$$\begin{aligned}
&\{{}^{1}\mathsf{f}(\underline{\mathsf{f}}(0))\}\\
\Rightarrow_{\mathcal{P}_{\mathsf{unary}}}\ &\{{}^{1/2}\mathsf{f}(\mathsf{b}),\ {}^{1/2}\mathsf{f}(\underline{\mathsf{f}}(\mathsf{s}(0)))\}\\
\Rightarrow_{\mathcal{P}_{\mathsf{unary}}}\ &\{\quad\dots\quad,\ {}^{1/4}\mathsf{f}(\mathsf{b}),\ {}^{1/4}\mathsf{f}(\underline{\mathsf{f}}(\mathsf{s}^2(0)))\}\\
\Rightarrow_{\mathcal{P}_{\mathsf{unary}}}\ &\{\quad\dots\quad,\quad\dots\quad,\ {}^{1/8}\mathsf{f}(\mathsf{b}),\ {}^{1/8}\mathsf{f}(\underline{\mathsf{f}}(\mathsf{s}^3(0)))\}
\end{aligned}$$

However, this is not the case, because the $0$ below the $\mathsf{f}$ symbol is crucial for the exponential runtime. The term $\mathsf{f}(\mathsf{b})$ without any $0$ only has the finite expected derivation height $\mathrm{edh}_{\mathcal{P}_{\mathsf{unary}}}(\mathsf{f}(\mathsf{b})) = 3$. Thus, every infinite extension of this rewrite sequence has finite expected derivation length.

Counterexample 3.4.18 and Counterexample 3.4.19 show that PAST is not preserved under signature extensions, as the first rewrite sequence given in Counterexample 3.4.19 is in fact a leftmost-innermost rewrite sequence.

**Theorem 3.4.20** – Signature Extensions for PAST.

There exist a finite PTRS $\mathcal{P}$ and signatures $\Sigma, \Sigma'$ with $\Sigma \subset \Sigma'$ such that $\mathtt{PAST}(\mathcal{P})$ holds over the signature $\Sigma$, but not over $\Sigma'$. The same holds for $\mathtt{PAST}(\xrightarrow[\mathcal{P}]{\mathsf{i}})$ instead of PAST.

In contrast, when analyzing modularity of AST and SAST in Chapter 5, we show that they are both closed under signature extensions.

The core idea of both examples in Counterexample 3.4.19 is that in the limit we can reach an infinite multi-distribution with infinitely many entries whose terms contain a term like $\mathsf{f}(0)$ with unbounded expected derivation height. A PTRS that allows such sequences is said to *admit infinite splits*.

Definition 3.4.21 – Infinite Splits.

*A PTRS $\mathcal{P}$ admits infinite splits if for every term $t \in \mathcal{T}$ there exists a term $t' \in \mathcal{T}$ and a $\to_{\mathcal{P}}$-RT that starts with $t'$ such that there are infinitely many leaves $x$ labeled with terms $t_x$ that have $t$ as an occurrence, i.e., $t \trianglelefteq t_x$.*

Example 3.4.22. $\mathcal{P}_{\mathsf{unary}}$ over the signature containing a function symbol of arity $\geq 2$ and the non-erasing PTRS $\mathcal{P}'_{\mathsf{unary}}$ both admit infinite splits. However, $\mathcal{P}_{\mathsf{unary}}$ over the signature $\Sigma = \Sigma_0 \uplus \Sigma_1 = \{0, \mathsf{b}\} \uplus \{\mathsf{f}, \mathsf{s}, \mathsf{g}, \mathsf{g}_1, \mathsf{h}, \mathsf{a}, \mathsf{q}, \mathsf{q}_1, \mathsf{q}_2\}$ does not admit infinite splits.

As shown by the following theorem, admitting infinite splits implies that $\mathtt{PAST}(\mathcal{P})$ is the same as $\mathtt{SAST}(\mathcal{P})$.

Theorem 3.4.23 – Equivalence of PAST and SAST via Infinite Splits.

If a PTRS $\mathcal{P}$ admits infinite splits, then $\mathtt{PAST}(\mathcal{P}) \iff \mathtt{SAST}(\mathcal{P})$.

*Proof.* We only have to prove "$\Longrightarrow$", and do this via contraposition. Assume that $\mathtt{SAST}(\mathcal{P})$ does not hold. Then by Corollary 3.3.41, there exists a term $t \in \mathcal{T}$ such that for every $n \in \mathbb{N}$ there exists a $\to_{\mathcal{P}}$-RT $\mathfrak{T}^t_n$ starting with $t$ such that $\mathrm{edl}(\mathfrak{T}^t_n) \geq n$. We now construct a single $\to_{\mathcal{P}}$-RT $\mathfrak{T}^\infty$ with $\mathrm{edl}(\mathfrak{T}^\infty) = \infty$, which implies that $\mathtt{PAST}(\mathcal{P})$ does not hold either by Corollary 3.3.40. Since $\mathcal{P}$ admits infinite splits, there exists a term $t' \in \mathcal{T}$ and a $\to_{\mathcal{P}}$-RT $\mathfrak{T}$ whose root is labeled with ${}^1t'$ such that there are infinitely many leaves whose corresponding terms have $t$ as an occurrence. Let $v$ be a leaf in $\mathfrak{T}$ with $t^{\mathfrak{T}}_v = C[t\delta]$ for some context $C$ and substitution $\delta$, and let $n_v \in \mathbb{N}$ be such that $p^{\mathfrak{T}}_v \geq \frac{1}{2^{n_v}}$. Then we can replace the leaf by the $\to_{\mathcal{P}}$-RT $\mathfrak{T}^{C[t\delta]}_{2^{n_v}}$. Here, $\mathfrak{T}^{C[t\delta]}_{2^{n_v}}$ is the same tree as $\mathfrak{T}^t_{2^{n_v}}$ where in addition we instantiate the term of every node with $\delta$ and add the context $C$ around it (this is again a $\to_{\mathcal{P}}$-RT, as $\to_{\mathcal{P}}$ is closed under substitutions and contexts). Hence, $\mathrm{edl}(\mathfrak{T}^{C[t\delta]}_{2^{n_v}}) = \mathrm{edl}(\mathfrak{T}^t_{2^{n_v}}) \geq 2^{n_v}$. Let $\mathfrak{T}^\infty$ be the $\to_{\mathcal{P}}$-RT that results from performing this replacement for every leaf $v$ in $\mathfrak{T}$ that contains $t$ as an occurrence. Then, for $\mathfrak{T}^\infty$ we have

$$\begin{aligned} \mathrm{edl}(\mathfrak{T}^\infty) &\geq \sum_{v \in \mathrm{Leaf}(\mathfrak{T}),\, t^{\mathfrak{T}}_v = C[t\delta] \text{ for some context } C} p^{\mathfrak{T}}_v \cdot \mathrm{edl}(\mathfrak{T}^{C[t\delta]}_{2^{n_v}}) \\ &\geq \sum_{v \in \mathrm{Leaf}(\mathfrak{T}),\, t^{\mathfrak{T}}_v = C[t\delta]} \tfrac{1}{2^{n_v}} \cdot 2^{n_v} \;=\; \sum_{v \in \mathrm{Leaf}(\mathfrak{T}),\, t^{\mathfrak{T}}_v = C[t\delta]} 1 \;=\; \infty. \end{aligned}$$

*Proof* *QED*

Theorem 3.4.23 can also be adapted to strategies like innermost or leftmost-innermost rewriting, but then one has to ensure that in the infinitely many leaves of the RT in Definition 3.4.21 the subterm $t$ can be used as the next redex according to the respective strategy.

As an application of Theorem 3.4.23 we give two syntactical criteria that ensure that a PTRS $\mathcal{P}$ admits infinite splits, where both criteria are easy to check automatically. The first one (illustrated by the first PTRS in Counterexample 3.4.19) states that if we consider only PTRSs with finitely many rules, then the existence of a function symbol of arity at least 2 suffices. Thus, this novel observation shows that for almost all finite

PTRSs in practice, there is no difference between $\mathtt{PAST}(\mathcal{P})$ and $\mathtt{SAST}(\mathcal{P})$.

Theorem 3.4.24 – Equivalence of PAST and SAST (1).

If a PTRS $\mathcal{P}$ over a signature $\Sigma$ has only finitely many rules and $\Sigma$ contains a function symbol of arity at least 2, then $\mathtt{PAST}(\mathcal{P}) \iff \mathtt{SAST}(\mathcal{P})$.

*Proof.* Let $\mathcal{P}$ contain only finitely many rules and assume for a contradiction that we have $\mathtt{PAST}(\mathcal{P})$ but not $\mathtt{SAST}(\mathcal{P})$. Then there exists a term $t'$ and a $\to_{\mathcal{P}}$-RT whose root is labeled with ${}^{1}t'$ which has infinitely many leaves. (If there were only finitely many leaves for every term, then we would either have $\mathtt{SAST}(\mathcal{P})$ if there exists no infinite path (by König's Lemma, since $\mathcal{P}$ has only finitely many rules), or we would not have $\mathtt{PAST}(\mathcal{P})$ if there exists an infinite path.) Let $t \in \mathcal{T}$ be an arbitrary term, and $\mathtt{c}$ be a function symbol of arity $\geq 2$. We can now construct a $\to_{\mathcal{P}}$-RT that starts with $\mathtt{c}(t', t, x, \ldots, x)$ where $x \in \mathcal{V}$ such that there are infinitely many leaves labeled with terms that have $t$ as an occurrence. To do so, one simply restricts rewriting to the first argument of $\mathtt{c}$. Note that this construction is only possible because we assume that there is a function symbol of arity at least 2, such that both $t$ and $t'$ can be subterms of the same term $\mathtt{c}(t', t, \ldots)$. This shows that $\mathcal{P}$ admits infinite splits, and thus, we have $\mathtt{SAST}(\mathcal{P})$ by Theorem 3.4.23, which is our desired contradiction. *QED*

While Theorem 3.4.24 works for most PTRSs in practice, if one considers probabilistic *string rewriting* where one only considers signatures $\Sigma = \Sigma_1$ of unary function symbols, then Theorem 3.4.24 is generally not applicable. The second sufficient criterion for equivalence of $\mathtt{PAST}(\mathcal{P})$ and $\mathtt{SAST}(\mathcal{P})$ (illustrated by the second PTRS in Counterexample 3.4.19) requires specific forms of "non-erasing loops", and it also works for string rewriting.

Theorem 3.4.25 – Equivalence of PAST and SAST (2).

If a PTRS $\mathcal{P}$ contains a probabilistic rule $\ell \to \{{}^{p_1}C[\ell\sigma], {}^{p_2}s, \ldots\}$ such that there is a variable $x \in \mathcal{V}(\ell)$ with $x \in \mathcal{V}(x\sigma) \cap \mathcal{V}(s)$, then $\mathtt{PAST}(\mathcal{P}) \iff \mathtt{SAST}(\mathcal{P})$.

*Proof.* Let $t \in \mathcal{T}$ be an arbitrary term, and let $\delta$ be a substitution such that $x\delta = t$ for a variable $x \in \mathcal{V}(x\sigma) \cap \mathcal{V}(s)$. We can now construct a $\to_{\mathcal{P}}$-RT that starts with $\ell\delta$ such that there are infinitely many leaves labeled with terms that have $t$ as a subterm. To do so, one rewrites the redex $\ell\delta$ with the rule $\ell \to \{{}^{p_1}C[\ell\sigma], {}^{p_2}s, \ldots\}$, leading to a new leaf labeled with the term $s\delta$ (containing the subterm $t$ since $x \in \mathcal{V}(s)$) and a new node labeled with the term $C[\ell\sigma]\delta = C\delta[\ell\sigma\delta]$, where we can rewrite the redex $\ell\sigma\delta$ again. This in turn will lead to a leaf labeled with $C\delta[s\sigma\delta]$ (which again contains $t$ since $x \in \mathcal{V}(x\sigma) \cap \mathcal{V}(s)$ implies $x \in \mathcal{V}(s\sigma)$) and a new node labeled with a term containing $\ell\sigma^2\delta$, etc. Hence, $\mathcal{P}$ admits infinite splits and the theorem is implied by Theorem 3.4.23. *QED*

Theorem 3.4.25 can even be extended so that the loop does not consist of a single rewrite step, but we can have arbitrarily many steps during the loop.

### 3.4.5 Proving Termination and Complexity via Interpretations

We end this section by recapitulating the first technique to analyze SAST of PTRSs, which was presented in [ADLY20]. Additionally, we extend this result to compute upper bounds on the expected derivational and expected runtime complexity of the PTRS. Moreover, we present the first technique to analyze AST of PTRSs. Similar

to the approach in Section 3.2.5 both of these techniques use ordered algebras to orient the rules. Here, the notion of a ranking function is replaced by the notion of a *supermartingale*, which provides a quantitative measure of expected decrease. In both cases, we can use polynomial interpretations to fully automatically search for an interpretation that satisfies the conditions. The search for these polynomials is done via SMT solvers.

Note that the techniques we develop for analyzing PTRSs in this thesis are designed for *finite* PTRSs. Therefore, whenever we present a concrete proof technique, we assume that the analyzed PTRS is finite. More precisely, the results on evaluation strategies in Chapter 4 hold for arbitrary, possibly infinite PTRSs, whereas from Chapter 5 onwards, we always assume that all PTRSs are finite.

**Proving SAST Via Interpretations** While Theorem 3.2.62 has no further restriction on the ordered $\Sigma$-algebras except monotonicity, in the probabilistic setting we have to deal with probabilities and we have to consider algebras that allow the definition of an *expected value*. Therefore, we have to restrict ourselves to ordered $\Sigma$-algebras $(\mathfrak{I}, \succ)$ with carrier $A$ that form an ordered $\mathbb{R}_{\geq 0}$-semimodule, i.e., in addition to the order $\succ$ we have to have access to an addition $\oplus : (A \times A) \to A$ and a scalar multiplication $\odot : (\mathbb{R}_{\geq 0} \times A) \to A$. Using the addition and the scalar multiplication, we can then define the expected value $\mathbb{E}_{\mathfrak{I}}(\mu)$ of a distribution $\mu = \{{}^{p_1}t_1, \dots, {}^{p_k}t_k\}$ by $\mathbb{E}_{\mathfrak{I}}(\mu) = p_1 \odot \mathfrak{I}(t_1) \oplus \dots \oplus p_k \odot \mathfrak{I}(t_k) = \bigoplus_{i=1}^{k} p_i \odot \mathfrak{I}(t_i)$.

Definition 3.4.26 – $\mathbb{R}_{\geq 0}$-Semimodule.

*An $\mathbb{R}_{\geq 0}$-semimodule $(A, \oplus, \odot, \mathbf{0})$ (or simply $A$) is a set $A$ (called the universe) together with two binary functions $\oplus : (A \times A) \to A$ (called addition) and $\odot : (\mathbb{R}_{\geq 0} \times A) \to A$ (called scalar multiplication) such that $(A, \oplus, \mathbf{0})$ is a commutative monoid ($\oplus$ is commutative and associative with identity element $\mathbf{0}$), and such that for all $a, b \in A$, $r, s \in \mathbb{R}_{\geq 0}$ we have $r \odot (a \oplus b) = r \odot a \oplus r \odot b$, $(r+s) \odot a = r \odot a \oplus s \odot a$, $r \odot (s \odot a) = (r \cdot s) \odot a$, $1 \odot a = a$, and $0 \odot a = r \odot \mathbf{0} = \mathbf{0}$.*

Example 3.4.27. The polynomials $\mathbb{R}_{\geq 0}[\mathcal{V}]$ with non-negative real coefficients with the standard addition, the standard scalar multiplication with $\mathbb{R}_{\geq 0}$, and the standard 0 form an $\mathbb{R}_{\geq 0}$-semimodule. In particular, this semimodule contains all natural polynomials from $\mathbb{N}[\mathcal{V}]$. Another example of an $\mathbb{R}_{\geq 0}$-semimodule is the set of matrices over $\mathbb{R}_{\geq 0}$ of a fixed dimension.

Now that we have the additional operations on the carrier, we have to make sure that the order $\succ$ behaves as expected w.r.t. addition and scalar multiplication. Or in other words, the order has to be *compatible* with the semimodule operations and monotonic.

Definition 3.4.28 – Compatible Order.

*Let $(A, \oplus, \odot, \mathbf{0})$ be an $\mathbb{R}_{\geq 0}$-semimodule. A relation $\succ \subseteq A \times A$ is compatible with the semimodule structure of $(A, \oplus, \odot, \mathbf{0})$ if $x \succ y$ implies $x \oplus z \succ y \oplus z$ for all $z \in A$ and $r \odot x \succ r \odot y$ for all $r \in \mathbb{R}_{>0}$. Addition is monotonic w.r.t. $\succ$ if for all $x, y \in A$ we have $x \oplus y \succcurlyeq x$ and $x \oplus y \succ x$ if $y \neq \mathbf{0}$.*

Monotonicity of addition w.r.t. $\succ$ implies that $\mathbf{0}$ is the minimal element, i.e., we have $a \succcurlyeq \mathbf{0}$ for every $a \in A$.

Note that in the probabilistic setting we consider polynomials from $\mathbb{R}_{\geq 0}[\mathcal{V}]$ in general

(see Example 3.4.27). Even if the interpretation itself maps into $\mathbb{N}[\mathcal{V}]$, multiplying with probabilities and forming expected values yields polynomials with coefficients from $\mathbb{R}_{\geq 0}$. Moreover, for $\epsilon > 0$ we define the order $>_\epsilon$ on $\mathbb{R}_{\geq 0}$ by $c >_\epsilon c'$ if $c \geq c' + \epsilon$, and lift it to polynomials from $\mathbb{R}_{\geq 0}[\mathcal{V}]$ by $p >_\epsilon p'$ if $p(a_1, \ldots, a_n) >_\epsilon p'(a_1, \ldots, a_n)$ for all $a_1, \ldots, a_n \in \mathbb{R}_{\geq 0}$. So, the order $>_\epsilon$ requires a *uniform* minimal gap of $\epsilon$.

Example 3.4.29. The order $>$ on the polynomials $\mathbb{R}_{\geq 0}[\mathcal{V}]$ (defined analogously to Definition 3.2.63) is compatible with the standard addition and with the standard scalar multiplication. The gap order $>_\epsilon$ is compatible with addition, but *not* with scalar multiplication: scaling both sides by a factor $< 1$ shrinks the gap below $\epsilon$. For example, $x + 1 >_1 x$, but multiplying by $1/2$ yields only $1/2x + 1/2 >_{1/2} 1/2x$, where the gap has dropped from 1 to $1/2$.

The interpretation $\mathfrak{I}(t)$ of a term $t$ that contains variables is in general not an element of the carrier $A$ itself, but an element of $A[\mathcal{V}]$, i.e., an expression over $A$ that may still contain variables from $\mathcal{V}$ (such as a polynomial in $\mathbb{N}[\mathcal{V}]$ for the polynomial interpretations). We therefore extend the order $\succ$ from $A$ to $A[\mathcal{V}]$ as in Section 3.2.5 by defining $p \succ q$ for $p, q \in A[\mathcal{V}]$ if $p\sigma \succ q\sigma$ holds for all instantiations $\sigma : \mathcal{V} \to A$ that replace the occurring variables by elements of the carrier $A$, where $p\sigma, q\sigma \in A$. Moreover, we extend $\succcurlyeq$ (the reflexive closure of $\succ$ on $A$) to $A[\mathcal{V}]$ pointwise as well.

While we previously only needed monotonicity of the interpretation for the corresponding order, in this case we have to additionally require monotonicity w.r.t. expected values. As in [ADLY20], we require *concavity* of all functions $f_\mathfrak{I}$ to ensure that we have monotonicity w.r.t. expected values. So $\mathfrak{I}(t) \succ \mathbb{E}_\mathfrak{I}(\mu)$ implies

$$\begin{aligned} \mathfrak{I}(f(\ldots, t, \ldots)) &= f_\mathfrak{I}(\ldots, \mathfrak{I}(t), \ldots) \\ &\quad\downarrow \text{ by monotonicity} \\ &\succ f_\mathfrak{I}(\ldots, \mathbb{E}_\mathfrak{I}(\mu), \ldots) \\ &\quad\downarrow \text{ by concavity} \\ &\succcurlyeq \mathbb{E}(f_\mathfrak{I}(\ldots, \mathfrak{I}(\mu), \ldots)), \end{aligned}$$

where $\mathfrak{I}(\mu) = \{{}^{p_1}\mathfrak{I}(t_1), \ldots, {}^{p_k}\mathfrak{I}(t_k)\}$ for $\mu = \{{}^{p_1}t_1, \ldots, {}^{p_k}t_k\}$.

We summarize these requirements in the following definition of barycentric $\Sigma$-algebras. In the probabilistic setting we do not require well-foundedness of $\succ$ (e.g., $>$ on $\mathbb{R}_{\geq 0}$ is not well founded), which will be handled by the collapsibility introduced below.

Definition 3.4.30 – Barycentric $\Sigma$-Algebra.

*Let $A$ be an $\mathbb{R}_{\geq 0}$-semimodule. We define the barycentric operation $\mathbb{E}_A : \mathrm{FDist}(A) \to A$ as $\mathbb{E}_A(\{{}^{p_1}a_1, \ldots, {}^{p_k}a_k\}) = p_1 \odot a_1 \oplus \ldots \oplus p_k \odot a_k = \bigoplus_{i=1}^{k} p_i \odot a_i$. We write $\mathbb{E}$ instead of $\mathbb{E}_A$ if $A$ is clear from the context. A function $f : A^n \to B$ on two $\mathbb{R}_{\geq 0}$-semimodules $A$ and $B$ is called concave w.r.t. a relation $\succcurlyeq$ on $B$ if we have $f(\ldots, \mathbb{E}_A(\mu), \ldots) \succcurlyeq \mathbb{E}_B(f(\ldots, \mu, \ldots))$ where $f(\ldots, \{{}^{p_1}a_1, \ldots, {}^{p_k}a_k\}, \ldots) = \{{}^{p_1}f(\ldots, a_1, \ldots), \ldots, {}^{p_k}f(\ldots, a_k, \ldots)\}$ for every $\mu \in \mathrm{FDist}(A)$.*

*A barycentric $\Sigma$-algebra $(\mathfrak{I}, \succ)$ is a $\Sigma$-algebra $\mathfrak{I}$ on the universe $A$ of an $\mathbb{R}_{\geq 0}$-semimodule such that $\succ \subseteq A \times A$ is transitive, compatible with the semimodule structure, and the addition is monotonic w.r.t. $\succ$. Moreover, for every $f \in \Sigma$, $f_\mathfrak{I}$ must be concave with respect to the reflexive closure $\succcurlyeq$ of $\succ$.*

*A barycentric $\Sigma$-algebra $(\mathfrak{I}, \succ)$ is called monotonic if $x \succ y$ implies $f_\mathfrak{I}(\ldots, x, \ldots) \succ f_\mathfrak{I}(\ldots, y, \ldots)$ for all $f \in \Sigma$ and $x, y \in A$. Additionally, $(\mathfrak{I}, \succ)$ is called weakly monotonic if $x \succcurlyeq y$ implies $f_\mathfrak{I}(\ldots, x, \ldots) \succcurlyeq f_\mathfrak{I}(\ldots, y, \ldots)$ for all $f \in \Sigma$ and*

$x, y \in A$.

In addition to having the structure of an $\mathbb{R}_{\geq 0}$-semimodule, we require the barycentric $\Sigma$-algebra to be *collapsible*. Collapsibility provides a real-valued measure emb on the carrier, which allows us to bound the number of strict decreases. We use this property in the proofs of Theorem 3.4.33 and Theorem 3.4.38, and we explain its necessity in the proof of Theorem 3.4.38.

Definition 3.4.31 – Collapsibility.

*Let $(\mathfrak{I}, \succ)$ be a barycentric $\Sigma$-algebra with carrier $A$. We call $(\mathfrak{I}, \succ)$ collapsible if there is a function* $\mathrm{emb} : A \to \mathbb{R}_{\geq 0}$ *such that*

(a) $a \succcurlyeq b$ *implies* $\mathrm{emb}(a) \geq \mathrm{emb}(b)$ *(monotone), and*

(b) $\mathrm{emb}(\mathbb{E}_A(\mu)) \geq \mathbb{E}(\mathrm{emb} \circ \mu)$ *for every distribution* $\mu = \{{}^{p_1}b_1, \ldots, {}^{p_k}b_k\} \in \mathrm{FDist}(A)$ *over the carrier, where* $\mathbb{E}(\mathrm{emb} \circ \mu) = \sum_{1 \leq j \leq k} p_j \cdot \mathrm{emb}(b_j)$ *(concave),*

*and such that for every $\epsilon > 0$ the $\epsilon$-decrease $\succ_\epsilon \subseteq A \times A$, defined by*

$$a \succ_\epsilon b \quad \text{if} \quad a \succ b \ \text{ and } \ \mathrm{emb}(a) \geq \mathrm{emb}(b) + \epsilon,$$

*is closed under the barycentric addition and, for monotonic $\mathfrak{I}$, under the interpretation functions, i.e., $a \succ_\epsilon b$ implies $a \oplus c \succ_\epsilon b \oplus c$ for all $c \in A$, and $f_{\mathfrak{I}}(\ldots, a, \ldots) \succ_\epsilon f_{\mathfrak{I}}(\ldots, b, \ldots)$ for all $f \in \Sigma$ if $\mathfrak{I}$ is monotonic.*

As for $\succ$, we extend $\succ_\epsilon$ from $A$ to $A[\mathcal{V}]$ by defining $p \succ_\epsilon q$ if $p\sigma \succ_\epsilon q\sigma$ holds for all instantiations $\sigma : \mathcal{V} \to A$. The $\epsilon$-decrease $\succ_\epsilon$ refines $\succ$ by definition, and it *absorbs* $\succcurlyeq$ on both sides: if $a \succcurlyeq a' \succ_\epsilon b' \succcurlyeq b$, then $a \succcurlyeq a' \succ b' \succcurlyeq b$ yields $a \succ b$, and $\mathrm{emb}(a) \geq \mathrm{emb}(a') \geq \mathrm{emb}(b') + \epsilon \geq \mathrm{emb}(b) + \epsilon$ by monotonicity of emb, so $a \succ_\epsilon b$. In contrast to $\succ$, the relation $\succ_\epsilon$ is *not* required to be closed under scalar multiplication $\odot$, and in general it cannot be, since scaling by a factor $< 1$ shrinks the gap $\mathrm{emb}(a) - \mathrm{emb}(b)$ below $\epsilon$. This is exactly why $\succ_\epsilon$ is kept separate from the compatible order $\succ$. The strict decreases in our proofs propagate only through $\oplus$ and the interpretation functions, never through $\odot$.

Natural polynomial interpretations with carrier $\mathbb{R}_{\geq 0}$ are barycentric $\Sigma$-algebras, provided that Pol maps each function symbol to a *multilinear* polynomial, to ensure concavity. Such an interpretation is moreover always collapsible via the identity embedding $\mathrm{emb} = id$. Here, the induced $\epsilon$-decrease is the gap order $>_\epsilon$ on $\mathbb{R}_{\geq 0}$, which is closed under addition, and if Pol is monotonic, then for every $f_{\mathrm{Pol}}$ and every argument position $i$, the monomial $x_i$ occurs in $f_{\mathrm{Pol}}$ with a coefficient $c_i \geq 1$ (as $f_{\mathrm{Pol}}$ is multilinear, instantiating all other arguments by 0 leaves only the monomials $x_i$ and 1, so otherwise $f_{\mathrm{Pol}}$ would not be strictly monotonic in this argument, and without multilinearity, this fails, e.g., for $f_{\mathrm{Pol}}(x_1) = x_1^2$). Hence, as $f_{\mathrm{Pol}}$ is affine in its $i$-th argument, $a >_\epsilon b$ implies $f_{\mathrm{Pol}}(\ldots, a, \ldots) - f_{\mathrm{Pol}}(\ldots, b, \ldots) \geq c_i \cdot (a - b) \geq \epsilon$, i.e., every $f_{\mathrm{Pol}}$ preserves the $\epsilon$-gap.

Definition 3.4.32 – Multilinear.

*A polynomial interpretation* Pol *is multilinear if for every $f \in \Sigma$ the polynomial $f_{\mathrm{Pol}}$ only has monomials of the form $c \cdot x_1^{e_1} \cdot \ldots \cdot x_n^{e_n}$ with $c \in \mathbb{N}$ and $e_1, \ldots, e_n \in \{0, 1\}$.*

The same holds for matrix interpretations as in [EWZ06].

If we adapt Theorem 3.2.62 to the probabilistic setting by requiring a strict decrease in expectation (and restricting the theorem to monotonic, collapsible, barycentric $\Sigma$-algebras), then we can prove SAST of PTRSs, which yields the following theorem.

Theorem 3.4.33 – Proving SAST Via Interpretations, [ADLY20].

Let $\mathcal{P}$ be a finite PTRS and let $(\mathfrak{I}, \succ)$ be a monotonic, collapsible, barycentric $\Sigma$-algebra, and let $\epsilon > 0$. If (1) holds, then $\mathcal{P}$ is SAST. Here, $\mathbb{E}_{\mathfrak{I}}(\mu) = \bigoplus_{1\le j\le k} p_j \odot \mathfrak{I}(r_j)$ denotes the barycenter of the interpreted right-hand side $\mu = \{^{p_1}r_1, \dots, ^{p_k}r_k\}$ of a rule.

(1) $\quad \forall \ell \to \mu \in \mathcal{P}.\ \mathfrak{I}(\ell) \succ_\epsilon \mathbb{E}_{\mathfrak{I}}(\mu)$.

*Proof.* Similar to the proof of Theorem 3.2.62, we split the proof into two parts. We first show that condition (1) generalizes to arbitrary rewrite steps instead of just rules, and then we show that $\mathcal{P}$ is SAST. *Proof*

**Step 1.** Extend the conditions from rules to rewrite steps

Here, we show that the condition extends to rewrite steps: $\forall s \to_{\mathcal{P}} \mu.\ \mathfrak{I}(s) \succ_\epsilon \mathbb{E}_{\mathfrak{I}}(\mu)$. Let $s \to_{\mathcal{P}} \mu = \{^{p_1}t_1, \dots, ^{p_k}t_k\}$. By definition of the rewrite relation, there exist a rule $\ell \to \{^{p_1}r_1, \dots, ^{p_k}r_k\} \in \mathcal{P}$, a substitution $\sigma$, and a position $\pi$ of $s$ such that $s|_\pi = \ell\sigma$ and $t_j = s[r_j\sigma]_\pi$ for all $1 \le j \le k$.
We perform induction on $\pi$. So in the induction base $\pi = \varepsilon$ we have $s = \ell\sigma$ and $t_j = r_j\sigma$ for all $1 \le j \le k$. Then by (1), we know that $\mathfrak{I}(\ell) \succ_\epsilon \bigoplus_{1\le j\le k} p_j \odot \mathfrak{I}(r_j)$. This inequality holds for all instantiations of the occurring variables. Thus, we obtain

$$\mathfrak{I}(s) \;=\; \mathfrak{I}(\ell\sigma) \;\succ_\epsilon\; \bigoplus_{1\le j\le k} p_j \odot \mathfrak{I}(r_j\sigma) \;=\; \bigoplus_{1\le j\le k} p_j \odot \mathfrak{I}(t_j).$$

In the induction step, we have $\pi = i.\pi'$, $s = f(s_1, \dots, s_i, \dots, s_n)$, $s_i \to_{\mathcal{P}} \{^{p_1}t_{i,1}, \dots, ^{p_k}t_{i,k}\}$, and $t_j = f(s_1, \dots, t_{i,j}, \dots, s_n)$ with $t_{i,j} = s_i[r_j\sigma]_{\pi'}$ for all $1 \le j \le k$. Then by the induction hypothesis we have $\mathfrak{I}(s_i) \succ_\epsilon \bigoplus_{1\le j\le k} p_j \odot \mathfrak{I}(t_{i,j})$. And thus

$$\begin{array}{rcl}
\mathfrak{I}(s) & = & \mathfrak{I}(f(s_1, \dots, s_i, \dots, s_n)) \\
& = & f_{\mathfrak{I}}(\mathfrak{I}(s_1), \dots, \mathfrak{I}(s_i), \dots, \mathfrak{I}(s_n)) \\
& & \quad \downarrow \text{ by closure of } \succ_\epsilon \text{ under } f_{\mathfrak{I}} \text{ and } \mathfrak{I}(s_i) \succ_\epsilon \bigoplus_{1\le j\le k} p_j \odot \mathfrak{I}(t_{i,j}) \\
& \succ_\epsilon & f_{\mathfrak{I}}(\mathfrak{I}(s_1), \dots, \bigoplus_{1\le j\le k} p_j \odot \mathfrak{I}(t_{i,j}), \dots, \mathfrak{I}(s_n)) \\
& & \quad \downarrow \text{ as } f_{\mathfrak{I}} \text{ is componentwise concave} \\
& \succcurlyeq & \bigoplus_{1\le j\le k} p_j \odot f_{\mathfrak{I}}(\mathfrak{I}(s_1), \dots, \mathfrak{I}(t_{i,j}), \dots, \mathfrak{I}(s_n)) \\
& = & \bigoplus_{1\le j\le k} p_j \odot \mathfrak{I}(f(s_1, \dots, t_{i,j}, \dots, s_n)) \\
& = & \bigoplus_{1\le j\le k} p_j \odot \mathfrak{I}(t_j)
\end{array}$$

where the step regarding the concavity is due to Jensen's inequality. As $\succ_\epsilon$ absorbs $\succcurlyeq$ on the right, combining the strict $\succ_\epsilon$-step with the subsequent weak $\succcurlyeq$-step yields $\mathfrak{I}(s) \succ_\epsilon \bigoplus_{1\le j\le k} p_j \odot \mathfrak{I}(t_j)$, as desired.

**Step 2.** Prove that $\mathcal{P}$ is SAST

Let $\mathrm{emb} : A \to \mathbb{R}_{\ge 0}$ be the monotone, concave embedding from the collapsibility of $\mathfrak{I}$, and let $\mathfrak{I}_0$ be the interpretation that behaves like the algebra $\mathfrak{I}$ and additionally maps

all variables to $\mathbf{0} \in A$. We define the *value* $\mathcal{V}al : \mathcal{T} \to \mathbb{R}_{\geq 0}$ by

$$t \mapsto \begin{cases} \mathrm{emb}(\mathfrak{I}_0(t)) + 1, & \text{if } t \notin \mathtt{NF}_{\mathcal{P}}, \\ 0, & \text{if } t \in \mathtt{NF}_{\mathcal{P}}, \end{cases}$$

and the expected value of any $\mu \in \mathrm{FDist}(\mathcal{T})$ w.r.t. $\mathcal{V}al$ as $\mathbb{E}_{\mathcal{V}al}(\mu) = \sum_{p\, t \in \mu} p \cdot \mathcal{V}al(t)$. Note that $\mathfrak{I}_0(t), \mathbb{E}_{\mathfrak{I}_0}(\mu) \in A$ and $\mathcal{V}al(t), \mathbb{E}_{\mathcal{V}al}(\mu) \in \mathbb{R}_{\geq 0}$.
We have to bound the expected derivation height for every term $t \in \mathcal{T}$ by some finite constant. We first show that every rewrite step $s \to_{\mathcal{P}} \mu = \{{}^{p_1}t_1, \ldots, {}^{p_k}t_k\}$ with $s \notin \mathtt{NF}_{\mathcal{P}}$ decreases the value by at least $\epsilon$ in expectation. By subgoal $\mathtt{1}$, we have $\mathfrak{I}(s) \succ_\epsilon \mathbb{E}_{\mathfrak{I}}(\mu)$, and since this holds for all instantiations (e.g., mapping all variables to $\mathbf{0} \in A$), we have $\mathfrak{I}_0(s) \succ_\epsilon \mathbb{E}_{\mathfrak{I}_0}(\mu) = \bigoplus_{1 \leq j \leq k} p_j \odot \mathfrak{I}_0(t_j)$. By the definition of $\succ_\epsilon$ together with concavity of emb, we obtain

$$\begin{array}{rcl} \mathcal{V}al(s) & = & \mathrm{emb}(\mathfrak{I}_0(s)) + 1 \\ & & \quad \downarrow \ \text{as } \mathfrak{I}_0(s) \succ_\epsilon \mathbb{E}_{\mathfrak{I}_0}(\mu) \\ & \geq & \mathrm{emb}(\mathbb{E}_{\mathfrak{I}_0}(\mu)) + \epsilon + 1 \\ & & \quad \downarrow \ \text{as emb is concave} \\ & \geq & \sum_{1 \leq j \leq k} p_j \cdot \mathrm{emb}(\mathfrak{I}_0(t_j)) + \epsilon + 1 \\ & & \quad \downarrow \ \text{as } \mathcal{V}al(t_j) \leq \mathrm{emb}(\mathfrak{I}_0(t_j)) + 1 \text{ and } \sum_{1 \leq j \leq k} p_j = 1 \\ & \geq & \sum_{1 \leq j \leq k} p_j \cdot \mathcal{V}al(t_j) + \epsilon \ = \ \mathbb{E}_{\mathcal{V}al}(\mu) + \epsilon. \end{array}$$

We now use this per-step decrease to show that $\mathrm{edh}_{\to_{\mathcal{P}}}(t) \leq {}^1\!/\!_\epsilon \cdot \mathcal{V}al(t)$.
Let $\mathfrak{T} = (N, E)$ be a $\to_{\mathcal{P}}$-RT starting with $t$, and let $t_v$ denote the term of a node $v \in N$. We have $\mathcal{V}al(t)$ as the initial value at the root. Whenever we perform a rewrite step at a node $v$, we strictly decrease the value by at least $\epsilon \cdot p_v$ in expectation, see Figure 3.10. More precisely, for $n \in \mathbb{N}$, let $L_n \subseteq N$ consist of all nodes of depth $n$ and all leaves of depth less than $n$. For every inner node $v$ with the rewrite step $t_v \to_{\mathcal{P}} \mu_v = \{{}^{p_w/p_v}t_w \mid w \in vE\}$, we have $p_v \cdot \mathcal{V}al(t_v) \geq \sum_{w \in vE} p_w \cdot \mathcal{V}al(t_w) + \epsilon \cdot p_v$. Thus, by induction on $n$, we obtain

$$\sum_{v \in L_n} p_v \cdot \mathcal{V}al(t_v) \ \leq \ \mathcal{V}al(t) - \sum_{v \in N \setminus \mathrm{Leaf}(\mathfrak{T}),\, \mathrm{d}(v) < n} \epsilon \cdot p_v \qquad \text{for all } n \in \mathbb{N}.$$

Since the value is bounded from below by 0, the left-hand side is non-negative, and for $n \to \infty$ we get

$$\mathcal{V}al(t) - \sum_{v \in N \setminus \mathrm{Leaf}(\mathfrak{T})} \epsilon \cdot p_v \geq 0 \iff \sum_{v \in N \setminus \mathrm{Leaf}(\mathfrak{T})} \epsilon \cdot p_v \leq \mathcal{V}al(t). \tag{3.9}$$

Therefore, we get

$$\epsilon \cdot \mathrm{edl}(\mathfrak{T}) \overset{\text{(definition of edl)}}{=} \epsilon \cdot \sum_{v \in N \setminus \mathrm{Leaf}(\mathfrak{T})} p_v = \sum_{v \in N \setminus \mathrm{Leaf}(\mathfrak{T})} \epsilon \cdot p_v \overset{(3.9)}{\leq} \mathcal{V}al(t),$$

and therefore, $\mathrm{edl}(\mathfrak{T}) \leq {}^1\!/\!_\epsilon \cdot \mathcal{V}al(t)$. Since this holds for every $\to_{\mathcal{P}}$-RT starting with $t$, we get $\mathrm{edh}_{\to_{\mathcal{P}}}(t) \leq {}^1\!/\!_\epsilon \cdot \mathcal{V}al(t)$, and thus, $\mathcal{P}$ is SAST. *QED*

As a corollary, we obtain a result via polynomial interpretations. In the polynomial corollaries of this probabilistic setting, we use Pol as an algebra with carrier $\mathbb{R}_{\geq 0}$

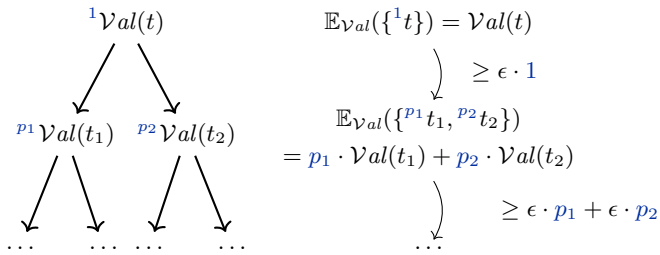



Figure 3.10: Expected decrease of $\mathcal{V}al$ in a $\to_{\mathcal{P}}$-RT.

instead of $\mathbb{N}$ (the carrier must form an $\mathbb{R}_{\geq 0}$-semimodule), i.e., every $f_{\mathrm{Pol}}$ is considered as a function on $\mathbb{R}_{\geq 0}$. Accordingly, the orders $\geq$ and $>$ between polynomials are read pointwise over instantiations from $\mathbb{R}_{\geq 0}$ (like $>_\epsilon$), matching the orders $\succcurlyeq$ and $\succ$ of the algebra lifted over its carrier, and monotonicity refers to these functions on $\mathbb{R}_{\geq 0}$ (for a natural, multilinear Pol that is monotonic on $\mathbb{N}$, every $f_{\mathrm{Pol}}$ is also monotonic on $\mathbb{R}_{\geq 0}$, since the monomial $x_i$ occurs in $f_{\mathrm{Pol}}$ with coefficient $\geq 1$ for every argument position).

Corollary 3.4.34 – Proving SAST Via Polynomial Interpretations, [ADLY20].

*Let $\mathcal{P}$ be a finite PTRS, let* $\mathrm{Pol} : \mathcal{T} \to \mathbb{N}[\mathcal{V}]$ *be a natural, monotonic, and multilinear polynomial interpretation, and let $\epsilon > 0$. If (1) holds, then $\mathcal{P}$ is* SAST.

(1) $\forall \ell \to \mu \in \mathcal{P}.\ \mathrm{Pol}(\ell) >_\epsilon \mathbb{E}_{\mathrm{Pol}}(\mu)$.

*Proof.* Corollary 3.4.34 instantiates Theorem 3.4.33 as follows. Since a multilinear polynomial with non-negative coefficients is affine in each variable with a non-negative constant part, it is concave, so $(\mathrm{Pol}, >)$ is a barycentric $\Sigma$-algebra. We take the identity embedding $\mathrm{emb} = id : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$, which is (a) monotone and (b) concave. The induced $\epsilon$-decrease $\succ_\epsilon$ then coincides with the standard gap order $>_\epsilon$ on $\mathbb{R}_{\geq 0}$, i.e., $a >_\epsilon b$ iff $a \geq b + \epsilon$, lifted over instantiations from the carrier $\mathbb{R}_{\geq 0}$. It is closed under addition, and it is closed under the interpretation functions, because for a monotonic multilinear $f_{\mathrm{Pol}}$, the monomial $x_i$ occurs with coefficient $\geq 1$ for every argument position $i$, so $f_{\mathrm{Pol}}$ preserves the $\epsilon$-gap (see the explanation after Definition 3.4.31). *Proof* *QED*

Example 3.4.35. Reconsider the PTRS $\mathcal{P}_{\mathsf{geo}}$ from Example 3.4.3 with the only rule $\mathsf{geo}(x) \to \{^{1/2}\mathsf{geo}(\mathsf{s}(x)),\ ^{1/2}x\}$. The polynomial interpretation Pol with $\mathsf{s}_{\mathrm{Pol}}(x) = x$, $\mathsf{geo}_{\mathrm{Pol}}(x) = x + 1$, and $\mathsf{0}_{\mathrm{Pol}} = 0$ is natural, monotonic, and multilinear. Moreover, we have $\mathrm{Pol}(\mathsf{geo}(x)) = x{+}1 >_{1/2} x{+}1/2 = 1/2{\cdot}(x{+}1){+}1/2{\cdot}x = 1/2{\cdot}\mathrm{Pol}(\mathsf{geo}(\mathsf{s}(x))){+}1/2{\cdot}\mathrm{Pol}(x)$, i.e., condition (1) holds with $\epsilon = 1/2$. Therefore, we can now use Corollary 3.4.34 to prove $\mathtt{SAST}(\mathcal{P}_{\mathsf{geo}})$.

**Inferring Complexity Bounds via Interpretations** Given a polynomial interpretation that proves SAST of a PTRS $\mathcal{P}$ via Corollary 3.4.34, we can additionally infer upper bounds on the expected derivational complexity and on the expected runtime complexity of $\mathcal{P}$. As shown in the proof of Theorem 3.4.33, the expression $1/\epsilon \cdot (\mathrm{emb}(\mathrm{Pol}_0(t)) + 1)$ of a term $t$ yields an upper bound on the expected derivation height of $t$, i.e., $\mathrm{edh}_{\to_{\mathcal{P}}}(t) \leq 1/\epsilon \cdot (\mathrm{emb}(\mathrm{Pol}_0(t)) + 1)$. Again, $\mathrm{Pol}_0$ denotes the polynomial interpretation that behaves like Pol, but additionally, maps all variables to 0. Since $1/\epsilon$

is a constant for a fixed polynomial interpretation and PTRS, we can reuse the analysis of the size of $\mathrm{Pol}(t)$ from Section 3.2.5, which then yields the following corollary.

Corollary 3.4.36 – Upper Bounds on Expected Complexity Via Polynomial Interpretations.

*Let $\mathcal{P}$ be a finite PTRS, let $\mathrm{Pol} : \mathcal{T} \to \mathbb{N}[\mathcal{V}]$ be a natural, monotonic, and multilinear polynomial interpretation, and let $\epsilon > 0$. If we have $\forall \ell \to \mu \in \mathcal{P}.\ \mathrm{Pol}(\ell) >_\epsilon \mathbb{E}_{\mathrm{Pol}}(\mu)$, then we can infer the following upper bounds on the expected derivational complexity and the expected runtime complexity of $\mathcal{P}$.*

- *If* Pol *is a hard CPI, then $\iota(\mathrm{edc}_\mathcal{P}) \sqsubseteq \mathbf{Pol}_1$ and $\iota(\mathrm{erc}_\mathcal{P}) \sqsubseteq \mathbf{Pol}_1$.*
- *If* Pol *is a CPI, then $\iota(\mathrm{erc}_\mathcal{P}) \sqsubseteq \mathbf{Pol}_a$, where $a$ is the maximum degree of a polynomial $f_{\mathrm{Pol}}$ for some $f \in \mathcal{D}$.*
- *If* Pol *is linear on constructors, then $\iota(\mathrm{erc}_\mathcal{P}) \sqsubseteq \mathbf{Exp}$.*
- *If* Pol *is linear, then $\iota(\mathrm{edc}_\mathcal{P}) \sqsubseteq \mathbf{Exp}$.*
- *If* Pol *is arbitrary, then $\iota(\mathrm{edc}_\mathcal{P}) \sqsubseteq \mathbf{2\text{-}Exp}$ and $\iota(\mathrm{erc}_\mathcal{P}) \sqsubseteq \mathbf{2\text{-}Exp}$.*

Example 3.4.37. Reconsider the PTRS $\mathcal{P}_{\mathsf{geo}}$ from Example 3.4.3 with the only rule $\mathsf{geo}(x) \to \{^{1/2}\mathsf{geo}(\mathsf{s}(x)),\ ^{1/2}x\}$ and the corresponding `SAST` proof using the polynomial interpretation Pol with $\mathsf{s}_{\mathrm{Pol}}(x) = x$, $\mathsf{geo}_{\mathrm{Pol}}(x) = x+1$, and $\mathsf{0}_{\mathrm{Pol}} = 0$ from Example 3.4.35. Pol is a hard CPI, since we only use linear polynomials with coefficients in $\{0, 1\}$. By Corollary 3.4.36 we get $\iota(\mathrm{erc}_{\mathcal{P}_{\mathsf{geo}}}) \sqsubseteq \mathbf{Pol}_1$ and $\iota(\mathrm{edc}_{\mathcal{P}_{\mathsf{geo}}}) \sqsubseteq \mathbf{Pol}_1$. Note that while $\iota(\mathrm{edc}_{\mathcal{P}_{\mathsf{geo}}})$ is actually equal to $\mathbf{Pol}_1$, we have $\iota(\mathrm{erc}_{\mathcal{P}_{\mathsf{geo}}}) = \mathbf{Pol}_0$. However, Corollary 3.4.36 cannot be used to prove constant runtime. We show in Chapter 8 another technique via *dependency pairs* that is able to show constant runtime complexity.

**Proving `AST` Via Interpretations** Finally, we introduce a novel approach to analyzing `AST` for a given PTRS via barycentric algebras. The core idea of the following theorem is based on [MMKK18], but it extends their approach from `pGCL` to term rewriting. While Theorem 3.4.33 requires a strict decrease in the expected value of each rule, for `AST` we only require a weak decrease of the expected value. In addition, we also have to require that at least one term in the support of the right-hand side must become strictly smaller. However, in contrast to [MMKK18], PTRSs can only deal with constant probabilities, since all variables stand for terms, not for numbers.

Theorem 3.4.38 – Proving `AST` Via Interpretations.

Let $\mathcal{P}$ be a finite PTRS and let $(\mathfrak{I}, \succ)$ be a monotonic, collapsible, barycentric $\Sigma$-algebra, and let $\epsilon > 0$. If (1) and (2) hold, then $\mathcal{P}$ is `AST`.

(1) $\quad \forall \ell \to \mu = \{^{p_1}r_1, \ldots, ^{p_k}r_k\} \in \mathcal{P}.\ \exists 1 \le j \le k.\ \mathfrak{I}(\ell) \succ_\epsilon \mathfrak{I}(r_j)$.

(2) $\quad \forall \ell \to \mu = \{^{p_1}r_1, \ldots, ^{p_k}r_k\} \in \mathcal{P}.\ \mathfrak{I}(\ell) \succcurlyeq \mathbb{E}_\mathfrak{I}(\mu) = \bigoplus_{1 \le j \le k} p_j \odot \mathfrak{I}(r_j)$.

*Proof.* The main idea for this proof comes from [MMKK18] and consists of the following five steps. *Proof*

1. We extend the conditions to rewrite steps instead of just rules
2. We consider adjusted sequences $\vec{\mu}^{\le N}$ for any $N \in \mathbb{N}$
3. We prove that $|\vec{\mu}^{\le N}| \ge p_{\min}^N > 0$ for any $N \in \mathbb{N}$

4. We prove that $|\vec{\mu}^{\leq N}| = 1$ for any $N \in \mathbb{N}$
5. Finally, we prove that $|\vec{\mu}| = 1$

Here, $p_{\min} > 0$ is the minimal probability that occurs in the rules of $\mathcal{P}$. As $\mathcal{P}$ has only finitely many rules and all occurring multi-distributions have finite support, this minimum is well defined. We have $p_{\min}^N = \underbrace{p_{\min} \cdot \dots \cdot p_{\min}}_{N \text{ times}}$ in step 3.

**Step 1.** Extend the conditions from rules to rewrite steps

We first show that the conditions (1) and (2) generalize to arbitrary rewrite steps instead of just rules, i.e., we want to show the following two conditions.

(a) $\forall s \to_{\mathcal{P}} \mu = \{{}^{p_1}t_1, \dots, {}^{p_k}t_k\}.\ \exists 1 \leq j \leq k.\ \mathfrak{I}(s) \succ_\epsilon \mathfrak{I}(t_j)$,

(b) $\forall s \to_{\mathcal{P}} \mu = \{{}^{p_1}t_1, \dots, {}^{p_k}t_k\}.\ \mathfrak{I}(s) \succcurlyeq \mathbb{E}_{\mathfrak{I}}(\mu) = \bigoplus_{1 \leq j \leq k} p_j \odot \mathfrak{I}(t_j)$.

Let $s \to_{\mathcal{P}} \mu = \{{}^{p_1}t_1, \dots, {}^{p_k}t_k\}$. By definition of the rewrite relation, there exist a rule $\ell \to \{{}^{p_1}r_1, \dots, {}^{p_k}r_k\} \in \mathcal{P}$, a substitution $\sigma$, and a position $\pi$ of $s$ such that $s|_\pi = \ell\sigma$ and $t_j = s[r_j\sigma]_\pi$ for all $1 \leq j \leq k$.

(a) We perform induction on $\pi$. So in the induction base, let $\pi = \varepsilon$. Hence, we have $s = \ell\sigma$ and $t_j = r_j\sigma$ for all $1 \leq j \leq k$. By requirement (1), there is a $1 \leq j \leq k$ with $\mathfrak{I}(\ell) \succ_\epsilon \mathfrak{I}(r_j)$. As these inequalities hold for all instantiations of the occurring variables, for $t_j = r_j\sigma$, we have $\mathfrak{I}(s) = \mathfrak{I}(\ell\sigma) \succ_\epsilon \mathfrak{I}(r_j\sigma) = \mathfrak{I}(t_j)$.

In the induction step, we have $\pi = i.\pi'$, $s = f(s_1, \dots, s_i, \dots, s_n)$, $s_i \to_{\mathcal{P}} \{{}^{p_1}t_{i,1}, \dots, {}^{p_k}t_{i,k}\}$, and $t_j = f(s_1, \dots, t_{i,j}, \dots, s_n)$ with $t_{i,j} = s_i[r_j\sigma]_{\pi'}$ for all $1 \leq j \leq k$. Then by the induction hypothesis there is a $1 \leq j \leq k$ with $\mathfrak{I}(s_i) \succ_\epsilon \mathfrak{I}(t_{i,j})$. For $t_j = f(s_1, \dots, t_{i,j}, \dots, s_n)$ we obtain

$$\begin{aligned}
\mathfrak{I}(s) &= \mathfrak{I}(f(s_1, \dots, s_i, \dots, s_n)) \\
&= f_{\mathfrak{I}}(\mathfrak{I}(s_1), \dots, \mathfrak{I}(s_i), \dots, \mathfrak{I}(s_n)) \\
&\qquad \downarrow \text{ by closure of } \succ_\epsilon \text{ under } f_{\mathfrak{I}} \text{ and } \mathfrak{I}(s_i) \succ_\epsilon \mathfrak{I}(t_{i,j}) \\
&\succ_\epsilon f_{\mathfrak{I}}(\mathfrak{I}(s_1), \dots, \mathfrak{I}(t_{i,j}), \dots, \mathfrak{I}(s_n)) \\
&= \mathfrak{I}(f(s_1, \dots, t_{i,j}, \dots, s_n)) \\
&= \mathfrak{I}(t_j).
\end{aligned}$$

(b) We again perform induction on $\pi$. So in the induction base $\pi = \varepsilon$ we again have $s = \ell\sigma$ and $t_j = r_j\sigma$ for all $1 \leq j \leq k$. Then by (2), we know that $\mathfrak{I}(\ell) \succcurlyeq \bigoplus_{1 \leq j \leq k} p_j \odot \mathfrak{I}(r_j)$. This inequality holds for all instantiations of the occurring variables. Thus, we obtain

$$\mathfrak{I}(s) = \mathfrak{I}(\ell\sigma) \succcurlyeq \bigoplus_{1 \leq j \leq k} p_j \odot \mathfrak{I}(r_j\sigma) = \bigoplus_{1 \leq j \leq k} p_j \odot \mathfrak{I}(t_j).$$

In the induction step, we have $\pi = i.\pi'$, $s = f(s_1, \dots, s_i, \dots, s_n)$, $s_i \to_{\mathcal{P}} \{{}^{p_1}t_{i,1}, \dots, {}^{p_k}t_{i,k}\}$, and $t_j = f(s_1, \dots, t_{i,j}, \dots, s_n)$ with $t_{i,j} = s_i[r_j\sigma]_{\pi'}$ for all $1 \leq j \leq k$. Then by the induction hypothesis we have $\mathfrak{I}(s_i) \succcurlyeq \bigoplus_{1 \leq j \leq k} p_j \odot \mathfrak{I}(t_{i,j})$.

And thus

$$\begin{aligned}
\mathfrak{I}(s) &= \mathfrak{I}(f(s_1,\dots,s_i,\dots,s_n)) \\
&= f_{\mathfrak{I}}(\mathfrak{I}(s_1),\dots,\mathfrak{I}(s_i),\dots,\mathfrak{I}(s_n)) \\
&\quad \downarrow \text{ by monotonicity of } f_{\mathfrak{I}} \text{ and } \mathfrak{I}(s_i) \succcurlyeq \textstyle\bigoplus_{1\le j\le k} p_j \odot \mathfrak{I}(t_{i,j}) \\
&\succcurlyeq f_{\mathfrak{I}}(\mathfrak{I}(s_1),\dots,\textstyle\bigoplus_{1\le j\le k} p_j \odot \mathfrak{I}(t_{i,j}),\dots,\mathfrak{I}(s_n)) \\
&\quad \downarrow \text{ as } f_{\mathfrak{I}} \text{ is componentwise concave} \\
&\succcurlyeq \textstyle\bigoplus_{1\le j\le k} p_j \odot f_{\mathfrak{I}}(\mathfrak{I}(s_1),\dots,\mathfrak{I}(t_{i,j}),\dots,\mathfrak{I}(s_n)) \\
&= \textstyle\bigoplus_{1\le j\le k} p_j \odot \mathfrak{I}(f(s_1,\dots,t_{i,j},\dots,s_n)) \\
&= \textstyle\bigoplus_{1\le j\le k} p_j \odot \mathfrak{I}(t_j)
\end{aligned}$$

where the step regarding the concavity is due to Jensen's inequality.

**Step 2.** We create an adjusted sequence $\vec{\mu}^{\le N}$ for any $N \in \mathbb{N}$

As in the proof of Theorem 3.4.33, let $\mathrm{emb} : A \to \mathbb{R}_{\ge 0}$ be the monotone, concave embedding from the collapsibility of $\mathfrak{I}$. Let $\mathrm{emb}' = {}^1\!/\!_\epsilon \cdot \mathrm{emb}$, which is again monotone and concave, so that $a_1 \succ_\epsilon a_2$ implies $\mathrm{emb}'(a_1) \ge \mathrm{emb}'(a_2) + 1$. Moreover, let $\mathfrak{I}_0$ be the interpretation that behaves like the algebra $\mathfrak{I}$ and additionally maps all variables to $\mathbf{0} \in A$. We define the *value* $\mathcal{V}al : \mathcal{T} \to \mathbb{R}_{\ge 0}$ by

$$t \mapsto \begin{cases} \mathrm{emb}'(\mathfrak{I}_0(t)) + 1, & \text{if } t \notin \mathtt{NF}_{\mathcal{P}}, \\ 0, & \text{if } t \in \mathtt{NF}_{\mathcal{P}}, \end{cases}$$

and the expected value of any $\mu \in \mathrm{FDist}(\mathcal{T})$ w.r.t. $\mathcal{V}al$ as $\mathbb{E}_{\mathcal{V}al}(\mu) = \sum_{p_{t\in\mu}} p \cdot \mathcal{V}al(t)$. Note that $\mathfrak{I}_0(t), \mathbb{E}_{\mathfrak{I}_0}(\mu) \in A$ and $\mathcal{V}al(t), \mathbb{E}_{\mathcal{V}al}(\mu) \in \mathbb{R}_{\ge 0}$. Additionally, using $t \notin \mathtt{NF}_{\mathcal{P}}$ (so $\mathcal{V}al(t) = \mathrm{emb}'(\mathfrak{I}_0(t)) + 1$) and $\mathcal{V}al(s) \le \mathrm{emb}'(\mathfrak{I}_0(s)) + 1$ with $\mathrm{emb}' \ge 0$: $\mathfrak{I}_0(t) \succ_\epsilon \mathfrak{I}_0(s)$ implies $\mathcal{V}al(t) \ge \mathcal{V}al(s) + 1$ (by the gap of $\succ_\epsilon$), and $\mathfrak{I}_0(t) \succcurlyeq \mathfrak{I}_0(s)$ implies $\mathcal{V}al(t) \ge \mathcal{V}al(s)$ (by monotonicity of $\mathrm{emb}'$), for all $t \notin \mathtt{NF}_{\mathcal{P}}$ and $s \in \mathcal{T}$.
For any $\rightrightarrows_{\mathcal{P}}$-rewrite sequence $\vec{\mu} = (\mu_k)_{k\in\mathbb{N}}$ and any $N \in \mathbb{N}$, we now consider a modified sequence $(\mu_k^{\le N})_{k\in\mathbb{N}}$, where we inductively replace each $\mu_i$ by $\mu_i^{\le N}$ starting from $\mu_0$. To obtain $\mu_i^{\le N}$, we replace ${}^p t \in \mu_i$ by ${}^p\top$ if $\mathcal{V}al(t) \ge N + 1$. Otherwise, we keep ${}^p t$. Here, $\top$ is a newly introduced symbol with $\mathcal{V}al(\top) = N + 1$. Note that normal forms are never replaced, as $\mathcal{V}al(t) = 0$ for all $t \in \mathtt{NF}_{\mathcal{P}}$. If at least one term is replaced in a $\mu_i$, we replace $(\mu_k)_{k>i}$ by a new sequence before proceeding. Starting from $\mu_i^{\le N}$, we use the same rewrite steps as the original $\rightrightarrows_{\mathcal{P}}$-rewrite sequence, except for the new normal forms $\top$, which stay the same. Of course, the result is not a valid $\rightrightarrows_{\mathcal{P}}$-rewrite sequence anymore.
Our goal is to prove that every $\rightrightarrows_{\mathcal{P}}$-rewrite sequence $\vec{\mu} = (\mu_k)_{k\in\mathbb{N}}$ converges with probability one, i.e., that $|\vec{\mu}| = 1$. The proof proceeds similarly to [MMKK18]. First of all, we prove that $|\vec{\mu}^{\le N}| = 1$ for any $N \in \mathbb{N}$, i.e., we prove that this new kind of sequence converges with probability one. Note that we also consider $\top \in \mathtt{NF}_{\mathcal{P}}$. However, this does not yet prove that $|\vec{\mu}| = 1$, which we show in a second step.

**Step 3.** We prove that $|\vec{\mu}^{\le N}| \ge p_{\min}^N$ for any $N \in \mathbb{N}$

Due to (a), every rewrite step $s \to_{\mathcal{P}} \nu$ of a term $s$ that is not in normal form has some $t \in \mathrm{Supp}(\nu)$ with $\nu(t) \ge p_{\min}$ and $\mathfrak{I}(s) \succ_\epsilon \mathfrak{I}(t)$ (and thus, after instantiating all variables with $\mathbf{0}$, also $\mathcal{V}al(s) \ge \mathcal{V}al(t) + 1$). So for any $N \in \mathbb{N}$ and any $\rightrightarrows_{\mathcal{P}}$-sequence

$\vec{\mu} = (\mu_n)_{n\in\mathbb{N}}$, we have $|\mu_N^{\leq N}|_{\mathcal{P}} \geq p_{\min}^N$. The reason is that for every term in $\mathrm{Supp}(\mu_i^{\leq N})$ that is not in normal form, there is always a term in $\mathrm{Supp}(\mu_{i+1}^{\leq N})$ which decreases the value of $\mathcal{V}al$ by at least 1. The values along such a decreasing branch stay below $N+1$, so these terms are never replaced by $\top$, and since every term that is not in normal form has $\mathcal{V}al \geq 1$, a normal form is reached within $N$ steps. So for all terms in $\mathrm{Supp}(\mu_0^{\leq N})$ that are not in normal form, we reach a normal form in at most $N$ steps with at least probability $p_{\min}^N$.
Here, collapsibility ensures that $p_{\min}^N > 0$ is a lower bound on the termination probability. If one wants to adapt this theorem to more complex algebras (that are not collapsible), then one needs to find different ways to obtain a constant lower bound $> 0$ for every $N \in \mathbb{N}$.

**Step 4.** We prove that $|\vec{\mu}^{\leq N}| = 1$ for any $N \in \mathbb{N}$

We utilize the constant lower bound on the termination probability for all possible rewrite sequences $\vec{\mu}^{\leq N}$ to prove that all such rewrite sequences terminate with probability 1. This is also referred to as the zero-one law of probabilistic programs [MM05].
Note that $|\mu_k^{\leq N}|_{\mathcal{P}}$ is bounded and weakly monotonically increasing with increasing $k \in \mathbb{N}$, so that $|\vec{\mu}^{\leq N}|$ must exist and we have $|\vec{\mu}^{\leq N}| \geq |\mu_N^{\leq N}|_{\mathcal{P}}$. Remember that we write $|\vec{\mu}^{\leq N}|$ for $\lim_{k\to\infty} |\mu_k^{\leq N}|_{\mathcal{P}}$. Hence, for any $N \in \mathbb{N}$, we have

$$p_N^\star = \inf_{(\mu_k)_{k\in\mathbb{N}} \text{ is a } \rightrightarrows_{\mathcal{P}}\text{-rewrite sequence}} (|\vec{\mu}^{\leq N}|) \geq p_{\min}^N > 0. \tag{3.10}$$

We now prove by contradiction that this is enough to ensure $p_N^\star = 1$. So assume that $p_N^\star < 1$. Then we define $\epsilon' = \frac{p_N^\star \cdot (1-p_N^\star)}{2} > 0$. By definition of the infimum, $p_N^\star + \epsilon'$ is not a lower bound of $|\vec{\mu}^{\leq N}|$ for all $\rightrightarrows_{\mathcal{P}}$-rewrite sequences. Hence, there must exist a $\rightrightarrows_{\mathcal{P}}$-rewrite sequence $(\mu_k)_{k\in\mathbb{N}}$ such that

$$p_N^\star \leq |\vec{\mu}^{\leq N}| < p_N^\star + \epsilon'. \tag{3.11}$$

By the monotonicity of $|\cdot|$ w.r.t. $\rightrightarrows_{\mathcal{P}}$ steps, there must exist a natural number $m^* \in \mathbb{N}$ such that

$$|\mu_{m^*}^{\leq N}|_{\mathcal{P}} > \tfrac{p_N^\star}{2}. \tag{3.12}$$

Then we have

$$|\vec{\mu}^{\leq N}| = |\mu_{m^*}^{\leq N}|_{\mathcal{P}} + \sum_{p\, t\in\mu_{m^*}^{\leq N}, t\notin \mathtt{NF}_{\mathcal{P}}} p \cdot |\{{}^1 t\}^{\leq N}|, \tag{3.13}$$

where $\{{}^1 t\}_k^{\leq N}$ denotes the sequence starting with $\{{}^1 t\}$, where the same rules are applied as for $t$ in $(\mu_k^{\leq N})_{k\geq m^*}$.
Furthermore, we have

$$\sum_{p\, t\in\mu_{m^*}^{\leq N}, t\notin \mathtt{NF}_{\mathcal{P}}} p = 1 - |\mu_{m^*}^{\leq N}|_{\mathcal{P}}. \tag{3.14}$$

So we obtain

$$
\begin{aligned}
& p_N^\star + \epsilon' \\
& \quad \downarrow \quad \text{(by (3.11))} \\
> & \; |\vec{\mu}^{\leq N}| \\
& \quad \downarrow \quad \text{(by (3.13) and (3.10))} \\
= & \; |\mu_{m^*}^{\leq N}|_{\mathcal{P}} + \sum_{p_t \in \mu_{m^*}^{\leq N}, t \notin \mathtt{NF}_{\mathcal{P}}} p \cdot \underbrace{\lim_{k\to\infty} |\{^1 t\}_k^{\leq N}|_{\mathcal{P}}}_{\geq p_N^\star} \\
\geq & \; |\mu_{m^*}^{\leq N}|_{\mathcal{P}} + \sum_{p_t \in \mu_{m^*}^{\leq N}, t \notin \mathtt{NF}_{\mathcal{P}}} p \cdot p_N^\star \\
= & \; |\mu_{m^*}^{\leq N}|_{\mathcal{P}} + p_N^\star \cdot \sum_{p_t \in \mu_{m^*}^{\leq N}, t \notin \mathtt{NF}_{\mathcal{P}}} p \\
& \quad \downarrow \quad \text{(by (3.14))} \\
= & \; |\mu_{m^*}^{\leq N}|_{\mathcal{P}} + p_N^\star \cdot (1 - |\mu_{m^*}^{\leq N}|_{\mathcal{P}}) \\
= & \; p_N^\star + |\mu_{m^*}^{\leq N}|_{\mathcal{P}} - p_N^\star \cdot |\mu_{m^*}^{\leq N}|_{\mathcal{P}} \\
= & \; p_N^\star + |\mu_{m^*}^{\leq N}|_{\mathcal{P}} \cdot (1 - p_N^\star) \\
& \quad \downarrow \quad \text{(by (3.12))} \\
> & \; p_N^\star + (1 - p_N^\star) \cdot \tfrac{p_N^\star}{2} \\
= & \; p_N^\star + \epsilon', \quad \lightning
\end{aligned}
$$

a contradiction. So $p_N^\star = 1$. In particular, this means that for every $N \in \mathbb{N}$ and every $\rightrightarrows_{\mathcal{P}}$-rewrite sequence $(\mu_k)_{k\in\mathbb{N}}$ we have

$$|\vec{\mu}^{\leq N}| = \lim_{k\to\infty} |\mu_k^{\leq N}|_{\mathcal{P}} = 1. \tag{3.15}$$

**Step 5.** We prove that $|\vec{\mu}| = 1$

Finally, we have to prove that $|\vec{\mu}| = \lim_{k\to\infty} |\mu_k|_{\mathcal{P}} = 1$ holds as well, i.e., that every $\rightrightarrows_{\mathcal{P}}$-rewrite sequence converges with probability one. By (b), every rewrite step $t \to_{\mathcal{P}} \nu_t$ of a term $t \notin \mathtt{NF}_{\mathcal{P}}$ satisfies $\mathfrak{I}_0(t) \succcurlyeq \mathbb{E}_{\mathfrak{I}_0}(\nu_t)$, so by monotonicity and concavity of $\mathrm{emb}'$ we obtain

$$\mathcal{V}al(t) = \mathrm{emb}'(\mathfrak{I}_0(t)) + 1 \geq \mathrm{emb}'(\mathbb{E}_{\mathfrak{I}_0}(\nu_t)) + 1 \geq \mathbb{E}(\mathrm{emb}' \circ \mathfrak{I}_0(\nu_t)) + 1 \geq \mathbb{E}_{\mathcal{V}al}(\nu_t),$$

where the last step uses $\mathcal{V}al(u) \leq \mathrm{emb}'(\mathfrak{I}_0(u)) + 1$. Summing these inequalities over a step $\mu \rightrightarrows_{\mathcal{P}} \mu'$ yields $\mathbb{E}_{\mathcal{V}al}(\mu) \geq \mathbb{E}_{\mathcal{V}al}(\mu')$ where $\mathbb{E}_{\mathcal{V}al}(\mu) = \sum_{p_t \in \mu} p \cdot \mathcal{V}al(t)$. While our modified sequence $\vec{\mu}^{\leq N} = (\mu_k^{\leq N})_{k\in\mathbb{N}}$ is not a rewrite sequence anymore, we still have $\mathbb{E}_{\mathcal{V}al}(\mu_i^{\leq N}) \geq \mathbb{E}_{\mathcal{V}al}(\mu_{i+1}^{\leq N})$ because either this is a valid $\rightrightarrows_{\mathcal{P}}$-step or a term with a larger value is replaced by a smaller or equal one, i.e., $\top$. Additionally, $\mathbb{E}_{\mathcal{V}al}(\mu_i^{\leq N})$ is bounded from below by 0, so that $\lim_{k\to\infty}(\mathbb{E}_{\mathcal{V}al}(\mu_k^{\leq N}))$ exists and we have

$$\mathbb{E}_{\mathcal{V}al}(\mu_0^{\leq N}) \geq \lim_{k\to\infty} (\mathbb{E}_{\mathcal{V}al}(\mu_k^{\leq N})). \tag{3.16}$$

Now we fix $N \in \mathbb{N}$ and a $\rightrightarrows_{\mathcal{P}}$-rewrite sequence $(\mu_k)_{k\in\mathbb{N}}$, and obtain the corresponding transformed sequence $(\mu_k^{\leq N})_{k\in\mathbb{N}}$. Note that by (3.15) we have $|\vec{\mu}^{\leq N}| = 1 = q_N + (1 - q_N)$, where $q_N = \lim_{k\to\infty} \sum_{p\, t\in\mu_k^{\leq N}, t\in \mathtt{NF}_{\mathcal{P}}, t\neq\top} p$. So what is $\lim_{k\to\infty}(\mathbb{E}_{\mathcal{V}al}(\mu_k^{\leq N}))$? The probabilities of zero entries (i.e., terms $t \in \mathtt{NF}_{\mathcal{P}} \setminus \{\top\}$ where $\mathcal{V}al(t) = 0$) add up to $q_N$, while the entries $\top$ with value $\mathcal{V}al(t) \geq N+1$ add up to probability $1 - q_N$. So $\lim_{k\to\infty}(\mathbb{E}_{\mathcal{V}al}(\mu_k^{\leq N}))$ has at least the value $q_N \cdot 0 + (1 - q_N)\cdot(N+1) = (1 - q_N)\cdot(N+1)$. Thus,

$$\mathbb{E}_{\mathcal{V}al}(\mu_0) \geq \mathbb{E}_{\mathcal{V}al}(\mu_0^{\leq N}) \geq \lim_{k\to\infty}(\mathbb{E}_{\mathcal{V}al}(\mu_k^{\leq N})) \geq (1 - q_N)\cdot(N+1),$$

which implies $q_N \geq 1 - \frac{\mathbb{E}_{\mathcal{V}al}(\mu_0)}{N+1}$.
Note that $q_N$ is weakly monotonically increasing and bounded from above by 1 for $N \to \infty$. Hence, $q = \lim_{N\to\infty} q_N$ exists and $1 \geq q \geq \lim_{N\to\infty}(1 - \frac{\mathbb{E}_{\mathcal{V}al}(\mu_0)}{N+1}) = 1$, i.e., $q = 1$. Moreover, we have $q_N \leq |\vec{\mu}|$ for every $N \in \mathbb{N}$, since every true normal form reached in $\vec{\mu}^{\leq N}$ lies on a branch without any cut and is thus also reached in $\vec{\mu}$. Hence, we obtain $|\vec{\mu}| = \lim_{k\to\infty} |\mu_k|_{\mathcal{P}} \geq \lim_{N\to\infty} q_N = q = 1$, i.e., $|\vec{\mu}| = 1$. *QED*

If we fix the algebra to be a monotonic, multilinear polynomial interpretation, then we obtain the following corollary.

Corollary 3.4.39 – Proving AST Via Polynomial Interpretations.

*Let $\mathcal{P}$ be a finite PTRS, let $\mathrm{Pol} : \mathcal{T} \to \mathbb{N}[\mathcal{V}]$ be a natural, monotonic, and multilinear polynomial interpretation, and let $\epsilon > 0$. If (1) and (2) hold, then $\mathcal{P}$ is* AST.

(1) $\forall \ell \to \mu = \{{}^{p_1}r_1, \ldots, {}^{p_k}r_k\} \in \mathcal{P}.\ \exists 1 \leq j \leq k.\ \mathrm{Pol}(\ell) >_\epsilon \mathrm{Pol}(r_j)$.

(2) $\forall \ell \to \mu = \{{}^{p_1}r_1, \ldots, {}^{p_k}r_k\} \in \mathcal{P}.\ \mathrm{Pol}(\ell) \geq \mathbb{E}_{\mathrm{Pol}}(\mu) = \sum_{1\leq j\leq k} p_j \cdot \mathrm{Pol}(r_j)$.

*Proof.* Corollary 3.4.39 instantiates Theorem 3.4.38 in the same way as in the proof of Corollary 3.4.34: $(\mathrm{Pol}, >)$ is a monotonic, collapsible, barycentric $\Sigma$-algebra with the identity embedding $\mathrm{emb} = id$, whose $\epsilon$-decrease $\succ_\epsilon$ coincides with the standard gap order $>_\epsilon$. *Proof* *QED*

Example 3.4.40. Consider the PTRS $\mathcal{P}_{\mathsf{rw}}$ from Example 3.4.2 with the only rule $\mathsf{g}(x) \to \{{}^{1/2}x, {}^{1/2}\mathsf{g}(\mathsf{b}(\mathsf{g}(x)))\}$. Since $\mathcal{P}_{\mathsf{rw}}$ is not SAST, there exists no monotonic, multilinear polynomial interpretation that satisfies the condition of Corollary 3.4.34 (we formally prove $\neg\mathtt{SAST}(\mathcal{P}_{\mathsf{rw}})$ in Chapter 9). However, a polynomial interpretation that satisfies (1) (with $\epsilon = 1$) and (2) for every rule of $\mathcal{P}_{\mathsf{rw}}$ maps $\mathsf{g}_{\mathrm{Pol}}(x) = x + 1$ and $\mathsf{b}_{\mathrm{Pol}}(x) = x$. Hence, $\mathcal{P}_{\mathsf{rw}}$ is AST by Corollary 3.4.39.

Example 3.4.41. Consider the signature $\Sigma = \{\mathsf{0}, \mathsf{s}, \mathsf{len}, \mathsf{cons}, \mathsf{nil}\}$, a variable set with $\{x, y\} \subseteq \mathcal{V}$, and the following PTRS $\mathcal{P}_{\mathsf{len}}$ that is a probabilistic variant of the TRS from Example 3.2.31:

$$\mathsf{len}(\mathsf{nil}) \to \{{}^{1/2}\mathsf{len}(\mathsf{nil}), {}^{1/2}\mathsf{0}\} \tag{3.17}$$

$$\mathsf{len}(\mathsf{cons}(x, y)) \to \{{}^{1/2}\mathsf{len}(\mathsf{cons}(x, y)), {}^{1/2}\mathsf{s}(\mathsf{len}(y))\} \tag{3.18}$$

Here, we have a chance of $1/2$ to do the actual rewrite step and a chance of $1/2$ to do nothing.

We can prove AST using the same polynomial interpretation as in Example 3.2.66 that maps $\mathsf{len}_{\mathrm{Pol}}(x) = x+1$, $\mathsf{s}_{\mathrm{Pol}}(x) = x$, $\mathsf{cons}_{\mathrm{Pol}}(x, y) = x+y+1$, and both $\mathsf{nil}_{\mathrm{Pol}} = \mathsf{0}_{\mathrm{Pol}} = 0$.

This interpretation is natural, monotonic, and multilinear, and (1) (with $\epsilon = 1$) and (2) hold for both rules, so $\mathcal{P}_{\mathsf{len}}$ is AST by Corollary 3.4.39.

Finally, we compare the techniques presented in this chapter to the ones presented in [ADLY20, KG23] for PTRSs, and the technique from Theorem 3.2.62 for TRSs.

Remark 3.4.42 – Relation to Theorem 3.2.62.

For a non-probabilistic PTRS (where all rules have the form $\ell \to \{^1 r\}$), the conditions of Theorem 3.4.33 and Theorem 3.4.38 both boil down to the single constraint $\mathfrak{I}(\ell) \succ_\epsilon \mathfrak{I}(r)$. Then $(\mathfrak{I}, \succ_\epsilon)$ is itself a monotonic, well-founded ordered $\Sigma$-algebra ($\succ_\epsilon$ is transitive, closed under contexts, and every $\succ_\epsilon$-chain starting in $a_0$ has length at most ${}^{\mathrm{emb}(a_0)}/_\epsilon$), so every such proof is also a proof via the classical Theorem 3.2.62. The converse does not hold. The classical theorem also admits well-founded orders with unbounded descent from a single element (e.g., the lexicographic order on $\mathbb{N}^2$), which cannot be collapsible, as the embedding bounds the length of every descent. This restriction is inherent to our proofs, which require a uniform decrease instead of just well-foundedness.

Remark 3.4.43 – Relation to [KG23, KG26].

Theorem 3.4.38 and Corollary 3.4.39 correspond to the results of [KG23, KG26], where similar criteria are defined directly for natural polynomial interpretations, with all orders read over instantiations from $\mathbb{N}$. There, no explicit $\epsilon$ is needed for element-wise strict decreases, since a strict decrease between natural values is a decrease by at least 1. One may generalize both formulations by lifting the orders on $A[\mathcal{V}]$ only over instantiations from a subset $B \subseteq A$ that contains $\mathbf{0}$ and is closed under all interpretation functions $f_{\mathfrak{I}}$: the proofs instantiate the conditions only at interpretations of terms and at $\mathbf{0}$, which always remain in such a set $B$. For natural polynomial interpretations with $B = \mathbb{N}$, one then obtains the criteria of these works with $\epsilon = 1$, while $B = A$ yields the formulation used here. Moreover, the $\mathbb{N}$-collapsibility of [KG26], which requires a concave embedding into $\mathbb{N}$, is replaced here by the real-valued embedding emb together with the explicit $\epsilon$-decrease $\succ_\epsilon$ from Definition 3.4.31.

Remark 3.4.44 – Relation to [ADLY20].

Theorem 3.4.33 and Corollary 3.4.34 correspond to the results of [ADLY20], who also prove SAST via barycentric algebras. However, we require the carrier to form an $\mathbb{R}_{\geq 0}$-semimodule with a compatible order instead of only requiring the existence of a barycentric operation $\mathbb{E}$, since we need this additional structure for the $\sharp$-sums of the reduction pair processors in Chapters 7 and 8.

SECTION 3.5

## Related Work & Open Problems

Term rewrite systems and their termination have been studied for decades, with foundational work already appearing in the early 1970s and 1980s. The classical termination literature developed several ranking functions, i.e., ordering-based proof methods, for termination of TRSs. To soundly prove termination, an order (an irreflexive and transitive binary relation) on terms has to be well founded, closed

under substitutions, and closed under contexts. An order that satisfies those three requirements is called a *reduction order*. If all rules of a given TRS $\mathcal{R}$ are strictly decreasing w.r.t. a reduction order, then $\mathcal{R}$ is terminating.

The first and simplest reduction order is the *embedding order* that compares terms structurally [Der82]. To prove termination of a broader class one has to extend the embedding order to *simplification orders* that are reduction orders containing the embedding order [Der87]. Examples of simplification orders are the *Knuth-Bendix order* (KBO), the *lexicographic path order* (LPO), the *recursive path order* (RPO), and the *weighted path order* (WPO). The Knuth-Bendix order compares terms by a precedence on function symbols and an additional weight for each function symbol [KB70]. The lexicographic path order extends the precedence on function symbols by a lexicographic comparison of arguments [KL80]. Instead of lexicographic comparisons, we can also extend orders by an additional order on multisets of terms, leading to the recursive path order [Der82]. The weighted path order refines the previously mentioned simplification orders by extending them with an additional $\Sigma$-algebra, which increases the class of TRSs that can be handled automatically [Yam14, YKS15].

Instead of simplification orders, we focus solely on ranking functions based on $\Sigma$-algebras in this thesis, as we have to be able to define an *expected value* of a distribution over terms in the probabilistic setting. Examples of $\Sigma$-algebras used for termination analysis include *polynomial interpretations*, *matrix interpretations*, and *tuple interpretations*. Polynomial interpretations as described in Definition 3.2.63 assign polynomials over some universe like $\mathbb{N}$ [MN70, Lan79, Gie95] or $\mathbb{R}$ [Luc05, Luc07] to terms. Matrix interpretations generalize polynomial interpretations by interpreting terms as vectors or even matrices [EWZ06, HW06, EWZ08, KW08, ALNM09, CGP10, Luc10, NM12, Wal15]. The tuple interpretation method subsumes many of these variants. Tuple interpretations were recently developed for higher-order rewriting [KV21] and for first-order rewriting [Yam22b].

It is undecidable to check whether there exists a simplification order that proves termination [MG95] or even just a linear polynomial interpretation that proves termination [MMT24]. Moreover, only using ranking functions directly on the whole set of rules is limited in power. To yield a powerful termination tool in practice, one has to complement these ranking functions by the dependency pair method, which reduces termination proofs recursively into smaller sub-problems via a divide-and-conquer approach [AG00, GTSF06, HM05]. We adapt the dependency pair method to prove `AST` and `SAST` in Chapter 7 and Chapter 8, respectively.

Ranking functions can not only be used to prove termination, but also to infer upper complexity bounds, as shown in Corollary 3.2.68. The relation between termination proofs via ranking functions and corresponding upper bounds on the complexity has been investigated for the Knuth-Bendix order in [Lep01, Hof03, Mos06], for lexicographic path orders in [Wei95], for recursive path orders in [Hof90], for polynomial interpretations in [HL89, BCMT01], for ground TRSs in [Hof01], and finally, more generally in [MP15].

Other probabilistic programming formalisms for verification apart from PTRSs include pGCL [Koz81, Koz85, MM05], probabilistic integer transition systems [MHG21], probabilistic versions of the $\lambda$-calculus [DL20], and the rewriting-logic-based specification language PMaude [AMS06]. The semantic foundations of probabilistic programming and pGCL were laid by Kozen [Koz81, Koz85] and later developed extensively by

McIver and Morgan [MM05]. Furthermore, this weakest-precondition-style reasoning was extended to the analysis of expected runtimes in [KKMO16, KKMO18]. Integer transition systems have been successfully used as an intermediate model for termination and complexity analysis of programs [BEF+16] and have been extended to a probabilistic programming language in [MHG21]. For the probabilistic $\lambda$-calculus, there is a substantial line of work ranging from operational semantics to type-based termination and runtime analysis [DLZ12, DLG19, ADLG19, DL20, DLFDR21, BO21, KO21]. Compared to these other formalisms, PTRSs have specific advantages: they form a basic probabilistic formalism that is easy to analyze while still being expressive enough to model data structures. A concrete related-work section about verification techniques of probabilistic programs in different probabilistic programming formalisms can be found in Section 10.6.

Lastly, we conjecture that our comparison between `PAST` and `SAST` in Theorem 3.4.24 also holds for other programming formalisms that allow multiple executions to be considered in parallel.

Now that we have seen the rich literature in the field of termination analysis of probabilistic programs and in the rewriting setting, we also want to mention some open problems and future work.

First, while there already exist computable transformations from programming languages used in industry like Java and C to rewriting-based programming formalisms [OBvG10, SGB+17], there does not exist such a transformation in the probabilistic setting. To use all the results we present in this thesis in real-world applications, such a transformation is crucial. However, before defining such a transformation one has to analyze what kind of probabilistic programming languages are used the most in practice.

> Open Problem 3.5.1 – Practically Relevant Translations to PTRSs.
>
> Develop a termination-preserving translation from user-level probabilistic programming languages to PTRSs. The translation should keep the generated PTRS suitable for automated termination and runtime analysis.

Second, there is a plethora of future work to enhance current ranking functions used in the probabilistic setting. The two most notable approaches would be to extend more orders from the termination analysis of TRSs to PTRSs, and to extend more ranking functions and proof rules from imperative probabilistic programs to PTRSs.

> Open Problem 3.5.2 – Simplification Orders for PTRSs.
>
> Extend simplification orders to the probabilistic setting for `AST` and `SAST`.

Here, the main problem is to find a way to deal with expected values, which is the reason why we use polynomial interpretations. Consequently, it is worth investigating the power of other interpretation-based ranking functions in the probabilistic setting.

> Open Problem 3.5.3 – Evaluation of Matrix Interpretations.
>
> Evaluate the power of different variations of matrix interpretations and tuple interpretations for proving `AST` and `SAST` of PTRSs.

There also exist other proof rules in the probabilistic setting that one can investigate.

> Open Problem 3.5.4 – Transfer of Further Proof Rules.
>
> Transfer further proof rules like the one described in [MS25] regarding `AST` of probabilistic imperative programs to PTRSs.

Finally, the current computational model of PTRSs is still quite restricted. While it is Turing complete, multiple additional features could help in increasing the practical power of the formalism. One example is to allow more complex probability distributions in the right-hand sides of rules, and more specifically, to allow non-constant probabilities in such distributions that depend on the redex used.

> Open Problem 3.5.5 – Richer PTRS Models (1).
>
> Extend PTRSs to allow non-constant probabilities and complex distributions.

Moreover, one can try to integrate integers directly into the formalism, as in integer term rewrite systems [FGP+09] or logically constrained term rewrite systems [KN13].

> Open Problem 3.5.6 – Richer PTRS Models (2).
>
> Extend PTRSs to signatures with additional SMT-LIB theories.

Both extensions would significantly broaden the class of probabilistic programs that can be modeled without using abstraction.

# 4 Relating Evaluation Strategies

We have seen in Section 3.2.4 that for witnesses of innermost non-termination, we can guarantee stronger structural properties than for non-termination in general. This was a first glimpse of what is generally the case when proving termination and upper complexity bounds: *innermost evaluation is the easiest evaluation strategy to analyze*. With more structural properties on the non-termination witnesses, we are able to develop more sophisticated termination techniques that take advantage of these properties.

In Chapter 7 and Chapter 8 we present powerful techniques to prove $\mathtt{AST}$ and $\mathtt{SAST}$, respectively, based on a *dependency pair (DP) framework*. This is a *divide-and-conquer* framework to simplify termination proofs by splitting a large problem into simpler sub-problems. In the non-probabilistic setting, DPs are one of the most powerful termination techniques used in all the termination tools analyzing term rewriting. More importantly for this chapter, we show that the DP framework for innermost $\mathtt{AST}$ is substantially more powerful than the one for $\mathtt{AST}$. Indeed, already in the non-probabilistic setting, DPs for innermost rewriting are substantially more powerful than for full rewriting, see, e.g., [AG00, GTSF06]. The same holds for non-probabilistic complexity analysis, where the DP framework of [NEG13] is restricted to innermost rewriting and the framework of [AM16] is considerably more powerful for innermost than for full rewriting. Our adaptation of these DP frameworks for expected runtime complexity in Chapter 8 also requires fixing the innermost evaluation strategy.

To lift innermost termination proofs and bounds on the innermost runtime complexity to arbitrary evaluation strategies, in the non-probabilistic setting there exist several sufficient criteria which ensure that innermost termination implies full termination ($\mathtt{SN}(\xrightarrow{\mathtt{i}}_{\mathcal{R}}) \iff \mathtt{SN}(\xrightarrow{\mathtt{f}}_{\mathcal{R}})$) [Gra95] and that innermost runtime complexity coincides with full runtime complexity ($\mathrm{rc}_{\xrightarrow{\mathtt{f}}_{\mathcal{R}}} = \mathrm{rc}_{\xrightarrow{\mathtt{i}}_{\mathcal{R}}}$) [FG17a]. Remember that *full rewriting* simply refers to the ordinary rewrite relation $\xrightarrow{\mathtt{f}}_{\mathcal{R}} = \to_{\mathcal{R}}$ and is primarily used to avoid ambiguity when comparing evaluation strategies.

Up to now, no such results were known in the probabilistic setting. In this chapter, we present the first sufficient criteria for PTRSs which ensure that, e.g., innermost $\mathtt{AST}$ coincides with $\mathtt{AST}$ in general, and we also show similar results for other evaluation strategies like *leftmost-innermost* rewriting or *outermost* rewriting. We focus on criteria that can be checked automatically, so we can combine our results with the remaining termination techniques of this thesis. In this way, we can automatically lift proofs, e.g., for $\mathtt{AST}$, w.r.t. innermost rewriting to proofs for $\mathtt{AST}$ w.r.t. an arbitrary evaluation strategy. Our criteria also hold for $\mathtt{PAST}$, $\mathtt{SAST}$, and expected complexity.

While the relation between full runtime complexity and innermost runtime complexity of a given TRS has been studied in the literature, the relation between full derivational complexity and innermost derivational complexity has not been investigated. As a corollary of our results in the probabilistic setting, we also develop the first results relating derivational and innermost derivational complexity for ordinary non-probabilistic TRSs. This illustrates how investigating a more complex formalism like PTRSs can yield novel theoretical and practical insights for simpler formalisms like TRSs. By necessity, working with the complex formalism requires thinking beyond conventional boundaries, thereby uncovering novel results for the original formalism.

Instead of sufficient criteria that yield equivalence of innermost termination and full termination, one can also try a *transformational* approach. Here, the idea is to transform the given TRS $\mathcal{R}$ into another TRS $\mathcal{R}'$ such that (innermost) properties of $\mathcal{R}'$ imply properties of $\mathcal{R}$. We present two such transformations for the probabilistic setting. The first one (which is based on [Fuh19]) transforms a given PTRS $\mathcal{P}$ into a new PTRS $\mathcal{P}'$ such that the expected runtime complexity of $\mathcal{P}'$ yields an upper bound on the expected derivational complexity of the original system $\mathcal{P}$. This is again important, since the DP framework presented later is only able to analyze expected runtime complexity instead of expected derivational complexity. The second one (which is based on [Thi09]) transforms a given PTRS $\mathcal{P}$ into a new PTRS $\mathcal{P}'$ such that innermost $\mathtt{AST}$ of $\mathcal{P}'$ implies outermost $\mathtt{AST}$ of the original system $\mathcal{P}$. Both of the transformations suggest that many of the transformational approaches for TRSs also work in the probabilistic setting.

### Contributions

We present the following contributions in this chapter. The results marked with !⋆! are novel, while the remaining contributions can be found in the following peer-reviewed publication: ⋆ [KG25]. Whenever we speak about "termination" we mean all three notions $\mathtt{AST}$, $\mathtt{PAST}$, and $\mathtt{SAST}$.

- ⋆ Sufficient criteria for PTRSs ensuring that innermost termination coincides with full termination, extending results from the non-probabilistic setting (Theorem 4.2.2). The same criteria can be used to yield equivalence of expected complexity for full and innermost rewriting (Theorem 4.2.5).
- ⋆ Sufficient criteria for relating full termination to weak termination for PTRSs (Theorem 4.2.8).
- ⋆ Sufficient criteria for relating leftmost-innermost rewriting to innermost rewriting for PTRSs (Theorem 4.2.12).
- ⋆ First results relating derivational complexity and innermost derivational complexity for TRSs (Corollary 4.2.7).
- ⋆ Weakening of linearity requirements of the sufficient criteria mentioned above, improving applicability of termination proofs (Theorems 4.3.3 and 4.3.6).
- ⋆ Transformational approach to lift expected runtime complexity bounds to expected derivational complexity bounds (Theorems 4.4.3 and 4.4.5).

!★! Transformational approach to lift innermost AST proofs to outermost AST proofs (Theorem 4.4.8).

## Structure

Since the sufficient criteria for $\mathtt{SN}(\xrightarrow{\mathtt{i}}_{\mathcal{R}}) \iff \mathtt{SN}(\xrightarrow{\mathtt{f}}_{\mathcal{R}})$ in the non-probabilistic setting represent the minimum set of criteria we need in the probabilistic setting, we first recapitulate the relations between the different evaluation strategies for TRSs in Section 4.1. In Section 4.2 we show that the properties that ensure equivalence of innermost and full termination do not suffice in general for the probabilistic setting and extend them accordingly. In particular, we show that innermost and full AST coincide for PTRSs that are non-overlapping and linear. This result also holds for PAST, SAST, and the expected runtime complexity, as well as for strategies like leftmost-innermost evaluation. In Section 4.3 we try to weaken the linearity requirement in order to prove full AST for a larger class of PTRSs. Finally, in Section 4.4 we present transformational approaches to utilize the power of techniques for analyzing innermost rewriting and for runtime complexity in the analysis of other properties. We end this chapter by performing an experimental evaluation in Section 4.5, and discussing related work and giving important open problems for future work in Section 4.6.

SECTION 4.1

# Relating Evaluation Strategies for TRSs

In this section, we recapitulate the most important results on the relation between different evaluation strategies in the non-probabilistic setting. First, we consider the relations w.r.t. termination in Section 4.1.1, and then we consider the relations w.r.t. the different types of complexity in Section 4.1.2.

## 4.1.1 Termination

We start by recapitulating the relations between $\mathtt{SN}(\xrightarrow{\mathtt{f}}_{\mathcal{R}})$, $\mathtt{SN}(\xrightarrow{\mathtt{i}}_{\mathcal{R}})$, $\mathtt{SN}(\xrightarrow{\mathtt{o}}_{\mathcal{R}})$, $\mathtt{SN}(\xrightarrow{\mathtt{li}}_{\mathcal{R}})$, and $\mathtt{WN}(\xrightarrow{\mathtt{f}}_{\mathcal{R}})$ in the non-probabilistic setting. Obviously, the stronger notion always implies the weaker one, e.g., $\mathtt{SN}(\xrightarrow{\mathtt{f}}_{\mathcal{R}})$ implies $\mathtt{SN}(\xrightarrow{\mathtt{i}}_{\mathcal{R}})$ and $\mathtt{WN}(\xrightarrow{\mathtt{i}}_{\mathcal{R}})$ implies $\mathtt{WN}(\xrightarrow{\mathtt{f}}_{\mathcal{R}})$ since $\xrightarrow{\mathtt{i}}_{\mathcal{R}} \subseteq \xrightarrow{\mathtt{f}}_{\mathcal{R}}$ and both relations have the same normal forms. The interesting question is for which classes of TRSs the weaker and the stronger notion are equivalent.

The investigation of every relation always follows the same structure. First, we present a counterexample which illustrates why the relation does not hold in general. Next, we analyze the properties exhibited by such counterexamples. Finally, we state a corresponding theorem showing that, in the absence of these properties, the termination notions of the respective evaluation strategies are equivalent.

We start with the relation between $\mathtt{SN}(\xrightarrow{\mathtt{f}}_{\mathcal{R}})$ and $\mathtt{SN}(\xrightarrow{\mathtt{i}}_{\mathcal{R}})$.

Counterexample 4.1.1 (Toyama's Counterexample [Toy87a]). For the TRS $\mathcal{R}_1$ with the rules $\mathsf{f}(\mathsf{a},\mathsf{b},x) \to \mathsf{f}(x,x,x)$, $\mathsf{g} \to \mathsf{a}$, and $\mathsf{g} \to \mathsf{b}$, we do not have $\mathtt{SN}(\xrightarrow{\mathtt{f}}_{\mathcal{R}_1})$ due to the infinite rewrite sequence $\mathsf{f}(\mathsf{a},\mathsf{b},\mathsf{g}) \xrightarrow{\mathtt{f}}_{\mathcal{R}_1} \mathsf{f}(\mathsf{g},\mathsf{g},\mathsf{g}) \xrightarrow{\mathtt{f}}_{\mathcal{R}_1} \mathsf{f}(\mathsf{a},\mathsf{g},\mathsf{g}) \xrightarrow{\mathtt{f}}_{\mathcal{R}_1} \mathsf{f}(\mathsf{a},\mathsf{b},\mathsf{g}) \xrightarrow{\mathtt{f}}_{\mathcal{R}_1} \cdots$. The only innermost rewrite sequences starting with $\mathsf{f}(\mathsf{a},\mathsf{b},\mathsf{g})$ are $\mathsf{f}(\mathsf{a},\mathsf{b},\mathsf{g}) \xrightarrow{\mathtt{i}}_{\mathcal{R}_1} \mathsf{f}(\mathsf{a},\mathsf{b},\mathsf{a}) \xrightarrow{\mathtt{i}}_{\mathcal{R}_1} \mathsf{f}(\mathsf{a},\mathsf{a},\mathsf{a})$ and $\mathsf{f}(\mathsf{a},\mathsf{b},\mathsf{g}) \xrightarrow{\mathtt{i}}_{\mathcal{R}_1} \mathsf{f}(\mathsf{a},\mathsf{b},\mathsf{b}) \xrightarrow{\mathtt{i}}_{\mathcal{R}_1} \mathsf{f}(\mathsf{b},\mathsf{b},\mathsf{b})$,

and both of them reach normal forms in the end. In fact, $\mathtt{SN}(\xrightarrow{\mathtt{i}}_{\mathcal{R}_1})$ holds as we have to rewrite the inner $\mathsf{g}$ before we can use the $\mathsf{f}$-rule.

The first property that was known to ensure equivalence of $\mathtt{SN}(\xrightarrow{\mathtt{f}}_{\mathcal{R}})$ and $\mathtt{SN}(\xrightarrow{\mathtt{i}}_{\mathcal{R}})$ is orthogonality, which was already shown in [O'D77]. A TRS is *orthogonal* (OR) if it is non-overlapping and left-linear. The TRS $\mathcal{R}_1$ from Counterexample 4.1.1 is not orthogonal, because the two rules $\mathsf{g} \to \mathsf{a}$ and $\mathsf{g} \to \mathsf{b}$ are overlapping.

Theorem 4.1.2 – From $\mathtt{SN}(\xrightarrow{\mathtt{i}}_{\mathcal{R}})$ to $\mathtt{SN}(\xrightarrow{\mathtt{f}}_{\mathcal{R}})$ (1), [O'D77].
If a TRS $\mathcal{R}$ is OR, then $\mathtt{SN}(\xrightarrow{\mathtt{f}}_{\mathcal{R}}) \iff \mathtt{SN}(\xrightarrow{\mathtt{i}}_{\mathcal{R}})$.

*Proof.* For the proof we refer to [O'D77]. *Proof QED*

Then, in [Gra95], it was shown that one can remove the requirement of left-linearity.

Theorem 4.1.3 – From $\mathtt{SN}(\xrightarrow{\mathtt{i}}_{\mathcal{R}})$ to $\mathtt{SN}(\xrightarrow{\mathtt{f}}_{\mathcal{R}})$ (2), [Gra95].
If a TRS $\mathcal{R}$ is NO, then $\mathtt{SN}(\xrightarrow{\mathtt{f}}_{\mathcal{R}}) \iff \mathtt{SN}(\xrightarrow{\mathtt{i}}_{\mathcal{R}})$.

*Proof.* For the proof we refer to [Gra95]. *Proof QED*

Moreover, [Gra95] refined Theorem 4.1.3 further. A TRS $\mathcal{R}$ is an *overlay system* (OS) if its rules may only overlap at the root position, i.e., $\pi = \varepsilon$ in Definition 3.2.35. Furthermore, a TRS is *locally confluent* (also called *weakly confluent* or *weakly Church-Rosser*, abbreviated WCR) if for all terms $s, t_1, t_2 \in \mathcal{T}$ such that $t_1 \xleftarrow{\mathtt{f}}_{\mathcal{R}} s \xrightarrow{\mathtt{f}}_{\mathcal{R}} t_2$ the terms $t_1$ and $t_2$ are joinable, see Definition 3.1.14. For instance, $\mathcal{R}_1$ from Counterexample 4.1.1 is an overlay system. However, $\mathcal{R}_1$ is *not* WCR, as we have $\mathsf{a} \xleftarrow{\mathtt{f}}_{\mathcal{R}_1} \mathsf{g} \xrightarrow{\mathtt{f}}_{\mathcal{R}_1} \mathsf{b}$, but $\mathsf{a}$ and $\mathsf{b}$ are not joinable. If a TRS has both of these properties, then $\mathtt{SN}(\xrightarrow{\mathtt{i}}_{\mathcal{R}})$ and $\mathtt{SN}(\xrightarrow{\mathtt{f}}_{\mathcal{R}})$ are again equivalent.

Theorem 4.1.4 – From $\mathtt{SN}(\xrightarrow{\mathtt{i}}_{\mathcal{R}})$ to $\mathtt{SN}(\xrightarrow{\mathtt{f}}_{\mathcal{R}})$ (3), [Gra95].
If a TRS $\mathcal{R}$ is OS and WCR, then $\mathtt{SN}(\xrightarrow{\mathtt{f}}_{\mathcal{R}}) \iff \mathtt{SN}(\xrightarrow{\mathtt{i}}_{\mathcal{R}})$.

*Proof.* For the proof we refer to [Gra95]. *Proof QED*

Theorem 4.1.4 is stronger than Theorem 4.1.3 as every non-overlapping TRS is a locally confluent overlay system [KB70].

Next, we recapitulate the results on the relation between $\mathtt{WN}(\xrightarrow{\mathtt{f}}_{\mathcal{R}})$ and $\mathtt{SN}(\xrightarrow{\mathtt{f}}_{\mathcal{R}})$.

Counterexample 4.1.5. Consider the TRS $\mathcal{R}_2$ with the rules $\mathsf{f}(x) \to \mathsf{b}$ and $\mathsf{a} \to \mathsf{f}(\mathsf{a})$. We do not have $\mathtt{SN}(\xrightarrow{\mathtt{f}}_{\mathcal{R}_2})$ since we can always rewrite the inner $\mathsf{a}$ to get $\mathsf{a} \xrightarrow{\mathtt{f}}_{\mathcal{R}_2} \mathsf{f}(\mathsf{a}) \xrightarrow{\mathtt{f}}_{\mathcal{R}_2} \mathsf{f}(\mathsf{f}(\mathsf{a})) \xrightarrow{\mathtt{f}}_{\mathcal{R}_2} \cdots$. On the other hand, $\mathtt{WN}(\xrightarrow{\mathtt{f}}_{\mathcal{R}})$ holds: after using the $\mathsf{a}$-rule once, we can rewrite the outer $\mathsf{f}(\ldots)$ redex, resulting in the term $\mathsf{b}$, which is a normal form. For the TRS $\mathcal{R}_3$ with the rules $\mathsf{f}(\mathsf{a}) \to \mathsf{b}$ and $\mathsf{a} \to \mathsf{f}(\mathsf{a})$, the situation is similar.

The TRS $\mathcal{R}_2$ from Counterexample 4.1.5 is erasing and $\mathcal{R}_3$ is overlapping. For TRSs with neither of those two properties, $\mathtt{SN}(\xrightarrow{\mathtt{f}}_{\mathcal{R}})$ and $\mathtt{WN}(\xrightarrow{\mathtt{f}}_{\mathcal{R}})$ are equivalent.

Theorem 4.1.6 – From $\mathtt{WN}(\xrightarrow{\mathtt{f}}_{\mathcal{R}})$ to $\mathtt{SN}(\xrightarrow{\mathtt{f}}_{\mathcal{R}})$, [Gra95].
If a TRS $\mathcal{R}$ is NO and NE, then $\mathtt{SN}(\xrightarrow{\mathtt{f}}_{\mathcal{R}}) \iff \mathtt{WN}(\xrightarrow{\mathtt{f}}_{\mathcal{R}})$.

*Proof.* For the proof we refer to [Gra95]. *Proof QED*

The relation between full termination and weak termination relates full termination to

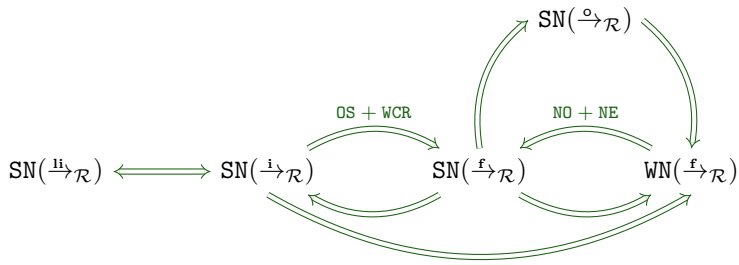



Figure 4.1: Relations between evaluation strategies w.r.t. termination.

every other evaluation strategy. For example, we directly get a corresponding result for the relation between full rewriting and outermost rewriting: If a TRS $\mathcal{R}$ is NO and NE, then $\mathtt{SN}(\xrightarrow{\mathtt{f}}_{\mathcal{R}}) \iff \mathtt{SN}(\xrightarrow{\mathtt{o}}_{\mathcal{R}})$. While for innermost rewriting, we were able to remove the NE property, for outermost rewriting, we cannot remove it as one can see in Counterexample 4.1.5, which not only works for weak termination, but also for outermost termination.

Finally, we look at the relation between evaluation strategies that use an ordering for parallel redexes like leftmost-innermost rewriting compared to innermost rewriting. It turns out that such an ordering does not interfere with termination at all.

Theorem 4.1.7 – From $\mathtt{SN}(\xrightarrow{\mathtt{li}}_{\mathcal{R}})$ to $\mathtt{SN}(\xrightarrow{\mathtt{i}}_{\mathcal{R}})$, [KR00].
For all TRSs $\mathcal{R}$ we have $\mathtt{SN}(\xrightarrow{\mathtt{i}}_{\mathcal{R}}) \iff \mathtt{SN}(\xrightarrow{\mathtt{li}}_{\mathcal{R}})$.

*Proof.* For the proof we refer to [KR00]. *Proof QED*

The relations between the different termination properties for non-probabilistic TRSs (given in Theorems 4.1.4, 4.1.6 and 4.1.7) are summarized in Figure 4.1. Note that we do not have to consider additional relations between, e.g., $\mathtt{WN}(\xrightarrow{\mathtt{li}}_{\mathcal{R}})$ and $\mathtt{WN}(\xrightarrow{\mathtt{i}}_{\mathcal{R}})$. The reason is that when analyzing under which conditions $\mathtt{WN}(\xrightarrow{\mathtt{f}}_{\mathcal{R}})$ implies $\mathtt{SN}(\xrightarrow{\mathtt{f}}_{\mathcal{R}})$, we also know that under these conditions $\mathtt{WN}(\xrightarrow{s}_{\mathcal{R}})$ implies $\mathtt{SN}(\xrightarrow{s}_{\mathcal{R}})$ for all strategies $s \in \{\mathbf{f}, \mathbf{i}, \mathbf{o}, \mathbf{li}\}$, since $\mathtt{WN}(\xrightarrow{s}_{\mathcal{R}}) \implies \mathtt{WN}(\xrightarrow{\mathtt{f}}_{\mathcal{R}})$ and $\mathtt{SN}(\xrightarrow{\mathtt{f}}_{\mathcal{R}}) \implies \mathtt{SN}(\xrightarrow{s}_{\mathcal{R}})$ hold (note that all these strategies have the same normal forms as full rewriting, see Section 3.2.3). Moreover, we have $\mathtt{SN}(\xrightarrow{s}_{\mathcal{R}}) \implies \mathtt{WN}(\xrightarrow{s}_{\mathcal{R}})$ for all $s \in \{\mathbf{f}, \mathbf{i}, \mathbf{o}, \mathbf{li}\}$.

### 4.1.2 Complexity

There are not only criteria to relate evaluation strategies w.r.t. termination, but similar results also exist for complexity. In the following, we recapitulate results from the literature on the complexity of TRSs under different evaluation strategies. Remember that there are two standard notions of complexity used in term rewriting: *derivational* and *runtime complexity*.

Prior to the results of this chapter, no non-trivial classes of TRSs were known where innermost and full derivational complexity coincide, i.e., where we have $\mathrm{dc}_{\xrightarrow{\mathtt{f}}_{\mathcal{R}}}(n) = \mathrm{dc}_{\xrightarrow{\mathtt{i}}_{\mathcal{R}}}(n)$ for all $n \in \mathbb{N}$ and all TRSs $\mathcal{R}$ from that class. However, for runtime complexity, there exist sufficient criteria for TRSs $\mathcal{R}$ which imply $\mathrm{rc}_{\xrightarrow{\mathtt{f}}_{\mathcal{R}}} = \mathrm{rc}_{\xrightarrow{\mathtt{i}}_{\mathcal{R}}}$ [FG17a]. To relate innermost and full runtime complexity, it does not suffice to require that the rules are non-overlapping, but we also have to make sure that we cannot duplicate redexes during an evaluation, as shown by the following counterexample from [FG17a].

Counterexample 4.1.8. Let $\mathcal{R}_4$ consist of the rules

$$\mathsf{f}(0, y) \to y \qquad \mathsf{g}(x) \to \mathsf{f}(x, \mathsf{a})$$
$$\mathsf{f}(\mathsf{s}(x), y) \to \mathsf{f}(x, \mathsf{node}(y, y)) \qquad \mathsf{a} \to \mathsf{b}$$

Note that $\mathcal{R}_4$ is non-overlapping. For the basic term $\mathsf{g}(\mathsf{s}^n(0))$ of size $n+2$ we have $\mathsf{g}(\mathsf{s}^n(0)) \xrightarrow{\mathtt{f}}_{\mathcal{R}_4} \mathsf{f}(\mathsf{s}^n(0), \mathsf{a})$. Next, one could apply the second $\mathsf{f}$-rule repeatedly and obtain a term corresponding to a full binary tree of height $n$ whose exponentially many leaves all correspond to the symbol $\mathsf{a}$. Finally, these leaves can all be reduced to $\mathsf{b}$ in $2^n$ steps, hence $\mathrm{rc}_{\xrightarrow{\mathtt{f}}_{\mathcal{R}_4}}(n) \in \Theta(2^n)$ and $\iota(\mathrm{rc}_{\xrightarrow{\mathtt{f}}_{\mathcal{R}_4}}) = \mathbf{Exp}$. On the other hand, any basic term of size $n$ only leads to innermost rewrite sequences of length $\mathcal{O}(n)$, as for example in $\mathsf{f}(\mathsf{s}^n(0), \mathsf{a})$ we have to rewrite the inner $\mathsf{a}$ before we are able to duplicate it. Thus, we obtain $\mathrm{rc}_{\xrightarrow{\mathtt{i}}_{\mathcal{R}_4}}(n) \in \Theta(n)$ and $\iota(\mathrm{rc}_{\xrightarrow{\mathtt{i}}_{\mathcal{R}_4}}) = \mathbf{Pol}_1$.

The property that no redexes are duplicated during a rewrite sequence that starts with a basic term is called *spareness*.

Definition 4.1.9 – Spareness.

*For a TRS $\mathcal{R}$, a rewrite step using the rule $\ell \to r \in \mathcal{R}$ and the substitution $\sigma$ is spare if $\sigma(x) \in \mathtt{NF}_{\mathcal{R}}$ for every $x \in \mathcal{V}$ that occurs more than once in $r$. A $\to_{\mathcal{R}}$-rewrite sequence is spare if each of its $\to_{\mathcal{R}}$-steps is spare. $\mathcal{R}$ is spare ($\mathtt{SP}$) if each $\to_{\mathcal{R}}$-rewrite sequence that starts with a basic term $t \in \mathcal{T}_{\mathcal{B}}$ is spare.*

In general, it is undecidable whether a TRS is spare. However, there exist computable sufficient conditions for spareness, see [FG17a].

Similar to the results regarding termination, for the equivalence of $\mathrm{rc}_{\xrightarrow{\mathtt{f}}_{\mathcal{R}}}$ and $\mathrm{rc}_{\xrightarrow{\mathtt{i}}_{\mathcal{R}}}$ it suffices to require overlay systems instead of non-overlapping TRSs. However, while we have to require spareness, in contrast to Theorem 4.1.4 we do not need local confluence.

Theorem 4.1.10 – From $\mathrm{rc}_{\xrightarrow{\mathtt{i}}_{\mathcal{R}}}$ to $\mathrm{rc}_{\xrightarrow{\mathtt{f}}_{\mathcal{R}}}$, [FG17a].

If a TRS $\mathcal{R}$ is $\mathtt{OS}$ and $\mathtt{SP}$, then $\mathrm{rc}_{\xrightarrow{\mathtt{f}}_{\mathcal{R}}} = \mathrm{rc}_{\xrightarrow{\mathtt{i}}_{\mathcal{R}}}$.

*Proof.* For the proof we refer to [FG17a]. *Proof QED*

Counterexample 4.1.8 also works for derivational complexity, showing that duplication of redexes needs to be prohibited for equivalence of $\mathrm{dc}_{\xrightarrow{\mathtt{f}}_{\mathcal{R}}}$ and $\mathrm{dc}_{\xrightarrow{\mathtt{i}}_{\mathcal{R}}}$ as well. However, note that *spareness* requires basic start terms, so an analogous theorem for derivational complexity would have to require a stronger property, e.g., right-linearity of $\mathcal{R}$. We will present such a result as a corollary of our analysis of probabilistic rewriting in Section 4.2 (Corollary 4.2.7). Finally, the relation between the runtime complexities for evaluation strategies that use an ordering for parallel redexes has not been studied so far. Here, our probabilistic analysis will also yield a corresponding corollary (Corollary 4.2.15).

SECTION 4.2

## Relating Evaluation Strategies for PTRSs

The goal for this section is to relate the different evaluation strategies in terms of all the probabilistic termination properties ($\mathtt{AST}$, $\mathtt{PAST}$, and $\mathtt{SAST}$) and expected complexity. To be precise, we first relate full rewriting to innermost rewriting in Section 4.2.1. Then we relate full termination to weak termination in Section 4.2.2, and in the end,

we relate innermost to leftmost-innermost rewriting in Section 4.2.3. We want to find properties of a PTRS $\mathcal{P}$ which are suitable for automated checking and which guarantee that, e.g., $\mathtt{AST}(\xrightarrow{s_1}_{\mathcal{P}}) \iff \mathtt{AST}(\xrightarrow{s_2}_{\mathcal{P}})$ for $s_1, s_2 \in \{\mathbf{f}, \mathbf{i}, \mathbf{o}, \mathbf{li}\}$. Then, for example, we can use existing tools that analyze $\mathtt{AST}(\xrightarrow{\mathbf{i}}_{\mathcal{P}})$ in order to prove $\mathtt{AST}(\xrightarrow{\mathbf{f}}_{\mathcal{P}})$, if we have successfully checked the properties that guarantee equivalence of $\mathtt{AST}(\xrightarrow{\mathbf{f}}_{\mathcal{P}})$ and $\mathtt{AST}(\xrightarrow{\mathbf{i}}_{\mathcal{P}})$. Let $\mathtt{PSN} \in \{\mathtt{AST}, \mathtt{PAST}, \mathtt{SAST}\}$. As most of our results hold for all these three termination properties, we use $\mathtt{PSN}$ (*probabilistic strong normalization*) to refer to all of them. Similarly, we use $\mathtt{wPSN} \in \{\mathtt{wAST}, \mathtt{wPAST}\}$ for the corresponding weak notion, where $\mathtt{wPAST}$ also serves as the weak counterpart of $\mathtt{SAST}$, since there is no weak variant of $\mathtt{SAST}$ (see Section 3.3.1). Clearly, we have to require at least the same properties as in the non-probabilistic setting, as every TRS $\mathcal{R}$ can be transformed into a non-probabilistic PTRS $\mathcal{P}$ by replacing every rule $\ell \to r$ with $\ell \to \{^1 r\}$. Then for every $s \in \{\mathbf{f}, \mathbf{i}, \mathbf{o}, \mathbf{li}\}$, we have $\mathtt{SN}(\xrightarrow{s}_{\mathcal{R}}) \iff \mathtt{AST}(\xrightarrow{s}_{\mathcal{P}})$ and $\mathtt{WN}(\xrightarrow{\mathbf{f}}_{\mathcal{R}}) \iff \mathtt{wAST}(\xrightarrow{\mathbf{f}}_{\mathcal{P}})$.

The following subsections are all structured similarly. We first give counterexamples to show why the criteria from the non-probabilistic setting do not carry over to the probabilistic setting. Then, we explain the criteria needed in the probabilistic setting, state the corresponding theorem, and usually give a lemma to explain the main proof idea. We also use these lemmas to reason about expected complexity.

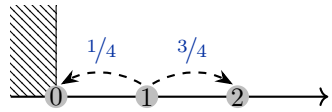
### 4.2.1 From Innermost to Full Rewriting

We start by analyzing the relation between innermost and full rewriting. The following example shows that Theorem 4.1.2 does not carry over to the probabilistic setting, i.e., orthogonality is not sufficient to ensure that $\mathtt{PSN}(\xrightarrow{\mathbf{i}}_{\mathcal{P}})$ implies $\mathtt{PSN}(\xrightarrow{\mathbf{f}}_{\mathcal{P}})$.

Counterexample 4.2.1 (Orthogonality Does Not Suffice). Consider the orthogonal PTRS $\mathcal{P}_{\mathsf{dup}}$ with the two rules shown in Figure 4.2.

$$\mathsf{g} \to \{^{3/4}\mathsf{d}(\mathsf{g}), {}^{1/4}0\}$$
$$\mathsf{d}(x) \to \{^1\mathsf{c}(x, x)\}$$

Figure 4.2: Rules of $\mathcal{P}_{\mathsf{dup}}$.




Count $\|t\| = |t|_{\mathsf{g}}$

Figure 4.3: Positively drifting random walk.

We do not have $\mathtt{AST}(\xrightarrow{\mathbf{f}}_{\mathcal{P}_{\mathsf{dup}}})$ (hence also neither $\mathtt{PAST}(\xrightarrow{\mathbf{f}}_{\mathcal{P}_{\mathsf{dup}}})$ nor $\mathtt{SAST}(\xrightarrow{\mathbf{f}}_{\mathcal{P}_{\mathsf{dup}}})$), because $\{^1\mathsf{g}\} \xrightarrow{\mathbf{f}}{}^2_{\mathcal{P}_{\mathsf{dup}}} \{^{3/4}\mathsf{c}(\mathsf{g}, \mathsf{g}), {}^{1/4}0\}$, which corresponds to a positively drifting random walk on the number $\|t\| = |t|_{\mathsf{g}}$ of $\mathsf{g}$ symbols in a term $t$, see Figure 4.3.

However, the $\mathsf{d}$-rule can only duplicate normal forms in innermost evaluations. To see that we have $\mathtt{SAST}(\xrightarrow{\mathbf{i}}_{\mathcal{P}_{\mathsf{dup}}})$ (hence $\mathtt{PAST}(\xrightarrow{\mathbf{i}}_{\mathcal{P}_{\mathsf{dup}}})$ and $\mathtt{AST}(\xrightarrow{\mathbf{i}}_{\mathcal{P}_{\mathsf{dup}}})$), consider the only possible innermost rewrite sequence $\vec{\mu}$ starting with $\mathsf{g}$, which is

$$\{^1\mathsf{g}\} \xrightarrow{\mathbf{i}}_{\mathcal{P}_{\mathsf{dup}}} \{^{3/4}\mathsf{d}(\mathsf{g}), {}^{1/4}0\} \xrightarrow{\mathbf{i}}_{\mathcal{P}_{\mathsf{dup}}} \{^{(3/4)^2}\mathsf{d}(\mathsf{d}(\mathsf{g})),\ {}^{1/4\cdot 3/4}\mathsf{d}(0),\ {}^{1/4}0\} \xrightarrow{\mathbf{i}}_{\mathcal{P}_{\mathsf{dup}}} \cdots .$$

The corresponding $\xrightarrow{\mathbf{i}}_{\mathcal{P}_{\mathsf{dup}}}$-RT for this $\xrightarrow{\mathbf{i}}_{\mathcal{P}_{\mathsf{dup}}}$-sequence is depicted in Figure 4.4.

The branch to the right that starts with $0$ stops after 0 innermost steps, the branch that starts with $\mathsf{d}(0)$ stops after 1 innermost step, the branch that starts with $\mathsf{d}(\mathsf{d}(0))$ stops after 2 innermost steps, and so on. So if we start with the term $\mathsf{d}^n(0)$, then we reach a normal form after $n$ steps, and we reach $\mathsf{d}^n(0)$ after $n+1$ steps from the initial term $\mathsf{g}$. For every $k \in \mathbb{N}$ we have $|\mu_{2\cdot k+1}|_{\mathcal{P}_{\mathsf{dup}}} = |\mu_{2\cdot k+2}|_{\mathcal{P}_{\mathsf{dup}}} = \sum_{n=0}^{k} 1/4 \cdot (3/4)^n$

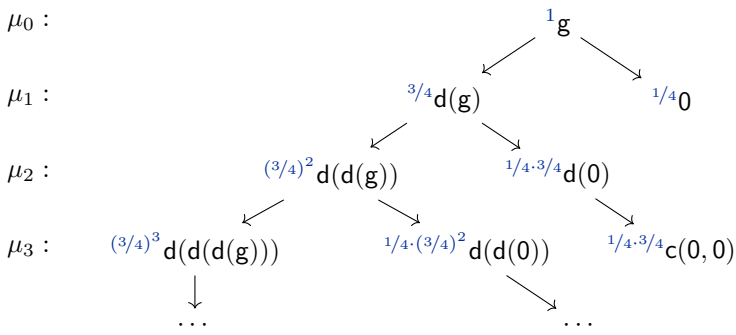



Figure 4.4: The corresponding $\xrightarrow{\mathsf{i}}_{\mathcal{P}_{\mathsf{dup}}}$-RT of the $\overset{\mathsf{i}}{\Rightarrow}_{\mathcal{P}_{\mathsf{dup}}}$-sequence.

and thus

$$\begin{array}{rclcl} \mathrm{edl}(\vec{\mu}) & = & \sum_{n=0}^{\infty}(1-|\mu_n|_{\mathcal{P}_{\mathsf{dup}}}) & = & 1+2\cdot\sum_{k\in\mathbb{N}}(1-|\mu_{2\cdot k+1}|_{\mathcal{P}_{\mathsf{dup}}}) \\ & = & 1+2\cdot\sum_{k\in\mathbb{N}}(1-\sum_{n=0}^{k}1/4\cdot(3/4)^n) & = & 1+2\cdot\sum_{k\in\mathbb{N}}(3/4)^{k+1} \\ & = & (2\cdot\sum_{k\in\mathbb{N}}(3/4)^k)-1 & = & 7. \end{array}$$

Due to innermost rewriting, there is no non-determinism in this sequence, i.e., when starting with $\{^{1}\mathsf{g}\}$, there is no rewrite sequence with higher expected derivation length. Thus, we have $\mathrm{edh}_{\xrightarrow{\mathsf{i}}_{\mathcal{P}_{\mathsf{dup}}}}(\mathsf{g}) = 7$. Analogously, in all other innermost rewrite sequences, the $\mathsf{d}$-rule can only duplicate normal forms. Thus, all terms have finite expected derivation height w.r.t. innermost rewriting. Therefore, $\mathtt{SAST}(\xrightarrow{\mathsf{i}}_{\mathcal{P}_{\mathsf{dup}}})$ and thus, $\mathtt{PAST}(\xrightarrow{\mathsf{i}}_{\mathcal{P}_{\mathsf{dup}}})$ and $\mathtt{AST}(\xrightarrow{\mathsf{i}}_{\mathcal{P}_{\mathsf{dup}}})$ hold.

To construct a counterexample, we exploited the fact that $\mathcal{P}_{\mathsf{dup}}$ is not right-linear, which allows us to duplicate the redex $\mathsf{g}$ repeatedly during the rewrite sequence. Similar to the complexity analysis in the non-probabilistic setting (see Counterexample 4.1.8), we need to prohibit the duplication of redexes. Indeed, requiring right-linearity prevents this kind of duplication and yields our desired result.

Theorem 4.2.2 – From $\mathtt{PSN}(\xrightarrow{\mathsf{i}}_{\mathcal{P}})$ to $\mathtt{PSN}(\xrightarrow{\mathsf{f}}_{\mathcal{P}})$.

If a PTRS $\mathcal{P}$ is $\mathtt{OR}$ and $\mathtt{RL}$ (i.e., $\mathtt{NO}$ and linear), then

$$\mathtt{PSN}(\xrightarrow{\mathsf{f}}_{\mathcal{P}}) \iff \mathtt{PSN}(\xrightarrow{\mathsf{i}}_{\mathcal{P}}).$$

To prove Theorem 4.2.2, we prove the following lemma.

Lemma 4.2.3 – From Innermost to Full Rewriting.

*If a PTRS $\mathcal{P}$ is $\mathtt{OR}$ and $\mathtt{RL}$ (i.e., $\mathtt{NO}$ and linear), then for every $\xrightarrow{\mathsf{f}}_{\mathcal{P}}$-RT $\mathfrak{T}$ starting with some $t \in \mathcal{T}$ there exists a $\xrightarrow{\mathsf{i}}_{\mathcal{P}}$-RT $\mathfrak{T}'$ starting with $t$ such that*

1. $|\mathfrak{T}'| \leq |\mathfrak{T}|$ (*Lower bound on the termination probability*)
2. $\mathrm{edl}(\mathfrak{T}') \geq \mathrm{edl}(\mathfrak{T})$ (*Upper bound on the expected derivation length*)

*Proof.* The proofs for all such lemmas in this chapter follow a similar structure. We iteratively replace rewrite steps that do not adhere to the desired evaluation strategy by ones that use the desired evaluation strategy, and we ensure that this neither increases the probability of convergence nor decreases the expected derivation length. *Proof*

In the end, we regard the limit of a sequence of such transformation steps.
Let $\mathcal{P}$ be a PTRS that is non-overlapping and linear. Furthermore, let $\mathfrak{T}$ be an $\xrightarrow{\mathrm{f}}_{\mathcal{P}}$-RT. We create a new $\xrightarrow{\mathrm{i}}_{\mathcal{P}}$-RT $\mathfrak{T}^{(\infty)}$ such that $|\mathfrak{T}^{(\infty)}| \leq |\mathfrak{T}|$ and $\mathrm{edl}(\mathfrak{T}^{(\infty)}) \geq \mathrm{edl}(\mathfrak{T})$. W.l.o.g., at least one rewrite step in $\mathfrak{T}$ is performed at some node $x$ with a redex that is not an innermost redex (otherwise we can use $\mathfrak{T}^{(\infty)} = \mathfrak{T}$). The core steps of the proof are:

1. We iteratively move innermost rewrite steps to a higher position in the tree using a construction $\Phi(\,\cdot\,)$. The limit of this iteration, namely $\mathfrak{T}^{(\infty)}$, is the desired $\xrightarrow{\mathrm{i}}_{\mathcal{P}}$-RT with $|\mathfrak{T}^{(\infty)}| \leq |\mathfrak{T}|$. During each iteration step we
   1.1. formally define the construction $\Phi(\,\cdot\,)$ that replaces a certain subtree $\mathfrak{T}_x$ by a new subtree $\Phi(\mathfrak{T}_x)$ with an innermost step to the root of $\Phi(\mathfrak{T}_x)$,
   1.2. show $|\Phi(\mathfrak{T}_x)| = |\mathfrak{T}_x|$, and
   1.3. show that $\Phi(\mathfrak{T}_x)$ is indeed a valid RT.
2. We show that the same construction also guarantees $\mathrm{edl}(\mathfrak{T}^{(\infty)}) \geq \mathrm{edl}(\mathfrak{T})$.
   2.1. First we prove $\mathrm{edl}(\mathfrak{T}) \leq \mathrm{edl}(\mathfrak{T}^{(1)}) \leq \mathrm{edl}(\mathfrak{T}^{(2)}) \leq \cdots$.
   2.2. Then, we prove $\mathrm{edl}(\mathfrak{T}) \leq \mathrm{edl}(\mathfrak{T}^{(\infty)})$.

**Step 1** Move innermost rewrite steps to a higher position

In $\mathfrak{T}$ there exists at least one rewrite step performed at some node $x$, which is not an innermost rewrite step. Furthermore, we can assume that this is the first such rewrite step in the path from the root to the node $x$ and that $x$ is a node of minimum depth with this property. Let $\mathfrak{T}_x$ be the subtree that starts at node $x$, i.e., $\mathfrak{T}_x = \mathfrak{T}[xE^*]$, where $xE^*$ is the set of all reachable nodes (via the edge relation) from $x$. We then construct a new tree $\Phi(\mathfrak{T}_x)$ such that $|\Phi(\mathfrak{T}_x)| = |\mathfrak{T}_x|$, where we use an innermost rewrite step at the root node $x$ instead of the old one, i.e., we pushed the first non-innermost rewrite step deeper into the tree. This construction only works because $\mathcal{P}$ is non-overlapping and linear. Then, by replacing the subtree $\mathfrak{T}_x$ with the new tree $\Phi(\mathfrak{T}_x)$ in $\mathfrak{T}$, we obtain an $\xrightarrow{\mathrm{f}}_{\mathcal{P}}$-RT $\mathfrak{T}^{(1)}$ with $|\mathfrak{T}^{(1)}| = |\mathfrak{T}|$, where we use an innermost rewrite step at node $x$ instead of the old rewrite step, as desired. We can then do such a replacement iteratively for every use of a non-innermost rewrite step, i.e., we again replace the first non-innermost rewrite step in $\mathfrak{T}^{(1)}$ to obtain $\mathfrak{T}^{(2)}$ with $|\mathfrak{T}^{(2)}| = |\mathfrak{T}^{(1)}|$, and so on. In the end, the limit of all these RTs $\lim_{i\to\infty} \mathfrak{T}^{(i)}$ is an $\xrightarrow{\mathrm{i}}_{\mathcal{P}}$-RT, which we denote by $\mathfrak{T}^{(\infty)}$, such that $|\mathfrak{T}^{(\infty)}| \leq |\mathfrak{T}|$. So while the termination probability remains the same in each step, it can decrease in the limit. As an example, consider a tree $\mathfrak{T}$ which is just a finite path, whose path length increases by one in each iteration. Then the limit $\mathfrak{T}^{(\infty)}$ is an infinite path and converges with probability 0, while $\mathfrak{T}, \mathfrak{T}^{(1)}, \ldots$ all converge with probability 1.
To see that $\mathfrak{T}^{(\infty)}$ is indeed a valid $\xrightarrow{\mathrm{i}}_{\mathcal{P}}$-RT, note that in every iteration of the construction we turn a non-innermost rewrite step at minimum depth into an innermost one. Moreover, every node of an RT has only finitely many successors, since the right-hand sides of the rules are finite multi-distributions (this also holds if $\mathcal{P}$ is infinite). So every RT has only finitely many nodes up to any depth. Hence, for every depth $H$ of the tree, we eventually turned every non-innermost rewrite step up to depth $H$ into an innermost one. So the construction will not change the tree above depth $H$ anymore, i.e., there exists an $m_H$ such that $\mathfrak{T}^{(\infty)}$ and $\mathfrak{T}^{(i)}$ are the same trees up to depth $H$ for all $i \geq m_H$. This means that the sequence $\lim_{i\to\infty} \mathfrak{T}^{(i)}$ really converges into an $\xrightarrow{\mathrm{i}}_{\mathcal{P}}$-RT.

Next, we want to prove that we have $|\mathfrak{T}^{(\infty)}| \leq |\mathfrak{T}|$. By induction on $n$ one can prove that $|\mathfrak{T}^{(i)}| = |\mathfrak{T}|$ for all $1 \leq i \leq n$, since we have $|\mathfrak{T}^{(i)}| = |\mathfrak{T}^{(i-1)}|$ for all $i \geq 2$ and $|\mathfrak{T}^{(1)}| = |\mathfrak{T}|$. Assume for a contradiction that $|\mathfrak{T}^{(\infty)}| > |\mathfrak{T}|$. Then there exists a depth $H \in \mathbb{N}$ such that $\sum_{x \in \mathrm{Leaf}^{\mathfrak{T}^{(\infty)}}, \mathrm{d}^{\mathfrak{T}^{(\infty)}}(x) \leq H} p_x > |\mathfrak{T}|$. Again, let $m_{H+1} \in \mathbb{N}$ such that $\mathfrak{T}^{(\infty)}$ and $\mathfrak{T}^{(m_{H+1})}$ are the same trees up to depth $H+1$ (so that they also have the same leaves up to depth $H$). But this would mean that $|\mathfrak{T}^{(m_{H+1})}| \geq \sum_{x \in \mathrm{Leaf}^{\mathfrak{T}^{(m_{H+1})}}, \mathrm{d}^{\mathfrak{T}^{(m_{H+1})}}(x) \leq H} p_x^{\mathfrak{T}^{(m_{H+1})}} = \sum_{x \in \mathrm{Leaf}^{\mathfrak{T}^{(\infty)}}, \mathrm{d}^{\mathfrak{T}^{(\infty)}}(x) \leq H} p_x^{\mathfrak{T}^{(\infty)}} > |\mathfrak{T}|$, which is a contradiction to $|\mathfrak{T}^{(m_{H+1})}| = |\mathfrak{T}|$.

**Step 1.1** Construction of $\Phi(\cdot)$

It remains to define the construction $\Phi(\cdot)$ mentioned above. Let $\mathfrak{T}_x$ be an $\xrightarrow{\mathrm{f}}_{\mathcal{P}}$-RT that performs a non-innermost rewrite step at the root node $x$. This step has the form $t_x^{\mathfrak{T}_x} \xrightarrow{\mathrm{f}}_{\mathcal{P}} \{{}^{p_{y_1}^{\mathfrak{T}_x}} t_{y_1}^{\mathfrak{T}_x}, \ldots, {}^{p_{y_k}^{\mathfrak{T}_x}} t_{y_k}^{\mathfrak{T}_x}\}$ using the rule $\bar{\ell} \to \{{}^{\bar{p}_1}\bar{r}_1, \ldots, {}^{\bar{p}_k}\bar{r}_k\}$, the substitution $\bar{\sigma}$, and the position $\bar{\pi}$ such that $t_x^{\mathfrak{T}_x}|_{\bar{\pi}} = \bar{\ell}\bar{\sigma}$. Then we have $t_{y_j}^{\mathfrak{T}_x} = t_x^{\mathfrak{T}_x}[\bar{r}_j\bar{\sigma}]_{\bar{\pi}}$ for all $1 \leq j \leq k$.

Instead of applying a non-innermost rewrite step at the root $x$ we want to directly apply an innermost rewrite step. Let $\tau$ be the position of some innermost redex in $t_x^{\mathfrak{T}_x}$ below $\bar{\pi}$. The construction creates a new $\xrightarrow{\mathrm{f}}_{\mathcal{P}}$-RT $\Phi(\mathfrak{T}_x) = (N', E')$ that starts with $t_x^{\mathfrak{T}_x}$ such that $|\Phi(\mathfrak{T}_x)| = |\mathfrak{T}_x|$, and that directly performs the first rewrite step at position $\tau$ in the original tree $\mathfrak{T}_x$ (which is an innermost rewrite step) at the root of the tree, by pushing it from the original nodes in the tree $\mathfrak{T}_x$ to the root of the new tree $\Phi(\mathfrak{T}_x)$. (It could also be that this innermost redex was never reduced in $\mathfrak{T}_x$.)

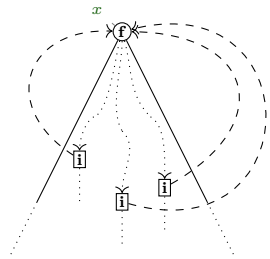



Figure 4.5: Tree $\mathfrak{T}_x$ in the construction of $\Phi(\cdot)$.

This can be seen in the diagram in Figure 4.5, which depicts the tree $\mathfrak{T}_x$. The boxes represent **i**nnermost rewrite steps at position $\tau$ and the dashed lines indicate that we push this rewrite step to the root. Since we are allowed to rewrite above $\tau$ in the original tree $\mathfrak{T}_x$, the actual position of the innermost redex that was originally at position $\tau$ might change during the application of a rewrite step. Hence, we recursively define the position $\varphi_\tau(v)$ that contains precisely this redex for each node $v$ in $\mathfrak{T}_x$ until we rewrite at this position. Initially, we have $\varphi_\tau(x) = \tau$. Whenever we have defined $\varphi_\tau(v)$ for some node $v$, and we have $t_v^{\mathfrak{T}_x} \xrightarrow{\mathrm{f}}_{\mathcal{P}} \{{}^{p_{w_1}^{\mathfrak{T}_x}} t_{w_1}^{\mathfrak{T}_x}, \ldots, {}^{p_{w_m}^{\mathfrak{T}_x}} t_{w_m}^{\mathfrak{T}_x}\}$ for the direct successors $vE = \{w_1, \ldots, w_m\}$, using the rule $\ell \to \{{}^{p_1}r_1, \ldots, {}^{p_m}r_m\}$, the substitution $\sigma$, and the position $\pi$, we do the following: If $\varphi_\tau(v) = \pi$, meaning that we rewrite this innermost redex, then we set $\varphi_\tau(w_j) = \bot$ for all $1 \leq j \leq m$ to indicate that we have rewritten the innermost redex. If we have $\varphi_\tau(v) \bot \pi$, meaning that the rewrite step takes place on a position that is parallel to $\varphi_\tau(v)$, then we set $\varphi_\tau(w_j) = \varphi_\tau(v)$ for all $1 \leq j \leq m$, as the position of the innermost redex did not change. Otherwise, we have $\pi < \varphi_\tau(v)$ (since we cannot rewrite below $\varphi_\tau(v)$ as it is an innermost redex), and thus there exists a $\chi \in \mathbb{N}^+$ such that $\pi.\chi = \varphi_\tau(v)$. Since the rules of $\mathcal{P}$ are non-overlapping, the redex must be completely "inside" the used substitution $\sigma$, and we can find a position $\alpha_q$ of a variable $q$ in $\ell$ and another position $\beta$ such that $\chi = \alpha_q.\beta$. Furthermore, since the rule is linear, $q$ only occurs once in $\ell$ and at most once in $r_j$ for all $1 \leq j \leq m$. If $q$ occurs in $r_j$ at a position $\rho_q^j$, then we set $\varphi_\tau(w_j) = \pi.\rho_q^j.\beta$.

Otherwise, we set $\varphi_\tau(w_j) = \top$ to indicate that the innermost redex was erased during the computation. Finally, if $\varphi_\tau(v) \in \{\bot, \top\}$, then we set $\varphi_\tau(w_j) = \varphi_\tau(v)$ for all $1 \leq j \leq m$ as well. So to summarize, $\varphi_\tau(v)$ is now either the position of the innermost redex in $t_v$, $\top$ to indicate that the redex was erased, or $\bot$ to indicate that we have rewritten the redex.

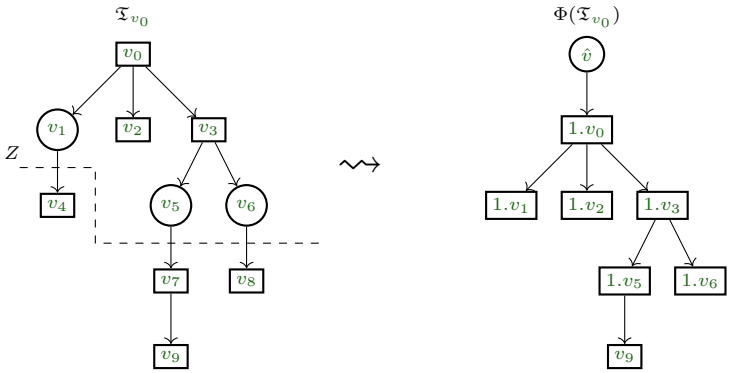



Figure 4.6: Illustration of $\Phi(\mathfrak{T}_x)$.

In Figure 4.6, we illustrate the effect of $\Phi$, where the original tree $\mathfrak{T}_x$ with $x = v_0$ is on the left and $\Phi(\mathfrak{T}_x)$ is on the right. The circled nodes represent the nodes where we perform a rewrite step at position $\varphi_\tau(v)$. We now define the $\xrightarrow{\mathrm{f}}_{\mathcal{P}}$-RT $\Phi(\mathfrak{T}_x)$ that starts with $t_x^{\mathfrak{T}_x}$ and that directly performs the rewrite step $t_x^{\mathfrak{T}_x} = t_{\hat{v}}^{\Phi(\mathfrak{T}_x)} \xrightarrow{\mathrm{i}}_{\mathcal{P},\tau} \{^{\hat{p}_1}t_{1.x}^{\Phi(\mathfrak{T}_x)}, \ldots, ^{\hat{p}_h}t_{h.x}^{\Phi(\mathfrak{T}_x)}\}$, with the rule $\hat{\ell} \to \{^{\hat{p}_1}\hat{r}_1, \ldots, ^{\hat{p}_h}\hat{r}_h\} \in \mathcal{P}$, a substitution $\hat{\sigma}$, and the position $\tau$, at the new root $\hat{v}$. We have $t_{\hat{v}}^{\Phi(\mathfrak{T}_x)}|_\tau = \hat{\ell}\hat{\sigma}$. Let $Z$ be the set of all nodes $z$ such that $\varphi_\tau(z) \neq \bot$, i.e., at a node $z \in Z$ we did not yet perform the rewrite step with the innermost redex. In the example we have $Z = \{v_0, \ldots, v_6\} \setminus \{v_4\}$. For each of these nodes $z \in Z$ and each $1 \leq e \leq h$, we create a new node $e.z \in N'$ with edges as in $\mathfrak{T}_x$ for the nodes in $Z$, e.g., for the node $1.v_3$ we create edges to $1.v_5$ and $1.v_6$. Furthermore, we add the edges from the new root $\hat{v}$ to the nodes $e.x$ for all $1 \leq e \leq h$. Note that $x$ was the root in the tree $\mathfrak{T}_x$ and has to be contained in $Z$. For example, for the node $\hat{v}$ we create an edge to $1.v_0$. We define the labeling of the nodes in $\Phi(\mathfrak{T}_x)$ as follows for all nodes $z$ in $Z$:

(T-1) $t_{e.z}^{\Phi(\mathfrak{T}_x)} = t_z^{\mathfrak{T}_x}[\hat{r}_e\hat{\sigma}]_{\varphi_\tau(z)}$ if $\varphi_\tau(z) \in \mathbb{N}^*$ and $t_{e.z}^{\Phi(\mathfrak{T}_x)} = t_z^{\mathfrak{T}_x}$ if $\varphi_\tau(z) = \top$

(T-2) $p_{e.z}^{\Phi(\mathfrak{T}_x)} = p_z^{\mathfrak{T}_x} \cdot \hat{p}_e$

Now, for a leaf $e.z' \in N'$ either $z' \in N$ is also a leaf (e.g., node $v_2$ in our example) or we rewrite the innermost redex at position $\varphi_\tau(z')$ at node $z'$ in $\mathfrak{T}_x$ (e.g., node $v_1$ in our example). In the latter case, due to non-overlappingness, the same rule $\hat{\ell} \to \{^{\hat{p}_1}\hat{r}_1, \ldots, ^{\hat{p}_h}\hat{r}_h\}$ and the same substitution $\hat{\sigma}$ were used. If we rewrite $t_{z'}^{\mathfrak{T}_x} \xrightarrow{\mathrm{i}}_{\mathcal{P},\varphi_\tau(z')} \{^{\hat{p}_1}t_{w_1'}^{\mathfrak{T}_x}, \ldots, ^{\hat{p}_h}t_{w_h'}^{\mathfrak{T}_x}\}$, then we have $t_{w_e'}^{\mathfrak{T}_x} = t_{z'}^{\mathfrak{T}_x}[\hat{r}_e\hat{\sigma}]_{\varphi_\tau(z')} \overset{\text{(T-1)}}{=} t_{e.z'}^{\Phi(\mathfrak{T}_x)}$ and $p_{w_e'}^{\mathfrak{T}_x} \overset{\text{(T-2)}}{=} p_{z'}^{\mathfrak{T}_x} \cdot \hat{p}_e = p_{e.z'}^{\Phi(\mathfrak{T}_x)}$. Thus, we can copy the rest of this subtree of $\mathfrak{T}_x$ to our newly generated tree $\Phi(\mathfrak{T}_x)$. In our example, $v_1$ has the only successor $v_4$, hence we can copy the subtree starting at node $v_4$, which is only the node itself, to the node $1.v_1$ in $\Phi(\mathfrak{T}_x)$. For $v_5$, we have the only successor $v_7$, hence we can copy the subtree starting at node $v_7$, which is the node itself together with its successor $v_9$, to the node

$1.v_5$ in $\Phi(\mathfrak{T}_x)$. So essentially, we just had to define how to construct $\Phi(\mathfrak{T}_x)$ for the part of the tree before we reach nodes $v$ with $\varphi_\tau(v) = \bot$ in $\mathfrak{T}_x$. Now we have to show that $|\Phi(\mathfrak{T}_x)| = |\mathfrak{T}_x|$ and that $\Phi(\mathfrak{T}_x)$ is indeed a valid $\xrightarrow{\mathrm{f}}_{\mathcal{P}}$-RT (i.e., that the edges between nodes $e.z$ with $z \in Z$ and its successors correspond to rewrite steps with $\mathcal{P}$).

**Step 1.2** Show $|\Phi(\mathfrak{T}_x)| = |\mathfrak{T}_x|$

Let $v$ be a leaf in $\Phi(\mathfrak{T}_x)$. If $v = e'.z$ for some node $z \in Z$ that is a leaf in $\mathfrak{T}_x$ (e.g., node $1.v_2$), then also $e.z$ must be a leaf in $\Phi(\mathfrak{T}_x)$ for every $1 \leq e \leq h$. Here, we get $\sum_{1\leq e\leq h} p^{\Phi(\mathfrak{T}_x)}_{e.z} \overset{\text{(T-2)}}{=} \sum_{1\leq e\leq h} p^{\mathfrak{T}_x}_z \cdot \hat{p}_e = p^{\mathfrak{T}_x}_z \cdot \sum_{1\leq e\leq h} \hat{p}_e = p^{\mathfrak{T}_x}_z \cdot 1 = p^{\mathfrak{T}_x}_z$, and thus

$$\sum_{\substack{e.z\in \mathrm{Leaf}^{\Phi(\mathfrak{T}_x)}\\ z\in Z, z\in \mathrm{Leaf}^{\mathfrak{T}_x}}} p^{\Phi(\mathfrak{T}_x)}_{e.z} = \sum_{\substack{z\in \mathrm{Leaf}^{\mathfrak{T}_x}\\ z\in Z}} \left( \sum_{1\leq e\leq h} p^{\Phi(\mathfrak{T}_x)}_{e.z} \right) = \sum_{\substack{z\in \mathrm{Leaf}^{\mathfrak{T}_x}\\ z\in Z}} p^{\mathfrak{T}_x}_z.$$

If $v = e.z$ for some node $z \in Z$ that is not a leaf in $\mathfrak{T}_x$ (e.g., node $1.v_1$), then we know by construction that the $e$-th successor $w_e$ of $z$ in $\mathfrak{T}_x$ is not contained in $Z$ and is a leaf of $\mathfrak{T}_x$. Let $(zE)_e$ denote the $e$-th successor of $z$. Here, we get $p^{\Phi(\mathfrak{T}_x)}_{e.z} \overset{\text{(T-2)}}{=} p^{\mathfrak{T}_x}_z \cdot \hat{p}_e = p^{\mathfrak{T}_x}_{w_e}$, and thus

$$\sum_{\substack{e.z\in \mathrm{Leaf}^{\Phi(\mathfrak{T}_x)}\\ z\in Z, z\notin \mathrm{Leaf}^{\mathfrak{T}_x}}} p^{\Phi(\mathfrak{T}_x)}_{e.z} = \sum_{\substack{w_e\in \mathrm{Leaf}^{\mathfrak{T}_x}\\ w_e=(zE)_e, w_e\notin Z, z\in Z}} p^{\Phi(\mathfrak{T}_x)}_{e.z} = \sum_{\substack{w_e\in \mathrm{Leaf}^{\mathfrak{T}_x}\\ w_e=(zE)_e, w_e\notin Z, z\in Z}} p^{\mathfrak{T}_x}_{w_e}.$$

Finally, if $v$ does not have the form $v = e.z$, then $v$ is also a leaf in $\mathfrak{T}_x$ with $p^{\Phi(\mathfrak{T}_x)}_v = p^{\mathfrak{T}_x}_v$ and for both $v$ and its predecessor $u$ we have $v, u \notin Z$, and thus

$$\sum_{\substack{v\in \mathrm{Leaf}^{\Phi(\mathfrak{T}_x)}\\ v\in \mathrm{Leaf}^{\mathfrak{T}_x}}} p^{\Phi(\mathfrak{T}_x)}_v = \sum_{\substack{v\in \mathrm{Leaf}^{\mathfrak{T}_x}\\ v\in uE, u\notin Z}} p^{\Phi(\mathfrak{T}_x)}_v = \sum_{\substack{v\in \mathrm{Leaf}^{\mathfrak{T}_x}\\ v\in uE, u\notin Z}} p^{\mathfrak{T}_x}_v.$$

Note that these cases cover each leaf of $\mathfrak{T}_x$ exactly once. These three equations imply:

$$\begin{aligned} |\Phi(\mathfrak{T}_x)| &= \sum_{z\in \mathrm{Leaf}^{\Phi(\mathfrak{T}_x)}} p^{\Phi(\mathfrak{T}_x)}_z = \sum_{\substack{e.z\in \mathrm{Leaf}^{\Phi(\mathfrak{T}_x)}\\ z\in Z, z\in \mathrm{Leaf}^{\mathfrak{T}_x}}} p^{\Phi(\mathfrak{T}_x)}_{e.z} + \sum_{\substack{e.z\in \mathrm{Leaf}^{\Phi(\mathfrak{T}_x)}\\ z\in Z, z\notin \mathrm{Leaf}^{\mathfrak{T}_x}}} p^{\Phi(\mathfrak{T}_x)}_{e.z} + \sum_{\substack{v\in \mathrm{Leaf}^{\Phi(\mathfrak{T}_x)}\\ v\in \mathrm{Leaf}^{\mathfrak{T}_x}}} p^{\Phi(\mathfrak{T}_x)}_v \\ &= \sum_{\substack{z\in \mathrm{Leaf}^{\mathfrak{T}_x}\\ z\in Z}} p^{\mathfrak{T}_x}_z + \sum_{\substack{w_e\in \mathrm{Leaf}^{\mathfrak{T}_x}\\ w_e=(zE)_e, w_e\notin Z, z\in Z}} p^{\mathfrak{T}_x}_{w_e} + \sum_{\substack{v\in \mathrm{Leaf}^{\mathfrak{T}_x}\\ v\in uE, u\notin Z}} p^{\mathfrak{T}_x}_v = \sum_{z\in \mathrm{Leaf}^{\mathfrak{T}_x}} p^{\mathfrak{T}_x}_z = |\mathfrak{T}_x|. \end{aligned}$$

**Step 1.3** Show that $\Phi(\mathfrak{T}_x)$ is indeed a valid RT.

Finally, we prove that $\Phi(\mathfrak{T}_x)$ is a valid $\xrightarrow{\mathrm{f}}_{\mathcal{P}}$-RT. Here, we only need to show that $t^{\Phi(\mathfrak{T}_x)}_{e.z} \xrightarrow{\mathrm{f}}_{\mathcal{P}} \{p^{\Phi(\mathfrak{T}_x)}_{e.w_1} t^{\Phi(\mathfrak{T}_x)}_{e.w_1}, \ldots, p^{\Phi(\mathfrak{T}_x)}_{e.w_m} t^{\Phi(\mathfrak{T}_x)}_{e.w_m}\}$ for all $z \in Z$ as all the other edges and labelings were already present in $\mathfrak{T}_x$, which is a valid $\xrightarrow{\mathrm{f}}_{\mathcal{P}}$-RT, and we have already seen that we have a valid innermost rewrite step at the new root $\hat{v}$.
Let $z \in Z$. In the following, we distinguish between two different cases for a rewrite step at a node $z$ of $\mathfrak{T}_x$:

(A) We rewrite at a position parallel to $\varphi_\tau(z)$ or we have $\varphi_\tau(z) = \top$.

(B) We rewrite at a position above $\varphi_\tau(z)$. Here, we need that $\mathcal{P}$ is NO and linear.

Let $t_z^{\mathfrak{T}_x} \xrightarrow{\mathtt{f}}_{\mathcal{P}} \{p_z^{\mathfrak{T}_x} \cdot p_1 t_{w_1}^{\mathfrak{T}_x}, \dots, p_z^{\mathfrak{T}_x} \cdot p_m t_{w_m}^{\mathfrak{T}_x}\}$, with a rule $\ell \to \{p_1 r_1, \dots, p_m r_m\} \in \mathcal{P}$, a substitution $\sigma$, and a position $\pi$ such that $t_z^{\mathfrak{T}_x}|_\pi = \ell\sigma$. We have $t_{w_j}^{\mathfrak{T}_x} = t_z^{\mathfrak{T}_x}[r_j\sigma]_\pi$ for all $1 \le j \le m$.

**(A)** We start with the case where we have $\pi \perp \varphi_\tau(z)$ or $\varphi_\tau(z) = \top$. By (T-1), we get $t_{e.z}^{\Phi(\mathfrak{T}_x)} = t_z^{\mathfrak{T}_x}[\hat{r}_e\hat{\sigma}]_{\varphi_\tau(z)}$ if $\varphi_\tau(z) \in \mathbb{N}^*$ and $t_{e.z}^{\Phi(\mathfrak{T}_x)} = t_z^{\mathfrak{T}_x}$ if $\varphi_\tau(z) = \top$. In both cases, we can rewrite $t_{e.z}^{\Phi(\mathfrak{T}_x)}$ using the same rule, the same substitution, and the same position, as we have $t_{e.z}^{\Phi(\mathfrak{T}_x)}|_\pi = t_z^{\mathfrak{T}_x}[\hat{r}_e\hat{\sigma}]_{\varphi_\tau(z)}|_\pi = t_z^{\mathfrak{T}_x}|_\pi = \ell\sigma$ or directly $t_{e.z}^{\Phi(\mathfrak{T}_x)}|_\pi = t_z^{\mathfrak{T}_x}|_\pi = \ell\sigma$. It remains to show that $t_{e.w_j}^{\Phi(\mathfrak{T}_x)} = t_{e.z}^{\Phi(\mathfrak{T}_x)}[r_j\sigma]_\pi$ for all $1 \le j \le m$, i.e., that the labeling we defined for $\Phi(\mathfrak{T}_x)$ corresponds to this rewrite step. Let $1 \le j \le m$. If $\varphi_\tau(z) \in \mathbb{N}^*$, then we have $t_{e.z}^{\Phi(\mathfrak{T}_x)}[r_j\sigma]_\pi = t_z^{\mathfrak{T}_x}[\hat{r}_e\hat{\sigma}]_{\varphi_\tau(z)}[r_j\sigma]_\pi \overset{\varphi_\tau(z)\perp\pi}{=} t_z^{\mathfrak{T}_x}[r_j\sigma]_\pi[\hat{r}_e\hat{\sigma}]_{\varphi_\tau(z)} = t_{w_j}^{\mathfrak{T}_x}[\hat{r}_e\hat{\sigma}]_{\varphi_\tau(z)} = t_{e.w_j}^{\Phi(\mathfrak{T}_x)}$. If $\varphi_\tau(z) = \top$, then $t_{e.z}^{\Phi(\mathfrak{T}_x)}[r_j\sigma]_\pi = t_z^{\mathfrak{T}_x}[r_j\sigma]_\pi = t_{w_j}^{\mathfrak{T}_x} = t_{e.w_j}^{\Phi(\mathfrak{T}_x)}$. Finally, note that the probabilities of the labeling are correct, as we are using the same rule with the same probabilities.

**(B)** If we have $\pi < \varphi_\tau(z)$, then there exists a $\chi \in \mathbb{N}^+$ such that $\pi.\chi = \varphi_\tau(z)$. Since the rules of $\mathcal{P}$ are non-overlapping, the redex must be completely "inside" the used substitution $\sigma$, and we can find a position $\alpha_q$ of a variable $q$ in $\ell$ and another position $\beta$ such that $\chi = \alpha_q.\beta$. Furthermore, since the rule is linear, $q$ only occurs once in $\ell$ and at most once in $r_j$ for all $1 \le j \le m$. Let $\rho_q^j$ be the position of $q$ in $r_j$ if it exists. By (T-1), we get $t_z^{\mathfrak{T}_x}[\hat{r}_e\hat{\sigma}]_{\varphi_\tau(z)} = t_{e.z}^{\Phi(\mathfrak{T}_x)}$. We can rewrite $t_{e.z}^{\Phi(\mathfrak{T}_x)}$ using the same rule, the substitution $\sigma'$ with $\sigma'(q) = \sigma(q)[\hat{r}_e\hat{\sigma}]_\beta$ and $\sigma'(q') = \sigma(q')$ for all other variables $q' \neq q$, and the same position, as we have $t_{e.z}^{\Phi(\mathfrak{T}_x)}|_\pi = t_z^{\mathfrak{T}_x}[\hat{r}_e\hat{\sigma}]_{\varphi_\tau(z)}|_\pi = t_z^{\mathfrak{T}_x}|_\pi[\hat{r}_e\hat{\sigma}]_\chi = \ell\sigma'$.
It remains to show that $t_{e.w_j}^{\Phi(\mathfrak{T}_x)} = t_{e.z}^{\Phi(\mathfrak{T}_x)}[r_j\sigma']_\pi$ for all $1 \le j \le m$, i.e., that the labeling we defined for $\Phi(\mathfrak{T}_x)$ corresponds to this rewrite step. Let $1 \le j \le m$. If $\rho_q^j$ exists, then we have $t_{e.z}^{\Phi(\mathfrak{T}_x)}[r_j\sigma']_\pi = t_z^{\mathfrak{T}_x}[\hat{r}_e\hat{\sigma}]_{\varphi_\tau(z)}[r_j\sigma']_\pi = t_z^{\mathfrak{T}_x}[\hat{r}_e\hat{\sigma}]_{\pi.\alpha_q.\beta}[r_j\sigma']_\pi = t_z^{\mathfrak{T}_x}[r_j\sigma]_\pi[\hat{r}_e\hat{\sigma}]_{\pi.\rho_q^j.\beta} = t_{w_j}^{\mathfrak{T}_x}[\hat{r}_e\hat{\sigma}]_{\varphi_\tau(w_j)} = t_{e.w_j}^{\Phi(\mathfrak{T}_x)}$. Otherwise, we erase the precise redex and obtain $t_{e.z}^{\Phi(\mathfrak{T}_x)}[r_j\sigma']_\pi = t_z^{\mathfrak{T}_x}[\hat{r}_e\hat{\sigma}]_{\varphi_\tau(z)}[r_j\sigma']_\pi = t_z^{\mathfrak{T}_x}[\hat{r}_e\hat{\sigma}]_{\pi.\alpha_q.\beta}[r_j\sigma']_\pi = t_z^{\mathfrak{T}_x}[r_j\sigma]_\pi = t_{w_j}^{\mathfrak{T}_x} = t_{e.w_j}^{\Phi(\mathfrak{T}_x)}$. Note that $q$ does not occur in $r_j$, and thus, $r_j\sigma' = r_j\sigma$. Again, the probabilities of the labeling are correct, as we are using the same rule with the same probabilities.

**Step 2** Analyze the expected derivation length of $\mathfrak{T}^{(\infty)}$

Our goal is to show that in the construction of the trees $\mathfrak{T}^{(1)}, \mathfrak{T}^{(2)}, \dots$, every leaf $v$ of $\mathfrak{T}$ is turned into leaves of $\mathfrak{T}^{(1)}, \mathfrak{T}^{(2)}, \dots$ whose probabilities sum up to $p_v^{\mathfrak{T}}$, and whose depths are greater than or equal to the original depth $\mathrm{d}^{\mathfrak{T}}(v)$ of $v$ in $\mathfrak{T}$. This implies $\mathrm{edl}(\mathfrak{T}) \le \mathrm{edl}(\mathfrak{T}^{(1)}) \le \mathrm{edl}(\mathfrak{T}^{(2)}) \le \cdots$. From this observation, we can then conclude $\mathrm{edl}(\mathfrak{T}) \le \mathrm{edl}(\mathfrak{T}^{(\infty)})$.

**Step 2.1** Prove $\mathrm{edl}(\mathfrak{T}) \le \mathrm{edl}(\mathfrak{T}^{(1)}) \le \mathrm{edl}(\mathfrak{T}^{(2)}) \le \cdots$

We start by considering how the leaves of $\mathfrak{T}$ correspond to the leaves of $\mathfrak{T}^{(1)}$. Let $v$ be a leaf in $\mathfrak{T}$. If $v \notin xE^*$, then $v$ is also a leaf in $\mathfrak{T}^{(1)}$, labeled with the same probability, and at the same depth. Otherwise, $v \in xE^*$, which means that either $v \in Z$, or $v \notin Z$

but its predecessor is in $Z$, or neither $v$ nor its predecessor are in $Z$.
If $v \in Z$ (like the node $v_2$ in Figure 4.6), then also $e.v$ must be a leaf in $\Phi(\mathfrak{T}_x)$ for every $1 \leq e \leq h$. Here, we again have $\sum_{1 \leq e \leq h} p_{e.v}^{\Phi(\mathfrak{T}_x)} = p_v^{\mathfrak{T}_x}$ as before. In addition, we also know that if $e.v$ is at depth $m$ in $\Phi(\mathfrak{T}_x)$, then $v$ is at depth $m-1$ in $\mathfrak{T}_x$.
If $v \notin Z$ but $v$ is the $e$-th successor of a node $z \in Z$ (like the node $v_8$ in Figure 4.6), then $e.z$ is a leaf in $\Phi(\mathfrak{T}_x)$. Here, we again have $p_{e.z}^{\Phi(\mathfrak{T}_x)} = p_v^{\mathfrak{T}_x}$ as before. In addition, we also know that if $e.z$ is at depth $m$ in $\Phi(\mathfrak{T}_x)$, then $v$ is at depth $m$ in $\mathfrak{T}_x$.
Finally, if neither $v$ nor its predecessor are in $Z$ (like the node $v_9$ in Figure 4.6), then $v$ is also a leaf in $\Phi(\mathfrak{T}_x)$ with $p_v^{\Phi(\mathfrak{T}_x)} = p_v^{\mathfrak{T}_x}$ and at the same depth.
These cases cover each leaf of $\mathfrak{T}^{(1)}$ exactly once and all leaves in $\Phi(\mathfrak{T}_x)$ are at a depth greater than or equal to that of the corresponding leaves in $\mathfrak{T}_x$, implying that for each leaf $u$ in $\mathfrak{T}$ we can find a set of leaves $\Omega_u^{(1)}$ in $\mathfrak{T}^{(1)}$ such that $\mathrm{d}^{\mathfrak{T}}(u) \leq \mathrm{d}^{\mathfrak{T}^{(1)}}(w)$ for each $w \in \Omega_u^{(1)}$, $\sum_{w \in \Omega_u^{(1)}} p_w^{\mathfrak{T}^{(1)}} = p_u^{\mathfrak{T}}$, and $\mathrm{Leaf}^{\mathfrak{T}^{(1)}} = \biguplus_{u \in \mathrm{Leaf}^{\mathfrak{T}}} \Omega_u^{(1)}$.
We can now use the same observation for each leaf $w$ in the tree $\mathfrak{T}^{(1)}$ in order to obtain a new set $\Gamma_w^{(2)}$ of leaves in $\mathfrak{T}^{(2)}$ and define $\Omega_u^{(2)} = \bigcup_{w \in \Omega_u^{(1)}} \Gamma_w^{(2)}$. Again, we get $\mathrm{d}^{\mathfrak{T}}(u) \leq \mathrm{d}^{\mathfrak{T}^{(2)}}(w)$ for each $w \in \Omega_u^{(2)}$, $\sum_{w \in \Omega_u^{(2)}} p_w^{\mathfrak{T}^{(2)}} = p_u^{\mathfrak{T}}$, and $\mathrm{Leaf}^{\mathfrak{T}^{(2)}} = \biguplus_{u \in \mathrm{Leaf}^{\mathfrak{T}}} \Omega_u^{(2)}$.
We now do this for each $i \in \mathbb{N}$ to define the set $\Omega_u^{(i)}$ for each leaf $u$ in the original tree $\mathfrak{T}$. Overall, this implies $\mathrm{edl}(\mathfrak{T}) \leq \mathrm{edl}(\mathfrak{T}^{(1)}) \leq \mathrm{edl}(\mathfrak{T}^{(2)}) \leq \cdots$.

**Step 2.2** Prove $\mathrm{edl}(\mathfrak{T}^{(\infty)}) \geq \mathrm{edl}(\mathfrak{T})$

From the construction of the $\Omega_u^{(i)}$ above, in the end, we obtain

$$\mathrm{Leaf}^{\mathfrak{T}^{(\infty)}} = \biguplus_{u \in \mathrm{Leaf}^{\mathfrak{T}}} \limsup_{i \to \infty} \Omega_u^{(i)},$$

where $\limsup_{i \to \infty} \Omega_u^{(i)} = \bigcap_{i' \in \mathbb{N}} \bigcup_{i > i'} \Omega_u^{(i)} = \{w \mid w \text{ is in infinitely many } \Omega_u^{(i)}\}$. To see this, let $v \in \mathrm{Leaf}^{\mathfrak{T}^{(\infty)}}$, and remember that for every depth $H$ of the tree, there exists an $m_H$ such that $\mathfrak{T}^{(\infty)}$ and $\mathfrak{T}^{(i)}$ are the same trees up to depth $H$ for all $i \geq m_H$. This means that the node $v$ must be a leaf in all trees $\mathfrak{T}^{(m)}$ with $m \geq m_{\mathrm{d}^{\mathfrak{T}^{(\infty)}}(v)+1}$, i.e., it is contained in $\biguplus_{u \in \mathrm{Leaf}^{\mathfrak{T}}} \limsup_{i \to \infty} \Omega_u^{(i)}$. The other direction of the equality follows in the same manner.
Since the probabilities of the leaves in $\mathfrak{T}^{(i)}$ always add up to the probability of the corresponding leaf in $\mathfrak{T}$, we have $\sum_{v \in \limsup_{i \to \infty} \Omega_u^{(i)}} p_v^{\mathfrak{T}^{(\infty)}} \leq p_u^{\mathfrak{T}}$ for all $u \in \mathrm{Leaf}^{\mathfrak{T}}$. We show $\mathrm{edl}(\mathfrak{T}) \leq \mathrm{edl}(\mathfrak{T}^{(\infty)})$ by considering the two cases: (1) $\sum_{v \in \limsup_{i \to \infty} \Omega_u^{(i)}} p_v^{\mathfrak{T}^{(\infty)}} < p_u^{\mathfrak{T}}$ for some $u \in \mathrm{Leaf}^{\mathfrak{T}}$, and (2) $\sum_{v \in \limsup_{i \to \infty} \Omega_u^{(i)}} p_v^{\mathfrak{T}^{(\infty)}} = p_u^{\mathfrak{T}}$ for all $u \in \mathrm{Leaf}^{\mathfrak{T}}$.

(1) If we have $\sum_{v \in \limsup_{i \to \infty} \Omega_u^{(i)}} p_v^{\mathfrak{T}^{(\infty)}} < p_u^{\mathfrak{T}}$ for some $u \in \mathrm{Leaf}^{\mathfrak{T}}$, then we obtain

$$\begin{aligned} 1 &\geq |\mathfrak{T}| &&= \textstyle\sum_{u \in \mathrm{Leaf}^{\mathfrak{T}}} p_u^{\mathfrak{T}} \\ &> \textstyle\sum_{u \in \mathrm{Leaf}^{\mathfrak{T}}} \sum_{v \in \limsup_{i \to \infty} \Omega_u^{(i)}} p_v^{\mathfrak{T}^{(\infty)}} &&= \textstyle\sum_{v \in \biguplus_{u \in \mathrm{Leaf}^{\mathfrak{T}}} \limsup_{i \to \infty} \Omega_u^{(i)}} p_v^{\mathfrak{T}^{(\infty)}} \\ &= \textstyle\sum_{v \in \mathrm{Leaf}^{\mathfrak{T}^{(\infty)}}} p_v^{\mathfrak{T}^{(\infty)}} &&= |\mathfrak{T}^{(\infty)}| \end{aligned}$$

and from $|\mathfrak{T}^{(\infty)}| < 1$ we directly get $\mathrm{edl}(\mathfrak{T}^{(\infty)}) = \infty$.

(2) Otherwise, we have $\sum_{v \in \limsup_{i \to \infty} \Omega_u^{(i)}} p_v^{\mathfrak{T}^{(\infty)}} = p_u^{\mathfrak{T}}$ for all $u \in \mathrm{Leaf}^{\mathfrak{T}}$, and thus we obtain

$$
\begin{aligned}
\mathrm{edl}(\mathfrak{T}) &= \sum_{n=0}^{\infty} \sum_{\substack{u \in N^{\mathfrak{T}} \setminus \mathrm{Leaf}^{\mathfrak{T}} \\ \mathrm{d}^{\mathfrak{T}}(u)=n}} p_u^{\mathfrak{T}} \\
&\quad \downarrow \quad \Big(\sum_{\substack{u \in N \setminus \mathrm{Leaf} \\ \mathrm{d}^{\mathfrak{T}}(u)=n}} p_u^{\mathfrak{T}} = 1 - \sum_{\substack{u \in \mathrm{Leaf} \\ \mathrm{d}^{\mathfrak{T}}(u) \leq n}} p_u^{\mathfrak{T}}\Big) \\
&= \sum_{n=0}^{\infty} \Big(1 - \sum_{\substack{u \in \mathrm{Leaf}^{\mathfrak{T}} \\ \mathrm{d}^{\mathfrak{T}}(u) \leq n}} p_u^{\mathfrak{T}}\Big) \\
&\quad \downarrow \quad \Big(p_u^{\mathfrak{T}} = \sum_{v \in \limsup_{i \to \infty} \Omega_u^{(i)}} p_v^{\mathfrak{T}^{(\infty)}}\Big) \\
&= \sum_{n=0}^{\infty} \Big(1 - \sum_{\substack{u \in \mathrm{Leaf}^{\mathfrak{T}} \\ \mathrm{d}^{\mathfrak{T}}(u) \leq n}} \sum_{v \in \limsup_{i \to \infty} \Omega_u^{(i)}} p_v^{\mathfrak{T}^{(\infty)}}\Big) \\
&\quad \downarrow \quad \big(\forall v \in \limsup_{i \to \infty} \Omega_u^{(i)}.\, \mathrm{d}^{\mathfrak{T}}(u) \leq \mathrm{d}^{\mathfrak{T}^{(\infty)}}(v)\big) \\
&\leq \sum_{n=0}^{\infty} \Big(1 - \sum_{u \in \mathrm{Leaf}^{\mathfrak{T}}} \sum_{\substack{v \in \limsup_{i \to \infty} \Omega_u^{(i)} \\ \mathrm{d}^{\mathfrak{T}^{(\infty)}}(v) \leq n}} p_v^{\mathfrak{T}^{(\infty)}}\Big) \\
&\quad \downarrow \quad \big(\Omega_u^{(i)} \text{ all disjoint}\big) \\
&= \sum_{n=0}^{\infty} \Big(1 - \sum_{\substack{v \in \biguplus_{u \in \mathrm{Leaf}^{\mathfrak{T}}} \limsup_{i \to \infty} \Omega_u^{(i)} \\ \mathrm{d}^{\mathfrak{T}^{(\infty)}}(v) \leq n}} p_v^{\mathfrak{T}^{(\infty)}}\Big) \\
&\quad \downarrow \quad \Big(\mathrm{Leaf}^{\mathfrak{T}^{(\infty)}} = \biguplus_{u \in \mathrm{Leaf}^{\mathfrak{T}}} \limsup_{i \to \infty} \Omega_u^{(i)}\Big) \\
&= \sum_{n=0}^{\infty} \Big(1 - \sum_{\substack{v \in \mathrm{Leaf}^{\mathfrak{T}^{(\infty)}} \\ \mathrm{d}^{\mathfrak{T}^{(\infty)}}(v) \leq n}} p_v^{\mathfrak{T}^{(\infty)}}\Big) \\
&\quad \downarrow \quad \Big(\sum_{\substack{u \in N \setminus \mathrm{Leaf}^{\mathfrak{T}^{(\infty)}} \\ \mathrm{d}^{\mathfrak{T}^{(\infty)}}(u)=n}} p_u^{\mathfrak{T}^{(\infty)}} = 1 - \sum_{\substack{v \in \mathrm{Leaf}^{\mathfrak{T}^{(\infty)}} \\ \mathrm{d}^{\mathfrak{T}^{(\infty)}}(v) \leq n}} p_v^{\mathfrak{T}^{(\infty)}}\Big) \\
&= \sum_{n=0}^{\infty} \sum_{\substack{u \in N \setminus \mathrm{Leaf}^{\mathfrak{T}^{(\infty)}} \\ \mathrm{d}^{\mathfrak{T}^{(\infty)}}(u)=n}} p_u^{\mathfrak{T}^{(\infty)}} \\
&= \mathrm{edl}(\mathfrak{T}^{(\infty)})
\end{aligned}
$$

and therefore, $\mathrm{edl}(\mathfrak{T}^{(\infty)}) \geq \mathrm{edl}(\mathfrak{T})$. *QED*

With Lemma 4.2.3 we can now prove Theorem 4.2.2.

*Proof of Theorem 4.2.2.* We only need to prove the "$\Longleftarrow$" direction. Let us first consider $\mathtt{AST}(\xrightarrow{\mathtt{f}}_{\mathcal{P}})$. Assume that $\mathcal{P}$ is `NO` and linear, and that $\mathtt{AST}(\xrightarrow{\mathtt{f}}_{\mathcal{P}})$ does not hold. Then, there exists an infinite $\xrightarrow{\mathtt{f}}_{\mathcal{P}}$-RT $\mathfrak{T}$ such that $|\mathfrak{T}| < 1$. By Lemma 4.2.3 we obtain an infinite $\xrightarrow{\mathtt{i}}_{\mathcal{P}}$-RT $\mathfrak{T}'$ such that $|\mathfrak{T}'| \leq |\mathfrak{T}| < 1$, and thus, $\mathtt{AST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}})$ does not hold either. The argument is completely analogous for $\mathtt{PAST}(\xrightarrow{\mathtt{f}}_{\mathcal{P}})$, just reasoning about the expected derivation length.
For $\mathtt{SAST}(\xrightarrow{\mathtt{f}}_{\mathcal{P}})$ we have two cases: If there exists a single $\xrightarrow{\mathtt{f}}_{\mathcal{P}}$-RT $\mathfrak{T}$ such that $\mathrm{edl}(\mathfrak{T}) = \infty$, then we proceed as for $\mathtt{PAST}(\xrightarrow{\mathtt{f}}_{\mathcal{P}})$. Otherwise, there exists an infinite set $\{\mathfrak{T}_i \mid i \in \mathbb{N}\}$ of $\xrightarrow{\mathtt{f}}_{\mathcal{P}}$-RTs $\mathfrak{T}_i$ with the same start term such that $\sup\{\mathrm{edl}(\mathfrak{T}_i) \mid i \in \mathbb{N}\} = \infty$. For each of these RTs we apply Lemma 4.2.3 as before such that we obtain $\xrightarrow{\mathtt{i}}_{\mathcal{P}}$-RTs $\mathfrak{T}'_i$ with $\mathrm{edl}(\mathfrak{T}'_i) \geq \mathrm{edl}(\mathfrak{T}_i)$ for all $i \in \mathbb{N}$. Thus, $\{\mathfrak{T}'_i \mid i \in \mathbb{N}\}$ is an infinite set of $\xrightarrow{\mathtt{i}}_{\mathcal{P}}$-RTs with the same start term such that $\sup\{\mathrm{edl}(\mathfrak{T}'_i) \mid i \in \mathbb{N}\} = \infty$, which proves that $\mathtt{SAST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}})$ does not hold either. *Proof* *QED*

One may wonder whether we can remove the left-linearity requirement from Theorem 4.2.2, as in the non-probabilistic setting. However, it turns out that this is not possible.

Counterexample 4.2.4 (Left-Linearity Cannot Be Removed). Consider the PTRS $\mathcal{P}_1$ with the two rules

$$\mathsf{f}(x,x) \to \{^{1}\mathsf{f}(\mathsf{a},\mathsf{a})\} \qquad\qquad \mathsf{a} \to \{^{1/2}\mathsf{b},\, ^{1/2}\mathsf{c}\}$$

We do not have $\mathtt{AST}(\xrightarrow{\mathbf{f}}_{\mathcal{P}_1})$ (hence also neither $\mathtt{PAST}(\xrightarrow{\mathbf{f}}_{\mathcal{P}_1})$ nor $\mathtt{SAST}(\xrightarrow{\mathbf{f}}_{\mathcal{P}_1})$), since $\{^{1}\mathsf{f}(\mathsf{a},\mathsf{a})\} \overset{\mathbf{f}}{\Rightarrow}_{\mathcal{P}_1} \{^{1}\mathsf{f}(\mathsf{a},\mathsf{a})\} \overset{\mathbf{f}}{\Rightarrow}_{\mathcal{P}_1} \cdots$ is an infinite $\overset{\mathbf{f}}{\Rightarrow}_{\mathcal{P}_1}$-sequence that converges with probability 0. However, we have $\mathtt{SAST}(\xrightarrow{\mathbf{i}}_{\mathcal{P}_1})$ (and hence, $\mathtt{PAST}(\xrightarrow{\mathbf{i}}_{\mathcal{P}_1})$ and $\mathtt{AST}(\xrightarrow{\mathbf{i}}_{\mathcal{P}_1})$) since the corresponding innermost sequence has the form $\{^{1}\mathsf{f}(\mathsf{a},\mathsf{a})\} \overset{\mathbf{i}}{\Rightarrow}_{\mathcal{P}_1} \{^{1/2}\mathsf{f}(\mathsf{b},\mathsf{a}), ^{1/2}\mathsf{f}(\mathsf{c},\mathsf{a})\} \overset{\mathbf{i}}{\Rightarrow}_{\mathcal{P}_1} \{^{1/4}\mathsf{f}(\mathsf{b},\mathsf{b}), ^{1/4}\mathsf{f}(\mathsf{b},\mathsf{c}), ^{1/4}\mathsf{f}(\mathsf{c},\mathsf{b}), ^{1/4}\mathsf{f}(\mathsf{c},\mathsf{c})\} \overset{\mathbf{i}}{\Rightarrow}_{\mathcal{P}_1} \cdots$. Here, the last distribution contains the two normal forms $\mathsf{f}(\mathsf{b},\mathsf{c})$ and $\mathsf{f}(\mathsf{c},\mathsf{b})$ that did not occur in the previous rewrite sequence, so that the expected derivation length of this $\overset{\mathbf{i}}{\Rightarrow}_{\mathcal{P}_1}$-rewrite sequence is $2 + 3 \cdot \sum_{i=1}^{\infty}(1/2)^i = 5$. More generally, every $\mathsf{f}$ in a start term leads to at most 6 innermost rewrite steps in expectation, and every $\mathsf{a}$ in a start term leads to one step. Hence, $\mathrm{edh}_{\xrightarrow{\mathbf{i}}_{\mathcal{P}_1}}(t) \leq 6 \cdot |t|_{\mathsf{f}} + |t|_{\mathsf{a}}$ is finite for each $t \in \mathcal{T}$, where $|t|_{\mathsf{f}}$ and $|t|_{\mathsf{a}}$ are the numbers of $\mathsf{f}$ and $\mathsf{a}$ symbols in $t$. Note that adding the requirement of being non-erasing would not help to get rid of the left-linearity requirement, as shown by the PTRS $\mathcal{P}_2$ which results from $\mathcal{P}_1$ by replacing the $\mathsf{f}$-rule with $\mathsf{f}(x,x) \to \{^{1}\mathsf{d}(\mathsf{f}(\mathsf{a},\mathsf{a}),x)\}$.

The problem here is that although we rewrite both occurrences of $\mathsf{a}$ with the same rewrite rule, the two $\mathsf{a}$-symbols are replaced by two different terms (each with a probability $> 0$). This would be impossible in the non-probabilistic setting.

Next, one could try to adapt Theorem 4.1.4 to the probabilistic setting (when requiring linearity in addition). So one could investigate whether $\mathtt{PSN}(\xrightarrow{\mathbf{i}}_{\mathcal{P}})$ implies $\mathtt{PSN}(\xrightarrow{\mathbf{f}}_{\mathcal{P}})$ for PTRSs that are linear locally confluent overlay systems. A PTRS $\mathcal{P}$ is *locally confluent* if for all multi-distributions $\mu, \mu_1, \mu_2$ such that $\mu_1 \;{}_{\mathcal{P}}\overset{\mathbf{f}}{\leftarrow}\; \mu \overset{\mathbf{f}}{\Rightarrow}_{\mathcal{P}} \mu_2$, there exists a multi-distribution $\mu'$ such that $\mu_1 \overset{\mathbf{f}}{\Rightarrow}^{*}_{\mathcal{P}} \mu' \;{}_{\mathcal{P}}\overset{*}{\leftarrow}\overset{\mathbf{f}}{\leftarrow}\; \mu_2$, see [DCM18]. Note that in contrast to the non-probabilistic setting, there are non-overlapping PTRSs that are not locally confluent (e.g., the variant $\mathcal{P}_1'$ of $\mathcal{P}_1$ that consists of the rules $\mathsf{f}(x,x) \to \{^{1}\mathsf{d}\}$ and $\mathsf{a} \to \{^{1/2}\mathsf{b}, ^{1/2}\mathsf{c}\}$, since we have $\{^{1}\mathsf{d}\} \;{}_{\mathcal{P}_1'}\overset{\mathbf{f}}{\leftarrow}\; \{^{1}\mathsf{f}(\mathsf{a},\mathsf{a})\} \overset{\mathbf{f}}{\Rightarrow}_{\mathcal{P}_1'} \{^{1/2}\mathsf{f}(\mathsf{b},\mathsf{a}), ^{1/2}\mathsf{f}(\mathsf{c},\mathsf{a})\}$ and the two resulting multi-distributions are not joinable). Whether every linear, non-overlapping PTRS is locally confluent is an open problem, and thus, it is open whether an adaptation of Theorem 4.1.4 would subsume Theorem 4.2.2 as in the non-probabilistic setting.

In contrast to the proof of Theorem 4.1.2, the proof of Theorem 4.1.4 relies on the *minimality* property of termination of TRSs, see Theorem 3.2.55. As a reminder, whenever a term $t$ starts an infinite rewrite sequence w.r.t. some TRS $\mathcal{R}$, then there exists a *minimal* infinite rewrite sequence beginning with $t$, where one only reduces redexes whose proper subterms are terminating. However, we show in Section 5.3 that such minimal infinite sequences do not exist in the probabilistic setting in general.

It remains open whether one can also adapt Theorem 4.1.4 to the probabilistic setting (e.g., if one can replace non-overlappingness in Theorem 4.2.2 by the requirement of locally confluent overlay systems). There are two main difficulties when trying to adapt the proof of Theorem 4.1.4 to PTRSs. First, the minimality requirement cannot be imposed in the probabilistic setting, as discussed above. In the non-probabilistic setting, this requirement is needed to ensure that any subterm below a position that was reduced in the original (minimal) infinite rewrite sequence is terminating. Second, the original proof of Theorem 4.1.4 uses Newman's Lemma [New42], which states that local confluence implies confluence for strongly normalizing terms $t$, and thus it implies

that $t$ has a unique normal form. Local confluence and adaptations of the unique normal form property for the probabilistic setting have been studied in [DCM18, Fag22], who indicated that obtaining an analogous statement to Newman's Lemma for PTRSs that are AST would be very difficult. The reason is that one cannot use well-founded induction on the length of a rewrite sequence of a PTRS that is AST, since these rewrite sequences may be infinite.

We can also use Lemma 4.2.3 for expected complexity, leading to the following result.

Theorem 4.2.5 – From Innermost to Full Expected Complexity.
If a PTRS $\mathcal{P}$ is OR and RL (i.e., NO and linear), then

$$\mathrm{edc}_{\xrightarrow{\mathtt{f}}_{\mathcal{P}}} = \mathrm{edc}_{\xrightarrow{\mathtt{i}}_{\mathcal{P}}} \qquad \text{and} \qquad \mathrm{erc}_{\xrightarrow{\mathtt{f}}_{\mathcal{P}}} = \mathrm{erc}_{\xrightarrow{\mathtt{i}}_{\mathcal{P}}} .$$

*Proof.* Similar to the proof of Theorem 4.2.2, this is a direct consequence of Lemma 4.2.3. *Proof* | *QED*

While it is open whether an adaptation to locally confluent overlay systems would work for $\mathtt{PSN}(\xrightarrow{\mathtt{i}}_{\mathcal{P}})$, we can give a counterexample to show that one cannot weaken the requirement of NO to overlay systems for expected complexity, i.e., Theorem 4.2.5 does not hold for linear overlay systems in general. In contrast, in the non-probabilistic setting we have $\mathrm{rc}_{\xrightarrow{\mathtt{f}}_{\mathcal{R}}} = \mathrm{rc}_{\xrightarrow{\mathtt{i}}_{\mathcal{R}}}$ for all right-linear overlay systems (since right-linearity implies spareness), see Theorem 4.1.10.

Counterexample 4.2.6. Consider the PTRS $\mathcal{P}_3$ with the six rules

$$\begin{aligned} \mathsf{f}(x) &\to \{{}^{1}\!/_{2}\mathsf{g}(x), {}^{1}\!/_{2}\mathsf{h}(x)\} & \mathsf{d} &\to \{{}^{1}\mathsf{f}(\mathsf{a})\} \\ \mathsf{g}(\mathsf{b}) &\to \{{}^{1}\mathsf{f}(\mathsf{a})\} & \mathsf{a} &\to \{{}^{1}\mathsf{b}\} \\ \mathsf{h}(\mathsf{c}) &\to \{{}^{1}\mathsf{f}(\mathsf{a})\} & \mathsf{a} &\to \{{}^{1}\mathsf{c}\} \end{aligned}$$

$\mathcal{P}_3$ is a linear overlay system. Moreover, we have $\iota(\mathrm{erc}_{\xrightarrow{\mathtt{f}}_{\mathcal{P}_3}}) = \mathbf{Inf}$ due to the infinite $\xrightarrow{\mathtt{f}}_{\mathcal{P}_3}$-rewrite sequence $\{{}^{1}\mathsf{d}\} \xrightarrow{\mathtt{f}}_{\mathcal{P}_3} \{{}^{1}\mathsf{f}(\mathsf{a})\} \xrightarrow{\mathtt{f}}_{\mathcal{P}_3} \{{}^{1}\!/_{2}\mathsf{g}(\mathsf{a}), {}^{1}\!/_{2}\mathsf{h}(\mathsf{a})\} \xrightarrow{\mathtt{f}}_{\mathcal{P}_3} \{{}^{1}\!/_{2}\mathsf{g}(\mathsf{b}), {}^{1}\!/_{2}\mathsf{h}(\mathsf{c})\} \xrightarrow{\mathtt{f}}_{\mathcal{P}_3} \{{}^{1}\!/_{2}\mathsf{f}(\mathsf{a}), {}^{1}\!/_{2}\mathsf{f}(\mathsf{a})\}$ that converges with probability 0. But for $\xrightarrow{\mathtt{i}}_{\mathcal{P}_3}$-rewrite sequences we have to rewrite the argument $\mathsf{a}$ of $\mathsf{f}(\mathsf{a})$ first, leading to a normal form with a chance of $1/2$ after two steps. Hence, $\mathrm{erc}_{\xrightarrow{\mathtt{i}}_{\mathcal{P}_3}} \in \mathcal{O}(1)$ (i.e., $\iota(\mathrm{erc}_{\xrightarrow{\mathtt{i}}_{\mathcal{P}_3}}) = \mathbf{Pol}_0$). For expected derivational complexity, we get $\iota(\mathrm{edc}_{\xrightarrow{\mathtt{f}}_{\mathcal{P}_3}}) = \mathbf{Inf}$ and $\mathrm{edc}_{\xrightarrow{\mathtt{i}}_{\mathcal{P}_3}} \in \mathcal{O}(n)$ ($\iota(\mathrm{edc}_{\xrightarrow{\mathtt{i}}_{\mathcal{P}_3}}) = \mathbf{Pol}_1$) via similar arguments. Note that we have $\mathrm{edc}_{\xrightarrow{\mathtt{i}}_{\mathcal{P}_3}} \in \mathcal{O}(n)$ and not $\mathrm{edc}_{\xrightarrow{\mathtt{i}}_{\mathcal{P}_3}} \in \mathcal{O}(1)$, since for derivational complexity we allow start terms like $\mathsf{f}^n(\mathsf{a})$ where $n+1$ steps are needed to reach a multi-distribution that also contains normal forms.

Since $\mathrm{edc}_{\xrightarrow{\mathtt{i}}_{\mathcal{P}_3}}(n)$ is finite for all $n \in \mathbb{N}$, we have $\mathtt{SAST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}_3})$ (and thus, $\mathtt{PAST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}_3})$, $\mathtt{AST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}_3})$, and $\mathtt{wPSN}(\xrightarrow{\mathtt{f}}_{\mathcal{P}_3})$), but not $\mathtt{AST}(\xrightarrow{\mathtt{f}}_{\mathcal{P}_3})$. As $\mathcal{P}_3$ is also non-erasing, it shows that NO cannot be weakened to OS in Theorems 4.2.2 and 4.2.8 either. Clearly, $\mathcal{P}_3$ is not locally confluent. Similar as for $\mathtt{PSN}(\xrightarrow{\mathtt{i}}_{\mathcal{P}})$, it remains open whether there is an adaptation of Theorem 4.2.5 for locally confluent overlay systems.

In Section 4.3, we show that for left-linear, non-overlapping, and *spare* PTRSs $\mathcal{P}$ we still have $\mathrm{erc}_{\xrightarrow{\mathtt{f}}_{\mathcal{P}}} = \mathrm{erc}_{\xrightarrow{\mathtt{i}}_{\mathcal{P}}}$, see Theorem 4.3.6. That is, right-linearity can be weakened to spareness.

Note that Theorem 4.2.5 of course also holds for non-probabilistic PTRSs. Thus, we can use Theorem 4.2.5 to obtain the first result on the relation between full and

innermost derivational complexity in the non-probabilistic setting. For TRSs that are additionally non-erasing, the equality also follows from the *random descent* property of such TRSs, i.e., all maximal rewrite sequences that start with the same term have the same length [New42].

Corollary 4.2.7 – From Innermost to Full Complexity.
*If a TRS $\mathcal{R}$ is* OR *and* RL *(i.e.,* NO *and linear), then*

$$\mathrm{dc}_{\xrightarrow{\mathtt{f}}_{\mathcal{R}}} = \mathrm{dc}_{\xrightarrow{\mathtt{i}}_{\mathcal{R}}} .$$

*Proof.* By Theorem 4.2.5. *Proof QED*

### 4.2.2 From Weak to Full Normalization

Next, we investigate weak termination. Since $\mathtt{PSN}(\xrightarrow{\mathtt{i}}_{\mathcal{P}})$ implies $\mathtt{wPSN}(\xrightarrow{\mathtt{f}}_{\mathcal{P}})$, we essentially have the same problems as for $\mathtt{PSN}(\xrightarrow{\mathtt{i}}_{\mathcal{P}})$, i.e., in addition to non-overlappingness, we need linearity. This can be seen in Counterexamples 4.2.1 and 4.2.4, as for $\mathcal{P} \in \{\mathcal{P}_{\mathsf{dup}}, \mathcal{P}_1, \mathcal{P}_2\}$ we have $\mathtt{PSN}(\xrightarrow{\mathtt{i}}_{\mathcal{P}})$ (and hence $\mathtt{wPSN}(\xrightarrow{\mathtt{f}}_{\mathcal{P}})$) but not $\mathtt{PSN}(\xrightarrow{\mathtt{f}}_{\mathcal{P}})$, while $\mathcal{P}_{\mathsf{dup}}$ and $\mathcal{P}_2$ are non-overlapping and non-erasing, but not linear. Furthermore, we need non-erasingness as we did in the non-probabilistic setting for the same reasons, see Counterexample 4.1.5. Once again, there are no known properties that yield better results for outermost rewriting.

Theorem 4.2.8 – From $\mathtt{wPSN}(\xrightarrow{\mathtt{f}}_{\mathcal{P}})$ to $\mathtt{PSN}(\xrightarrow{\mathtt{f}}_{\mathcal{P}})$.
If a PTRS $\mathcal{P}$ is NO, linear, and NE, then

$$\mathtt{PSN}(\xrightarrow{\mathtt{f}}_{\mathcal{P}}) \iff \mathtt{wPSN}(\xrightarrow{\mathtt{f}}_{\mathcal{P}}).$$

To prove Theorem 4.2.8, we first establish the following lemma, which shows that for PTRSs that are NO, linear, and NE, all fully evaluated $\xrightarrow{\mathtt{f}}_{\mathcal{P}}$-RTs that start with the same term have exactly the same termination probability and expected derivation length. So in this case, the evaluation strategy is completely irrelevant.

Lemma 4.2.9 – From Weak to Full Rewriting.
*If a PTRS $\mathcal{P}$ is* NO*, linear, and* NE*, then for every two fully evaluated $\xrightarrow{\mathtt{f}}_{\mathcal{P}}$-RTs $\mathfrak{T}$ and $\tilde{\mathfrak{T}}$ starting with some $t \in \mathcal{T}$ we have*

1. $|\tilde{\mathfrak{T}}| = |\mathfrak{T}|$ (*Equal termination probability*)
2. $\mathrm{edl}(\tilde{\mathfrak{T}}) = \mathrm{edl}(\mathfrak{T})$ (*Equal expected derivation length*)

*Proof.* Let $\mathcal{P}$ be a PTRS that is non-overlapping, linear, and non-erasing, and let $\mathfrak{T}$ and $\tilde{\mathfrak{T}}$ be two different fully evaluated $\xrightarrow{\mathtt{f}}_{\mathcal{P}}$-RTs starting with $t \in \mathcal{T}$. In $\mathfrak{T}$ at least one rewrite step is performed at some node $x$ that is different from the rewrite step performed in $\tilde{\mathfrak{T}}$. We choose such a node $x$ of minimal depth, so that $\mathfrak{T}$ and $\tilde{\mathfrak{T}}$ perform the same rewrite steps at all nodes of smaller depth, and thus, $x$ is labeled with the same term in both trees. The core steps of the proof are the same as for the proof of Lemma 4.2.3. We iteratively push the rewrite steps that would be performed in $\tilde{\mathfrak{T}}$ at node $x$ to this node in $\mathfrak{T}$. As in the proof of Lemma 4.2.3, every node has only finitely many successors, so for every depth $H$, the constructed trees coincide with $\tilde{\mathfrak{T}}$ up to depth $H$ after finitely many iterations. Then, the limit of this construction is exactly *Proof*

$\tilde{\mathfrak{T}}$, which implies that $|\tilde{\mathfrak{T}}| \leq |\mathfrak{T}|$ and $\mathrm{edl}(\tilde{\mathfrak{T}}) \geq \mathrm{edl}(\mathfrak{T})$. By switching the roles of $\mathfrak{T}$ and $\tilde{\mathfrak{T}}$ and applying the same logic, we also get $|\mathfrak{T}| \leq |\tilde{\mathfrak{T}}|$ and $\mathrm{edl}(\mathfrak{T}) \geq \mathrm{edl}(\tilde{\mathfrak{T}})$. All in all, this then implies $|\mathfrak{T}| = |\tilde{\mathfrak{T}}|$ and $\mathrm{edl}(\mathfrak{T}) = \mathrm{edl}(\tilde{\mathfrak{T}})$.
We only have to slightly adjust the construction $\Phi(\cdot)$, and the rest of the proof remains the same.

**Step 1.1** Construction of $\Phi(\cdot)$

Let $\mathfrak{T}_x = \mathfrak{T}[xE^*]$ be an $\xrightarrow{\mathtt{f}}_{\mathcal{P}}$-RT that performs a rewrite step at position $\zeta$ at the root node $x$, i.e., $t_x^{\mathfrak{T}_x} \xrightarrow{\mathtt{f}}_{\mathcal{P},\zeta} \{{}^{p_{y_1}^{\mathfrak{T}_x}}t_{y_1}^{\mathfrak{T}_x}, \ldots, {}^{p_{y_k}^{\mathfrak{T}_x}}t_{y_k}^{\mathfrak{T}_x}\}$ using the rule $\bar{\ell} \to \{{}^{\bar{p}_1}\bar{r}_1, \ldots, {}^{\bar{p}_k}\bar{r}_k\}$, and the substitution $\bar{\sigma}$ such that $t_x^{\mathfrak{T}_x}|_\zeta = \bar{\ell}\bar{\sigma}$. Then $t_{y_j}^{\mathfrak{T}_x} = t_x^{\mathfrak{T}_x}[\bar{r}_j\bar{\sigma}]_\zeta$ for all $1 \leq j \leq k$. Furthermore, assume that $\tilde{\mathfrak{T}}_x = \tilde{\mathfrak{T}}[x\tilde{E}^*]$ rewrites at position $\tau$ (using a rule $\hat{\ell} \to \{{}^{\hat{p}_1}\hat{r}_1, \ldots, {}^{\hat{p}_h}\hat{r}_h\} \in \mathcal{P}$ and the substitution $\hat{\sigma}$) with $\tau \neq \zeta$. (Note that if $\tau = \zeta$, then by non-overlappingness the rewrite step would be the same.) Instead of applying the rewrite step at position $\zeta$ at the root $x$ we want to directly apply the rewrite step at position $\tau$.
The construction creates a new RT $\Phi(\mathfrak{T}_x) = (N', E')$ such that $|\Phi(\mathfrak{T}_x)| = |\mathfrak{T}_x|$, and that directly performs a rewrite step at position $\tau$ at the root of the tree, by pushing it from the original nodes in the tree $\mathfrak{T}_x$ to the root. We need the non-erasing property now, because $\tau$ may be above $\zeta$, which was not possible in the proof of Lemma 4.2.3. Again, since we are allowed to rewrite above $\tau$ in the original tree $\mathfrak{T}_x$, the actual position of the redex that was originally at position $\tau$ might change during the application of a rewrite step. Hence, we recursively define the position $\varphi_\tau(v)$ that contains precisely this redex for each node $v$ in $\mathfrak{T}_x$ until we rewrite at this position. Compared to the proof of Lemma 4.2.3, since the rules in $\mathcal{P}$ are non-erasing, we only have $\varphi_\tau(v) \in \mathbb{N}^*$ or $\varphi_\tau(v) = \bot$. The option $\varphi_\tau(v) = \top$ is not possible anymore. Initially, we have $\varphi_\tau(x) = \tau$. Whenever we have defined $\varphi_\tau(v)$ for some node $v$, and we have $t_v^{\mathfrak{T}_x} \xrightarrow{\mathtt{f}}_{\mathcal{P}} \{{}^{p_{w_1}^{\mathfrak{T}_x}}t_{w_1}^{\mathfrak{T}_x}, \ldots, {}^{p_{w_m}^{\mathfrak{T}_x}}t_{w_m}^{\mathfrak{T}_x}\}$ for the direct successors $vE = \{w_1, \ldots, w_m\}$, using the rule $\ell \to \{{}^{p_1}r_1, \ldots, {}^{p_m}r_m\}$, the substitution $\sigma$, and position $\pi$, we do the following: If $\varphi_\tau(v) = \pi$, then we set $\varphi_\tau(w_j) = \bot$ for all $1 \leq j \leq m$ to indicate that we have rewritten the redex. If we have $\varphi_\tau(v)\bot\pi$, meaning that the rewrite step takes place parallel to $\varphi_\tau(v)$, then we set $\varphi_\tau(w_j) = \varphi_\tau(v)$ for all $1 \leq j \leq m$, as the position did not change. If we have $\varphi_\tau(v) < \pi$, then we set $\varphi_\tau(w_j) = \varphi_\tau(v)$ for all $1 \leq j \leq m$ as well, as the position did not change either. If we have $\pi < \varphi_\tau(v)$, then there exists a $\chi \in \mathbb{N}^+$ such that $\pi.\chi = \varphi_\tau(v)$. Since the rules of $\mathcal{P}$ are non-overlapping, the redex must be completely "inside" the used substitution $\sigma$, and we can find a position $\alpha_q$ of a variable $q$ in $\ell$ and another position $\beta$ such that $\chi = \alpha_q.\beta$. Furthermore, since the rule is linear and non-erasing, $q$ occurs once in $\ell$ and once in $r_j$ for all $1 \leq j \leq m$. Let $\rho_q^j$ be the position of $q$ in $r_j$. Here, we set $\varphi_\tau(w_j) = \pi.\rho_q^j.\beta$. Finally, if $\varphi_\tau(v) = \bot$, then we set $\varphi_\tau(w_j) = \varphi_\tau(v) = \bot$ for all $1 \leq j \leq m$ as well.
Again, we now define the $\xrightarrow{\mathtt{f}}_{\mathcal{P}}$-RT $\Phi(\mathfrak{T}_x)$ that starts with $t_x^{\mathfrak{T}_x}$ such that $|\Phi(\mathfrak{T}_x)| = |\mathfrak{T}_x|$, and that directly performs the rewrite step $t_x^{\mathfrak{T}_x} \xrightarrow{\mathtt{f}}_{\mathcal{P},\tau} \{{}^{\hat{p}_1}t_{1.x}^{\Phi(\mathfrak{T}_x)}, \ldots, {}^{\hat{p}_h}t_{h.x}^{\Phi(\mathfrak{T}_x)}\}$, with the rule $\hat{\ell} \to \{{}^{\hat{p}_1}\hat{r}_1, \ldots, {}^{\hat{p}_h}\hat{r}_h\} \in \mathcal{P}$, the substitution $\hat{\sigma}$, and the position $\tau$, at the new root $\hat{v}$. Here, we have $t_x^{\mathfrak{T}_x}|_\tau = \hat{\ell}\hat{\sigma}$. Let $Z$ be the set of all nodes $v$ such that $\varphi_\tau(v) \neq \bot$. For each of these nodes $z \in Z$ and each $1 \leq e \leq h$, we create a new node $e.z \in N'$ with edges as in $\mathfrak{T}_x$ for the nodes in $Z$. Furthermore, we add the edges from the new root $\hat{v}$ to the nodes $e.x$ for all $1 \leq e \leq h$. Remember that $x$ was the root in the tree $\mathfrak{T}_x$ and has to be contained in $Z$. We define the labeling of the nodes in $\Phi(\mathfrak{T}_x)$ as

follows for all nodes $z$ in $Z$:

(T-1) $t^{\Phi(\mathfrak{T}_x)}_{e.z} = t^{\mathfrak{T}_x}_z[\hat{r}_e\delta]_{\varphi_\tau(z)}$ for the substitution $\delta$ such that $t^{\mathfrak{T}_x}_z|_{\varphi_\tau(z)} = \hat{\ell}\delta$

(T-2) $p^{\Phi(\mathfrak{T}_x)}_{e.z} = p^{\mathfrak{T}_x}_z \cdot \hat{p}_e$

Now, for a leaf $e.z' \in N'$ either $z' \in N$ is also a leaf or we rewrite at the position $\varphi_\tau(z')$ in node $z'$ in $\mathfrak{T}_x$. If we rewrite $t^{\mathfrak{T}_x}_{z'} \xrightarrow{\mathrm{f}}_{\mathcal{P},\varphi_\tau(z')} \{{}^{p^{\mathfrak{T}_x}_{w'_1}}t^{\mathfrak{T}_x}_{w'_1}, \ldots, {}^{p^{\mathfrak{T}_x}_{w'_h}}t^{\mathfrak{T}_x}_{w'_h}\}$, then we have $t^{\mathfrak{T}_x}_{w'_e} = t^{\mathfrak{T}_x}_{z'}[\hat{r}_e\delta]_{\varphi_\tau(z')} \overset{\text{(T-1)}}{=} t^{\Phi(\mathfrak{T}_x)}_{e.z'}$ for the substitution $\delta$ such that $t^{\mathfrak{T}_x}_{z'}|_{\varphi_\tau(z')} = \hat{\ell}\delta$ and $p^{\mathfrak{T}_x}_{w'_e} \overset{\text{(T-2)}}{=} p^{\mathfrak{T}_x}_{z'} \cdot \hat{p}_e = p^{\Phi(\mathfrak{T}_x)}_{e.z'}$. Thus, we can copy the rest of this subtree of $\mathfrak{T}_x$ to our newly generated tree $\Phi(\mathfrak{T}_x)$. Again, we just had to define how to construct $\Phi(\mathfrak{T}_x)$ for the part of the tree before we reach the nodes $v$ with $\varphi_\tau(v) = \bot$ in $\mathfrak{T}_x$. As in the proof of Lemma 4.2.3, we obtain $|\Phi(\mathfrak{T}_x)| = |\mathfrak{T}_x|$. Moreover, $\Phi(\mathfrak{T}_x)$ is again fully evaluated. We only need to show that $\Phi(\mathfrak{T}_x)$ is a valid $\xrightarrow{\mathrm{f}}_{\mathcal{P}}$-RT, i.e., that $t^{\Phi(\mathfrak{T}_x)}_{e.z} \xrightarrow{\mathrm{f}}_{\mathcal{P}} \{{}^{p^{\Phi(\mathfrak{T}_x)}_{e.w_1}}t^{\Phi(\mathfrak{T}_x)}_{e.w_1}, \ldots, {}^{p^{\Phi(\mathfrak{T}_x)}_{e.w_m}}t^{\Phi(\mathfrak{T}_x)}_{e.w_m}\}$ for all $z \in Z$ as in the proof of Lemma 4.2.3.

**Step 1.3** Show that $\Phi(\mathfrak{T}_x)$ is indeed a valid RT.

Let $z \in Z$. We distinguish three different cases for a rewrite step at node $z$ of $\mathfrak{T}_x$:

(A) We rewrite at a position parallel to $\varphi_\tau(z)$.

(B) We rewrite at a position above $\varphi_\tau(z)$.

(C) We rewrite at a position below $\varphi_\tau(z)$.

The cases (A) and (B) are analogous to the proof of Lemma 4.2.3, but the option $\varphi_\tau(z) = \top$ in Case (A) no longer occurs, and in Case (B) the redex can no longer be erased. We only need to look at the new Case (C).

**(C)** If we have $t^{\mathfrak{T}_x}_z \xrightarrow{\mathrm{f}}_{\mathcal{P}} \{{}^{p^{\mathfrak{T}_x}_z \cdot p_1}t^{\mathfrak{T}_x}_{w_1}, \ldots, {}^{p^{\mathfrak{T}_x}_z \cdot p_m}t^{\mathfrak{T}_x}_{w_m}\}$, then there is a rule $\ell \to \{{}^{p_1}r_1, \ldots, {}^{p_m}r_m\} \in \mathcal{P}$, a substitution $\sigma$, and a position $\pi$ with $t^{\mathfrak{T}_x}_z|_\pi = \ell\sigma$. Then $t^{\mathfrak{T}_x}_{w_j} = t^{\mathfrak{T}_x}_z[r_j\sigma]_\pi$ for all $1 \le j \le m$. Additionally, we assume that $\varphi_\tau(z) < \pi$, and thus there exists a $\chi \in \mathbb{N}^+$ such that $\pi = \varphi_\tau(z).\chi$. By (T-1), we get $t^{\mathfrak{T}_x}_z[\hat{r}_e\delta]_{\varphi_\tau(z)} = t^{\Phi(\mathfrak{T}_x)}_{e.z}$ for the substitution $\delta$ such that $t^{\mathfrak{T}_x}_z|_{\varphi_\tau(z)} = \hat{\ell}\delta$. Since the rules of $\mathcal{P}$ are non-overlapping, including the rule $\hat{\ell} \to \{{}^{\hat{p}_1}\hat{r}_1, \ldots, {}^{\hat{p}_h}\hat{r}_h\}$ that we use at the root of $\Phi(\mathfrak{T}_x)$, the redex for the current rewrite step must be completely "inside" the substitution $\delta$, and we can find a variable position $\alpha_q$ of a variable $q$ in $\hat{\ell}$ and another position $\beta$ such that $\chi = \alpha_q.\beta$. Furthermore, since the rule is linear and non-erasing, $q$ occurs exactly once in $\hat{\ell}$ and exactly once in $\hat{r}_j$ for all $1 \le j \le h$. Let $\rho^j_q$ be the position of $q$ in $\hat{r}_j$. Since $q$ is at position $\rho^e_q$ in $\hat{r}_e$, the subterm $\delta(q)$ (and hence the redex $\ell\sigma$ at position $\beta$ inside $\delta(q)$) has moved to position $\rho^e_q.\beta$ within $\hat{r}_e\delta$. Let $\pi' = \varphi_\tau(z).\rho^e_q.\beta$. We can rewrite $t^{\Phi(\mathfrak{T}_x)}_{e.z}$ using the same rule and the same substitution $\sigma$, but now at position $\pi'$, as we have $t^{\Phi(\mathfrak{T}_x)}_{e.z}|_{\pi'} = (\hat{r}_e\delta)|_{\rho^e_q.\beta} = \delta(q)|_\beta = \ell\sigma$.

It remains to show that $t^{\Phi(\mathfrak{T}_x)}_{e.w_j} = t^{\Phi(\mathfrak{T}_x)}_{e.z}[r_j\sigma]_{\pi'}$ for all $1 \le j \le m$, i.e., that the labeling we defined for $\Phi(\mathfrak{T}_x)$ corresponds to this rewrite step. Let $1 \le j \le m$. We have $t^{\Phi(\mathfrak{T}_x)}_{e.z}[r_j\sigma]_{\pi'} = t^{\mathfrak{T}_x}_z[\hat{r}_e\delta]_{\varphi_\tau(z)}[r_j\sigma]_{\varphi_\tau(z).\rho^e_q.\beta} = t^{\mathfrak{T}_x}_z[\hat{r}_e\delta']_{\varphi_\tau(z)} = t^{\Phi(\mathfrak{T}_x)}_{e.w_j}$ for the substitution $\delta'$ with $\delta'(q) = \delta(q)[r_j\sigma]_\beta$ and $\delta'(q') = \delta(q')$ for all other variables $q' \neq q$. Here, $\hat{r}_e\delta' = (\hat{r}_e\delta)[r_j\sigma]_{\rho^e_q.\beta}$ since $q$ occurs exactly once in $\hat{r}_e$ at position $\rho^e_q$. With this

new substitution, we get $t^{\mathfrak{T}_x}_{w_j}|_{\varphi_\tau(w_j)} = t^{\mathfrak{T}_x}_{z}[r_j\sigma]_{\varphi_\tau(z).\alpha_q.\beta}|_{\varphi_\tau(z)} = t^{\mathfrak{T}_x}_{z}|_{\varphi_\tau(z)}[r_j\sigma]_{\alpha_q.\beta} = (\hat{\ell}\delta)[r_j\sigma]_{\alpha_q.\beta} = \hat{\ell}\delta'$, confirming that the labeling of $e.w_j$ via ($T$-1) uses the correctly updated substitution $\delta'$. Finally, note that the probabilities of the labeling are correct, as we are using the same rule with the same probabilities in both trees.

*QED*

With Lemma 4.2.9 we can now prove Theorem 4.2.8.

*Proof of Theorem 4.2.8.* We only need to prove the "$\Longleftarrow$" direction. Let us first consider $\mathtt{AST}(\xrightarrow{\mathbf{f}}_{\mathcal{P}})$. Assume that $\mathcal{P}$ is NO, linear, and NE, and that $\mathtt{AST}(\xrightarrow{\mathbf{f}}_{\mathcal{P}})$ does not hold. Then, there exists an infinite fully evaluated $\xrightarrow{\mathbf{f}}_{\mathcal{P}}$-RT $\mathfrak{T}$ such that $|\mathfrak{T}| < 1$ (remember that every $\xrightarrow{\mathbf{f}}_{\mathcal{P}}$-RT can be extended to a fully evaluated $\xrightarrow{\mathbf{f}}_{\mathcal{P}}$-RT without increasing its convergence probability, see Lemma 3.3.38). By Lemma 4.2.9 we know that then for every fully evaluated $\xrightarrow{\mathbf{f}}_{\mathcal{P}}$-RT $\mathfrak{T}'$ that starts with the same term as $\mathfrak{T}$ we have $|\mathfrak{T}'| = |\mathfrak{T}| < 1$. Thus, we do not have $\mathtt{wAST}(\xrightarrow{\mathbf{f}}_{\mathcal{P}})$ either. The argument can be lifted to PAST as in the proof of Theorem 4.2.2.

*Proof* *QED*

We can use Lemma 4.2.9 for expected complexity to relate full rewriting to every other evaluation strategy.

Theorem 4.2.10 – From Any Strategy to Full Expected Complexity.
If a PTRS $\mathcal{P}$ is NO, linear, and NE, then for all $s \in \{\mathbf{f}, \mathbf{i}, \mathbf{o}, \mathbf{li}\}$ we have

$$\mathrm{edc}_{\xrightarrow{\mathbf{f}}_{\mathcal{P}}} = \mathrm{edc}_{\xrightarrow{s}_{\mathcal{P}}} \qquad \text{and} \qquad \mathrm{erc}_{\xrightarrow{\mathbf{f}}_{\mathcal{P}}} = \mathrm{erc}_{\xrightarrow{s}_{\mathcal{P}}}\,.$$

*Proof.* This is a direct consequence of Lemma 4.2.9.

*Proof* *QED*

### 4.2.3 From Leftmost-Innermost to Innermost Rewriting

Finally, we look at leftmost-innermost rewriting as an example for an evaluation strategy that uses an ordering for parallel redexes. In contrast to the non-probabilistic setting, it turns out that $\mathtt{PSN}(\xrightarrow{\mathbf{li}}_{\mathcal{P}})$ and $\mathtt{PSN}(\xrightarrow{\mathbf{i}}_{\mathcal{P}})$ are not equivalent in general. The following counterexample is similar to Counterexample 4.2.4, which illustrated that $\mathtt{PSN}(\xrightarrow{\mathbf{f}}_{\mathcal{P}})$ and $\mathtt{PSN}(\xrightarrow{\mathbf{i}}_{\mathcal{P}})$ are not equivalent without left-linearity.

Counterexample 4.2.11. Consider the PTRS $\mathcal{P}_4$ with the five rules

$$\begin{aligned} \mathsf{a} &\to \{^{1}\mathsf{c}_1\} & \mathsf{b} &\to \{^{1/2}\mathsf{d}_1, {}^{1/2}\mathsf{d}_2\} \\ \mathsf{a} &\to \{^{1}\mathsf{c}_2\} & \mathsf{f}(\mathsf{c}_1,\mathsf{d}_1) &\to \{^{1}\mathsf{f}(\mathsf{a},\mathsf{b})\} \\ & & \mathsf{f}(\mathsf{c}_2,\mathsf{d}_2) &\to \{^{1}\mathsf{f}(\mathsf{a},\mathsf{b})\} \end{aligned}$$

We do not have $\mathtt{AST}(\xrightarrow{\mathbf{i}}_{\mathcal{P}_4})$ (hence also neither $\mathtt{PAST}(\xrightarrow{\mathbf{i}}_{\mathcal{P}_4})$ nor $\mathtt{SAST}(\xrightarrow{\mathbf{i}}_{\mathcal{P}_4})$), since there exists the infinite rewrite sequence $\{^{1}\mathsf{f}(\mathsf{a},\mathsf{b})\} \xrightarrow{\mathbf{i}}_{\mathcal{P}_4} \{^{1/2}\mathsf{f}(\mathsf{a},\mathsf{d}_1), {}^{1/2}\mathsf{f}(\mathsf{a},\mathsf{d}_2)\} \xrightarrow{\mathbf{i}}_{\mathcal{P}_4} \{^{1/2}\mathsf{f}(\mathsf{c}_1,\mathsf{d}_1), {}^{1/2}\mathsf{f}(\mathsf{c}_2,\mathsf{d}_2)\} \xrightarrow{\mathbf{i}}_{\mathcal{P}_4} \{^{1/2}\mathsf{f}(\mathsf{a},\mathsf{b}), {}^{1/2}\mathsf{f}(\mathsf{a},\mathsf{b})\} \xrightarrow{\mathbf{i}}_{\mathcal{P}_4} \cdots$, which converges with probability 0. It first "splits" the term $\mathsf{f}(\mathsf{a},\mathsf{b})$ with the b-rule, and then applies one of the two different a-rules to each of the resulting terms. In contrast, when applying a leftmost-innermost evaluation strategy, we have to decide which a-rule to use before we split the term with the b-rule. For example, we have $\{^{1}\mathsf{f}(\mathsf{a},\mathsf{b})\} \xrightarrow{\mathbf{li}}_{\mathcal{P}_4} \{^{1}\mathsf{f}(\mathsf{c}_1,\mathsf{b})\} \xrightarrow{\mathbf{li}}_{\mathcal{P}_4} \{^{1/2}\mathsf{f}(\mathsf{c}_1,\mathsf{d}_1), {}^{1/2}\mathsf{f}(\mathsf{c}_1,\mathsf{d}_2)\}$. Here, the second term $\mathsf{f}(\mathsf{c}_1,\mathsf{d}_2)$ is a normal form. Since all leftmost-innermost rewrite sequences keep on adding such normal forms after a certain number of steps for each start term, we have $\mathtt{SAST}(\xrightarrow{\mathbf{li}}_{\mathcal{P}_4})$ (and hence, $\mathtt{PAST}(\xrightarrow{\mathbf{li}}_{\mathcal{P}_4})$ and $\mathtt{AST}(\xrightarrow{\mathbf{li}}_{\mathcal{P}_4})$).

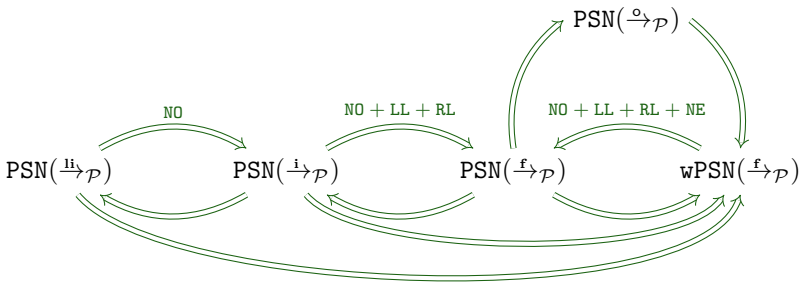



Figure 4.7: Relations between evaluation strategies w.r.t. probabilistic termination.

The counterexample above can easily be adapted to other positional evaluation strategies, e.g., *rightmost*-innermost rewriting.

However, $\mathtt{PSN}(\xrightarrow{\mathtt{li}}_{\mathcal{P}})$ and $\mathtt{PSN}(\xrightarrow{\mathtt{i}}_{\mathcal{P}})$ are again equivalent for non-overlapping PTRSs $\mathcal{P}$. For such PTRSs, at most one rule can be used to rewrite at a given position, which prevents the problem illustrated in Counterexample 4.2.11.

Theorem 4.2.12 – From $\mathtt{PSN}(\xrightarrow{\mathtt{li}}_{\mathcal{P}})$ to $\mathtt{PSN}(\xrightarrow{\mathtt{i}}_{\mathcal{P}})$.
If a PTRS $\mathcal{P}$ is NO, then

$$\mathtt{PSN}(\xrightarrow{\mathtt{i}}_{\mathcal{P}}) \iff \mathtt{PSN}(\xrightarrow{\mathtt{li}}_{\mathcal{P}}).$$

For the proof of Theorem 4.2.12, we use the following lemma, which immediately implies Theorem 4.2.12.

Lemma 4.2.13 – From Leftmost-Innermost to Innermost Rewriting.
*If a PTRS $\mathcal{P}$ is* NO*, then for every $\xrightarrow{\mathtt{i}}_{\mathcal{P}}$-RT $\mathfrak{T}$ starting with some $t \in \mathcal{T}$ there exists a $\xrightarrow{\mathtt{li}}_{\mathcal{P}}$-RT $\mathfrak{T}'$ starting with $t$ such that*

1. $|\mathfrak{T}'| \leq |\mathfrak{T}|$ (*Lower bound on the termination probability*)
2. $\operatorname{edl}(\mathfrak{T}') \geq \operatorname{edl}(\mathfrak{T})$ (*Upper bound on the expected derivation length*)

*Proof.* The idea and the construction of this proof are completely analogous to those of Lemma 4.2.3. We iteratively move leftmost-innermost rewrite steps to a higher position in the innermost RT. Hence, the resulting tree is a $\xrightarrow{\mathtt{li}}_{\mathcal{P}}$-RT.
The construction of $\Phi(\,\cdot\,)$ is also analogous to the one in Lemma 4.2.3. The only difference to the proof of Lemma 4.2.3 is that the original tree $\mathfrak{T}_x$ is already an $\xrightarrow{\mathtt{i}}_{\mathcal{P}}$-RT. This means that during our construction only Case (A) can occur, as we cannot rewrite above a redex in an $\xrightarrow{\mathtt{i}}_{\mathcal{P}}$-RT. This case does not need any property of $\mathcal{P}$, but we still need non-overlappingness to ensure that whenever $\mathfrak{T}_x$ rewrites the redex at position $\varphi_\tau(z')$, it uses the same rule as the new root (otherwise, two different rules for the same redex lead to different terms as in Counterexample 4.2.11). *Proof* *QED*

The relations between the different notions of AST, PAST, and SAST of PTRSs for different evaluation strategies (given in Theorems 4.2.2, 4.2.8 and 4.2.12) are summarized in Figure 4.7.

For expected complexity, we obtain the following results from Lemma 4.2.13.

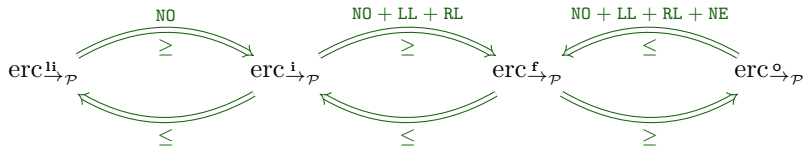



Figure 4.8: Results regarding expected runtime complexity.

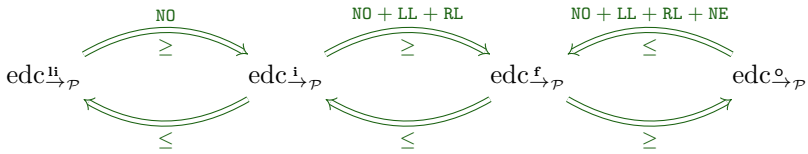



Figure 4.9: Results regarding expected derivational complexity.

Theorem 4.2.14 – From Leftmost-Innermost to Innermost Expected Complexity.
If a PTRS $\mathcal{P}$ is NO, then

$$\mathrm{edc}^{\mathtt{li}}_{\to\mathcal{P}} = \mathrm{edc}^{\mathtt{i}}_{\to\mathcal{P}} \qquad \text{and} \qquad \mathrm{erc}^{\mathtt{li}}_{\to\mathcal{P}} = \mathrm{erc}^{\mathtt{i}}_{\to\mathcal{P}} \,.$$

*Proof.* This is a direct consequence of Lemma 4.2.13. *Proof QED*

As shown by $\mathcal{P}_4$ from Counterexample 4.2.11, again we cannot change the requirement of non-overlapping PTRSs to overlay systems, since $\mathcal{P}_4$ is an overlay system where $\mathtt{PSN}(\xrightarrow{\mathtt{li}}_{\mathcal{P}_4})$ holds, but $\mathtt{PSN}(\xrightarrow{\mathtt{i}}_{\mathcal{P}_4})$ does not. The relations between the different expected complexities of PTRSs (given in Theorems 4.2.5, 4.2.10 and 4.2.14) are summarized in Figure 4.8 and Figure 4.9. Here, arrows " $\Longrightarrow$ " stand for "$\geq$".

For the non-probabilistic setting, Theorem 4.2.14 implies the following corollary.

Corollary 4.2.15 – From Leftmost-Innermost to Innermost Complexity.
*If a TRS $\mathcal{R}$ is* NO*, then*

$$\mathrm{dc}^{\mathtt{li}}_{\to\mathcal{R}} = \mathrm{dc}^{\mathtt{i}}_{\to\mathcal{R}} \qquad \textit{and} \qquad \mathrm{rc}^{\mathtt{li}}_{\to\mathcal{R}} = \mathrm{rc}^{\mathtt{i}}_{\to\mathcal{R}} \,.$$

*Proof.* This is a direct consequence of Theorem 4.2.14. *Proof QED*

SECTION 4.3

## Improving Applicability

In this section, we present an idea to improve the applicability of Theorem 4.2.2, which relates $\mathtt{PSN}(\xrightarrow{\mathtt{f}}_{\mathcal{P}})$ and $\mathtt{PSN}(\xrightarrow{\mathtt{i}}_{\mathcal{P}})$ — the most interesting theorem in practice. Recall that according to Theorem 4.2.2, $\mathtt{PSN}(\xrightarrow{}_{\mathcal{P}})$ implies $\mathtt{PSN}(\xrightarrow{\mathtt{f}}_{\mathcal{P}})$ for PTRSs $\mathcal{P}$ that are NO, LL, and RL. While in [KFG24] there was the idea to use a new rewrite relation, called *simultaneous rewriting*, to remove the necessity of left-linear rewrite rules, this idea was shown to be incorrect in [KG25]. Instead of left-linearity, we target right-linearity, and we show that it can be weakened to spareness if one only considers rewrite sequences that start with basic terms, as in the definition of runtime complexity.

In the non-probabilistic setting, runtime complexity is easier to analyze than

derivational complexity because of the restriction to basic start terms. In particular, this restriction also allows us to use notions like spareness such that full runtime complexity can be analyzed via innermost runtime complexity, see Theorem 4.1.10. Similarly, in the probabilistic setting we can also require spareness instead of right-linearity, if we only consider basic start terms.

Definition 4.3.1 – Spareness for PTRSs.

*Let $\mathcal{P}$ be a PTRS. A rewrite step using the rule $\ell \to \mu \in \mathcal{P}$ and the substitution $\sigma$ is called spare if $\sigma(x) \in \mathtt{NF}_{\mathcal{P}}$ for every $x \in \mathcal{V}$ that occurs more than once in some $r \in \mathrm{Supp}(\mu)$. A $\overset{\mathtt{f}}{\Rightarrow}_{\mathcal{P}}$-rewrite sequence is spare if each rewrite step used in it is spare. $\mathcal{P}$ is spare ($\mathtt{SP}$) if each $\overset{\mathtt{f}}{\Rightarrow}_{\mathcal{P}}$-rewrite sequence that starts with $t$ for a basic term $t \in \mathcal{T}_{\mathcal{B}}$ is spare.*

Example 4.3.2. Consider the PTRS $\mathcal{P}_5$ with the two rules

$$\mathsf{g} \to \{^{3/4}\mathsf{d}(\mathsf{0}), ^{1/4}\mathsf{g}\} \qquad\qquad \mathsf{d}(x) \to \{^{1}\mathsf{c}(x, x)\}$$

It is similar to the PTRS $\mathcal{P}_{\mathtt{dup}}$ from Counterexample 4.2.1, but we exchanged the symbols $\mathsf{g}$ and $\mathsf{0}$ in the right-hand side of the $\mathsf{g}$-rule. $\mathcal{P}_5$ is orthogonal but duplicating due to the $\mathsf{d}$-rule. However, in any rewrite sequence that starts with a basic term $t$ we can only duplicate terms with no defined symbols. Hence, $\mathcal{P}_5$ is spare.

If a PTRS $\mathcal{P}$ is spare, and we start with a basic term, then we will only duplicate normal forms with a duplicating rule. This means that duplicating rules cannot duplicate redexes, so the duplication neither increases the (expected) runtime nor decreases the probability of termination. This leads to the following theorem, which weakens the requirement of $\mathtt{RL}$ to $\mathtt{SP}$ in Theorem 4.2.2, where "*starting in $\mathcal{T}_{\mathcal{B}}$*" means that one only considers rewrite sequences that start with $t \in \mathcal{T}_{\mathcal{B}}$, where $\mathcal{T}_{\mathcal{B}}$ is again the set of all basic terms w.r.t. $\mathcal{P}$.

Theorem 4.3.3 – From $\mathtt{PSN}(\overset{\mathtt{i}}{\to}_{\mathcal{P}})$ Starting in $\mathcal{T}_{\mathcal{B}}$ to $\mathtt{PSN}(\overset{\mathtt{f}}{\to}_{\mathcal{P}})$ Starting in $\mathcal{T}_{\mathcal{B}}$.

If a PTRS $\mathcal{P}$ is $\mathtt{OR}$ and $\mathtt{SP}$, then

$$\mathtt{PSN}(\overset{\mathtt{f}}{\to}_{\mathcal{P}}) \text{ starting in } \mathcal{T}_{\mathcal{B}} \iff \mathtt{PSN}(\overset{\mathtt{i}}{\to}_{\mathcal{P}}) \text{ starting in } \mathcal{T}_{\mathcal{B}}.$$

For the proof of Theorem 4.3.3 we use the following lemma.

Lemma 4.3.4 – From Innermost to Full Rewriting Starting in $\mathcal{T}_{\mathcal{B}}$.

*If a PTRS $\mathcal{P}$ is $\mathtt{OR}$ and $\mathtt{SP}$, then for every $\overset{\mathtt{f}}{\to}_{\mathcal{P}}$-RT $\mathfrak{T}$ that starts with a basic term $t \in \mathcal{T}_{\mathcal{B}}$ there exists an $\overset{\mathtt{i}}{\to}_{\mathcal{P}}$-RT $\mathfrak{T}'$ that starts with $t$ such that*

1. $|\mathfrak{T}'| \leq |\mathfrak{T}|$ (*Lower bound on the termination probability*)
2. $\mathrm{edl}(\mathfrak{T}') \geq \mathrm{edl}(\mathfrak{T})$ (*Upper bound on the expected derivation length*)

*Proof.* The proof is completely analogous to that of Lemma 4.2.3. We iteratively move the innermost rewrite steps to a higher position using the construction $\Phi(\,\cdot\,)$. Note that since $\mathcal{P}$ is $\mathtt{OR}$ and $\mathtt{SP}$ (hence non-overlapping, left-linear, and spare), in the construction of $\Phi(\,\cdot\,)$ and in the proof of Case (B), if the innermost redex is below a redex $\ell\sigma$ that is reduced next via a rule $\ell \to \{^{p_1}r_1, \ldots, ^{p_m}r_m\}$, then the innermost redex is completely "inside" the used substitution $\sigma$, and it corresponds to a variable $q$

*Proof*

which occurs only once in $\ell$ and at most once in $r_j$ for all $1 \leq j \leq m$, due to spareness of $\mathcal{P}$ and the fact that we started with a basic term. Hence, we can use the same construction as in Lemma 4.2.3. *QED*

With Lemma 4.3.4 we can now prove Theorem 4.3.3.

*Proof of Theorem 4.3.3.* The direction "$\Longrightarrow$" holds since every $\xrightarrow{\mathtt{i}}_{\mathcal{P}}$-RT is also an $\xrightarrow{\mathtt{f}}_{\mathcal{P}}$-RT. The direction "$\Longleftarrow$" follows from Lemma 4.3.4 as in the proof of Theorem 4.2.2, where we only consider RTs that start with basic terms. *Proof* *QED*

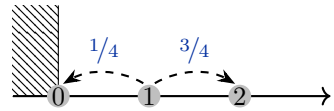
The requirement of basic start terms is a real restriction for orthogonal PTRSs, i.e., there exist very simple orthogonal PTRSs $\mathcal{P}$ where $\mathtt{PSN}(\xrightarrow{\mathtt{f}}_{\mathcal{P}})$ starting in $\mathcal{T}_{\mathcal{B}}$ holds, but not $\mathtt{PSN}(\xrightarrow{\mathtt{f}}_{\mathcal{P}})$ in general. One can see this already in the non-probabilistic setting for the TRS $\mathcal{R}$ consisting of the two rules $\mathsf{f}(\mathsf{g}(\mathsf{a})) \to \mathsf{f}(\mathsf{g}(\mathsf{a}))$ and $\mathsf{g}(\mathsf{b}) \to \mathsf{b}$, where we do not have $\mathtt{SN}(\xrightarrow{\mathtt{f}}_{\mathcal{R}})$, but $\mathtt{SN}(\xrightarrow{\mathtt{f}}_{\mathcal{R}})$ starting in $\mathcal{T}_{\mathcal{B}}$.

The next example shows that Theorem 4.3.3 does not hold for arbitrary start terms.

Counterexample 4.3.5. Consider the PTRS $\mathcal{P}_6$ with the two rules depicted in Figure 4.10.

$$\begin{aligned}\mathsf{f}(\mathsf{f}(\mathsf{a})) &\to \{^{3/4}\mathsf{d}(\mathsf{f}(\mathsf{f}(\mathsf{a}))), {}^{1/4}\mathsf{0}\}\\ \mathsf{d}(x) &\to \{^{1}\mathsf{c}(x,x)\}\end{aligned}$$

Figure 4.10: Rules of $\mathcal{P}_6$.




Count $\|t\| = |t|_{\mathsf{f}(\mathsf{f}(\mathsf{a}))}$

Figure 4.11: Positively drifting random walk.

This PTRS behaves similarly to $\mathcal{P}_{\mathsf{dup}}$ (see Counterexample 4.2.1). It is $\mathtt{OR}$ and $\mathtt{SP}$, since every basic term $\mathsf{f}(t)$ is a normal form, and $\mathsf{d}(t)$ only duplicates the normal form $t$. We do not have $\mathtt{AST}(\xrightarrow{\mathtt{f}}_{\mathcal{P}_6})$ (and hence, also neither $\mathtt{PAST}(\xrightarrow{\mathtt{f}}_{\mathcal{P}_6})$ nor $\mathtt{SAST}(\xrightarrow{\mathtt{f}}_{\mathcal{P}_6})$), as we have $\{^{1}\mathsf{f}(\mathsf{f}(\mathsf{a}))\} \xrightarrow{\mathtt{f}}{}^{2}_{\mathcal{P}_6} \{^{3/4}\mathsf{c}(\mathsf{f}(\mathsf{f}(\mathsf{a})), \mathsf{f}(\mathsf{f}(\mathsf{a}))), {}^{1/4}\mathsf{0}\}$, which corresponds to a positively drifting random walk on the number of $\mathsf{f}(\mathsf{f}(\mathsf{a}))$ occurrences, see Figure 4.11.

In contrast, innermost evaluation has to rewrite the redex $\mathsf{f}(\mathsf{f}(\mathsf{a}))$ below $\mathsf{d}$ first, so that $\mathsf{d}$ only duplicates normal forms. Hence, the expected derivation height of a term $t$ w.r.t. $\xrightarrow{\mathtt{i}}_{\mathcal{P}_6}$ is finite. So we have $\mathtt{SAST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}_6})$ (and thus, also $\mathtt{PAST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}_6})$ and $\mathtt{AST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}_6})$) for arbitrary start terms, and Theorem 4.3.3 cannot be extended to $\mathtt{PSN}(\xrightarrow{\mathtt{f}}_{\mathcal{P}})$ in general.

For expected complexity, we obtain a result that is analogous to Theorem 4.3.3. Since we require basic start terms, the following theorem only holds for expected runtime complexity and not for expected derivational complexity (e.g., for $\mathcal{P}_6$ we have $\iota(\mathrm{edc}_{\xrightarrow{\mathtt{f}}_{\mathcal{P}_6}}) = \mathbf{Inf}$ but $\iota(\mathrm{edc}_{\xrightarrow{\mathtt{i}}_{\mathcal{P}_6}}) = \mathbf{Pol}_1$).

Theorem 4.3.6 – From Innermost to Full Expected Runtime Complexity.
If a PTRS $\mathcal{P}$ is $\mathtt{OR}$ and $\mathtt{SP}$, then

$$\mathrm{erc}_{\xrightarrow{\mathtt{f}}_{\mathcal{P}}} = \mathrm{erc}_{\xrightarrow{\mathtt{i}}_{\mathcal{P}}}\,.$$

*Proof.* This follows directly from Lemma 4.3.4, analogously to the proof of Theorem 4.2.5. *Proof* *QED*

Thus, this theorem weakens the requirement of $\mathtt{RL}$ in Theorem 4.2.5 to $\mathtt{SP}$ (recall that right-linearity implies spareness). A corresponding corollary for the non-probabilistic

setting would be subsumed by Theorem 4.1.10, which ensures $\mathrm{rc}_{\xrightarrow{\mathtt{f}}_{\mathcal{R}}} = \mathrm{rc}_{\xrightarrow{\mathtt{i}}_{\mathcal{R}}}$ for all spare overlay systems $\mathcal{R}$. As shown by Counterexample 4.2.6, OS is not enough in the probabilistic setting, but one needs more, e.g., OR.

SECTION 4.4

## Transformational Approaches

Instead of finding criteria on the PTRS that ensure equivalence of certain properties like innermost AST and full AST, one can also try a *transformational approach*. Given a PTRS $\mathcal{P}$ where we want to analyze a property like full AST, we transform it into another PTRS trans($\mathcal{P}$) such that a property of trans($\mathcal{P}$) that is easier to analyze automatically (e.g., innermost AST, or AST on basic start terms) implies (or is equivalent to) the desired property of $\mathcal{P}$. Then, it suffices to analyze the easier property of trans($\mathcal{P}$). We present two different transformations. The first one helps in analyzing derivational complexity by runtime complexity and the second one helps in analyzing outermost termination by innermost termination.

### 4.4.1 From Derivational to Runtime Complexity

Remember that the difference between derivational complexity and runtime complexity is that runtime complexity only considers rewrite sequences that start with a basic term. Similarly, Theorem 4.3.3 from the previous section only works if one considers basic start terms. One may wonder whether proving $\mathtt{PSN}(\xrightarrow{\mathtt{f}}_{\mathcal{P}})$ for a PTRS $\mathcal{P}$ on *all* terms (instead of just basic start terms) can be reduced to considering only basic start terms by using a suitable transformation from $\mathcal{P}$ to another PTRS trans($\mathcal{P}$) such that $\mathtt{PSN}(\xrightarrow{\mathtt{f}}_{\mathcal{P}})$ holds for all terms iff $\mathtt{PSN}(\xrightarrow{\mathtt{f}}_{\mathrm{trans}(\mathcal{P})})$ holds when starting with basic terms. In [Fuh19], a transformation was presented that extends any TRS $\mathcal{R}$ by generator rules $\mathcal{G}(\mathcal{R})$ such that the derivational complexity of $\mathcal{R}$ is upper bounded by the runtime complexity of $\mathcal{R} \cup \mathcal{G}(\mathcal{R})$. This transformation can indeed be reused to move from $\mathtt{PSN}(\xrightarrow{\mathtt{f}}_{\mathcal{P}})$ *starting in* $\mathcal{T}_{\mathcal{B}}$ to $\mathtt{PSN}(\xrightarrow{\mathtt{f}}_{\mathcal{P}})$ on arbitrary terms.

The idea of the transformation is to introduce a new constructor symbol $\mathsf{cons}_f$ for every defined symbol $f \in \mathcal{D}$, and to introduce a new defined symbol $\mathsf{enc}_f$ for every function symbol $f \in \Sigma$. As an example for $\mathcal{P}_6$ from Counterexample 4.3.5, instead of starting with the non-basic term $\mathsf{d}(\mathsf{f}(\mathsf{f}(\mathsf{a})))$, we start with the basic term $\mathsf{enc}_{\mathsf{d}}(\mathsf{cons}_{\mathsf{f}}(\mathsf{cons}_{\mathsf{f}}(\mathsf{a})))$, its *basic variant*. The new defined symbol $\mathsf{enc}_{\mathsf{d}}$ is used to first build the term $\mathsf{d}(\mathsf{f}(\mathsf{f}(\mathsf{a})))$ at the beginning of the rewrite sequence, i.e., it converts all occurrences of $\mathsf{cons}_f$ for $f \in \mathcal{D}$ back into the defined symbol $f$, and then proceeds as if we started with the term $\mathsf{d}(\mathsf{f}(\mathsf{f}(\mathsf{a})))$ directly. For this conversion, we need another new defined symbol $\mathsf{argenc}$ that iterates through the term and replaces all new constructors $\mathsf{cons}_f$ by the original defined symbol $f$. Thus, we define the generator rules as in [Fuh19] (just with trivial probabilities in the right-hand sides $\ell \to \{^1 r\}$).

Definition 4.4.1 – Generator Rules $\mathcal{G}(\mathcal{P})$.

*Let $\mathcal{P}$ be a PTRS over the signature $\Sigma$. Its generator rules $\mathcal{G}(\mathcal{P})$ are*

$$\begin{aligned}
&\{\mathsf{enc}_f(x_1,\dots,x_k) \to \{^1 f(\mathsf{argenc}(x_1),\dots,\mathsf{argenc}(x_k))\} \mid f \in \Sigma\}\\
\cup\ &\{\mathsf{argenc}(\mathsf{cons}_f(x_1,\dots,x_k)) \to \{^1 f(\mathsf{argenc}(x_1),\dots,\mathsf{argenc}(x_k))\} \mid f \in \mathcal{D}\}\\
\cup\ &\{\mathsf{argenc}(f(x_1,\dots,x_k)) \to \{^1 f(\mathsf{argenc}(x_1),\dots,\mathsf{argenc}(x_k))\} \mid f \in \mathcal{C}\},
\end{aligned}$$

*where $k$ is the arity of $f$, $x_1, \ldots, x_k$ are pairwise different variables, and the function symbols* $\mathsf{argenc}$ *(of arity 1),* $\mathsf{cons}_f$*, and* $\mathsf{enc}_f$ *(both of arity $k$) are fresh (i.e., they do not occur in $\Sigma \cup \mathcal{V}$). Moreover, we define $\Sigma_{\mathcal{G}(\mathcal{P})} = \{\mathsf{enc}_f \mid f \in \Sigma\} \cup \{\mathsf{argenc}\} \cup \{\mathsf{cons}_f \mid f \in \mathcal{D}\}$.*

Example 4.4.2. For the PTRS $\mathcal{P}_6$ from Counterexample 4.3.5, we obtain the following generator rules $\mathcal{G}(\mathcal{P}_6)$:

$$\begin{aligned}
\mathsf{enc_f}(x_1) &\to \{^1\mathsf{f}(\mathsf{argenc}(x_1))\}\\
\mathsf{enc_d}(x_1) &\to \{^1\mathsf{d}(\mathsf{argenc}(x_1))\}\\
\mathsf{enc_c}(x_1, x_2) &\to \{^1\mathsf{c}(\mathsf{argenc}(x_1), \mathsf{argenc}(x_2))\}\\
\mathsf{enc_a} &\to \{^1\mathsf{a}\}\\
\mathsf{enc_0} &\to \{^1\mathsf{0}\}\\
\mathsf{argenc}(\mathsf{cons_f}(x_1)) &\to \{^1\mathsf{f}(\mathsf{argenc}(x_1))\}\\
\mathsf{argenc}(\mathsf{cons_d}(x_1)) &\to \{^1\mathsf{d}(\mathsf{argenc}(x_1))\}\\
\mathsf{argenc}(\mathsf{c}(x_1, x_2)) &\to \{^1\mathsf{c}(\mathsf{argenc}(x_1), \mathsf{argenc}(x_2))\}\\
\mathsf{argenc}(\mathsf{a}) &\to \{^1\mathsf{a}\}\\
\mathsf{argenc}(\mathsf{0}) &\to \{^1\mathsf{0}\}
\end{aligned}$$

As mentioned, using the symbols $\mathsf{cons}_f$ and $\mathsf{enc}_f$, as in [Fuh19] every non-variable term over $\Sigma$ can be transformed into a basic term over $\Sigma \cup \Sigma_{\mathcal{G}(\mathcal{P})}$.

The following theorem shows that by adding the generator rules, one can indeed reduce the problem of proving $\mathtt{AST}$ on all terms to $\mathtt{AST}$ starting in $\mathcal{T}_{\mathcal{B}}$. This works not only for full, but also for innermost rewriting, i.e., in the following two theorems we consider both $s = \mathbf{f}$ and $s = \mathbf{i}$. Here, $\mathcal{T}_{\mathcal{B}}$ denotes the set of basic terms w.r.t. $\mathcal{P} \cup \mathcal{G}(\mathcal{P})$.

Theorem 4.4.3 – From All Terms to Basic Terms (1).

For any PTRS $\mathcal{P}$ and $s \in \{\mathbf{f}, \mathbf{i}\}$ we have

$$\mathtt{AST}(\xrightarrow{s}_{\mathcal{P}}) \iff \mathtt{AST}(\xrightarrow{s}_{\mathcal{P} \cup \mathcal{G}(\mathcal{P})}) \text{ starting in } \mathcal{T}_{\mathcal{B}}.$$

To prove Theorem 4.4.3 we need some more auxiliary functions from [Fuh19] to decode a basic term over $\Sigma \cup \Sigma_{\mathcal{G}(\mathcal{P})}$ into the original term over $\Sigma$.

Definition 4.4.4 – Constructor Variant, Basic Variant, Decoded Variant.

*Let $\mathcal{P}$ be a PTRS over the signature $\Sigma$. For a term $t \in \mathcal{T}$, we define its constructor variant* $\mathrm{cv}(t)$ *inductively as follows*

- $\mathrm{cv}(x) = x$ *for* $x \in \mathcal{V}$,
- $\mathrm{cv}(f(t_1, \ldots, t_n)) = f(\mathrm{cv}(t_1), \ldots, \mathrm{cv}(t_n))$ *for* $f \in \mathcal{C}$, *and*
- $\mathrm{cv}(f(t_1, \ldots, t_n)) = \mathsf{cons}_f(\mathrm{cv}(t_1), \ldots, \mathrm{cv}(t_n))$ *for* $f \in \mathcal{D}$.

*For a term $t \in \mathcal{T}$ with $t = f(t_1, \ldots, t_n)$, we define its basic variant* $\mathrm{bv}(f(t_1, \ldots, t_n)) = \mathsf{enc}_f(\mathrm{cv}(t_1), \ldots, \mathrm{cv}(t_n))$. *For a term $t \in \mathcal{T}(\Sigma \cup \Sigma_{\mathcal{G}(\mathcal{P})}, \mathcal{V})$, we define its decoded variant* $\mathrm{dv}(t) \in \mathcal{T}$ *as follows*

- $\mathrm{dv}(x) = x$ *for* $x \in \mathcal{V}$,
- $\mathrm{dv}(\mathsf{argenc}(t)) = \mathrm{dv}(t)$,
- $\mathrm{dv}(f(t_1, \ldots, t_n)) = g(\mathrm{dv}(t_1), \ldots, \mathrm{dv}(t_n))$ *for* $f \in \{g, \mathsf{cons}_g, \mathsf{enc}_g\}$ *with* $g \in \mathcal{D}$, *and*

- $\mathrm{dv}(f(t_1,\dots,t_n)) = g(\mathrm{dv}(t_1),\dots,\mathrm{dv}(t_n))$ *for* $f \in \{g, \mathsf{enc}_g\}$ *with* $g \in \mathcal{C}$.

The only difference to the auxiliary functions from [Fuh19] is that dv now also removes the $\mathsf{argenc}$ symbols from a term. This was handled differently in [Fuh19] in order to ensure $|\mathrm{dv}(t)| = |t|$, but this is irrelevant for our proofs regarding $\mathtt{PSN}(\xrightarrow{\mathbf{f}}_{\mathcal{P}})$.

*Proof of Theorem 4.4.3.* We first prove the statement for full rewriting ($s = \mathbf{f}$), showing both directions individually. Afterwards, we adapt both constructions to innermost rewriting ($s = \mathbf{i}$). *Proof*

▶ *Proof*($\Longleftarrow$, $s = \mathbf{f}$): Let $\mathfrak{T} = (N, E)$ be an $\xrightarrow{\mathbf{f}}_{\mathcal{P}}$-RT that starts with $t \in \mathcal{T}$. If $t \in \mathcal{V}$, then $t$ is a normal form, so $\mathfrak{T}$ consists of its root only and converges with probability 1. Hence, let $t \notin \mathcal{V}$, so that $\mathrm{bv}(t)$ is defined. We construct an $\xrightarrow{\mathbf{f}}_{\mathcal{P}\cup\mathcal{G}(\mathcal{P})}$-RT $\mathfrak{T}' = (N', E')$ that starts with $\mathrm{bv}(t)$ where $|\mathfrak{T}'| = |\mathfrak{T}|$.
In [Fuh19] it was shown that $\{{}^1\mathrm{bv}(t)\} \xrightarrow{\mathbf{i}}{}^*_{\mathcal{G}(\mathcal{P})} \{{}^1t\delta_t\}$. (PTRSs were not considered in [Fuh19], but since all the probabilities in the rules of $\mathcal{G}(\mathcal{P})$ are trivial, the proof in [Fuh19] directly translates to the probabilistic setting.) Here, for any term $t \in \mathcal{T}$ the substitution $\delta_t$ is defined by $\delta_t(x) = \mathsf{argenc}(x)$ if $x \in \mathcal{V}(t)$ and $\delta_t(x) = x$ otherwise. The $\xrightarrow{\mathbf{f}}_{\mathcal{P}\cup\mathcal{G}(\mathcal{P})}$-RT $\mathfrak{T}'$ first performs these innermost rewrite steps to get from $\mathrm{bv}(t)$ to $t\delta_t$, and then we can mirror the rewrite steps from $\mathfrak{T}$. To be precise, we use the same underlying tree structure and an adjusted labeling such that $p_x^{\mathfrak{T}} = p_x^{\mathfrak{T}'}$ and $t_x^{\mathfrak{T}}\delta_t = t_x^{\mathfrak{T}'}$ for all $x \in N$, see Figure 4.12. Since the tree structure and the probabilities are the same, we then obtain $|\mathfrak{T}| = |\mathfrak{T}'|$.

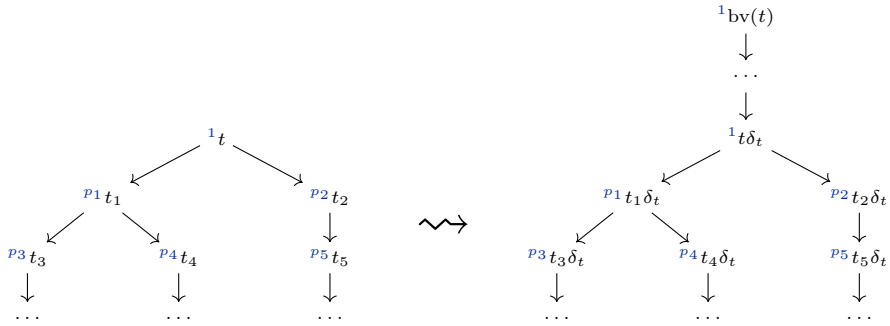



Figure 4.12: Construction of $\mathfrak{T}'$.

▶ *Proof*($\Longrightarrow$, $s = \mathbf{f}$): Let $\mathfrak{T} = (N, E)$ be an $\xrightarrow{\mathbf{f}}_{\mathcal{P}\cup\mathcal{G}(\mathcal{P})}$-RT that starts with $t \in \mathcal{T}_{\mathcal{B}}$. We construct an $\xrightarrow{\mathbf{f}}_{\mathcal{P}}$-RT $\mathfrak{T}' = (N', E')$ inductively such that for all leaves $x$ of $\mathfrak{T}'$ during the construction there exists a node $\varphi(x)$ of $\mathfrak{T}$ such that

$$t_x^{\mathfrak{T}'} = \mathrm{dv}(t_{\varphi(x)}^{\mathfrak{T}}) \quad \text{and} \quad p_x^{\mathfrak{T}'} = p_{\varphi(x)}^{\mathfrak{T}}. \tag{4.1}$$

Here, $\varphi$ is injective, i.e., every leaf $x$ of $\mathfrak{T}'$ is mapped to a (unique) node $\varphi(x)$ of $\mathfrak{T}$. Furthermore, after this construction, if $x$ is still a leaf in $\mathfrak{T}'$, then $\varphi(x)$ is also a leaf in $\mathfrak{T}$. Hence, we obtain $|\mathfrak{T}| = \sum_{x\in\mathrm{Leaf}^{\mathfrak{T}}} p_x^{\mathfrak{T}} \geq \sum_{x\in\mathrm{Leaf}^{\mathfrak{T}'}} p_x^{\mathfrak{T}'} = |\mathfrak{T}'|$.
The tree $\mathfrak{T}'$ starts with $\mathrm{dv}(t)$. By letting $\varphi$ map the root of $\mathfrak{T}'$ to the root of $\mathfrak{T}$, the claim (4.1) is clearly satisfied. As long as there is still a leaf $x$ of $\mathfrak{T}'$ such that $\varphi(x)$ is not a leaf in $\mathfrak{T}$, we do the following. If we perform a rewrite step with $\mathcal{G}(\mathcal{P})$ at node $\varphi(x)$, i.e., $t_{\varphi(x)}^{\mathfrak{T}} \xrightarrow{\mathbf{f}}_{\mathcal{G}(\mathcal{P})} \{{}^1t_y^{\mathfrak{T}}\}$ for the only successor $y$ of $\varphi(x)$, then we have $\mathrm{dv}(t_{\varphi(x)}^{\mathfrak{T}}) = \mathrm{dv}(t_y^{\mathfrak{T}})$, i.e., we do nothing in $\mathfrak{T}'$ but simply change the definition of $\varphi$ such that $\varphi(x)$ is

now $y$. To see why $\mathrm{dv}(t^{\mathfrak{T}}_{\varphi(x)}) = \mathrm{dv}(t^{\mathfrak{T}}_{y})$ holds, note that we have $\mathrm{dv}(\ell) = \mathrm{dv}(r)$ for all rules $\ell \to \{^1r\} \in \mathcal{G}(\mathcal{P})$, since $\mathrm{dv}(\mathsf{enc}_f(x_1,\dots,x_n)) = f(x_1,\dots,x_n) = \mathrm{dv}(f(\mathsf{argenc}(x_1),\dots,\mathsf{argenc}(x_n)))$, and analogously $\mathrm{dv}(\mathsf{argenc}(\mathsf{cons}_f(x_1,\dots,x_n))) = f(x_1,\dots,x_n) = \mathrm{dv}(f(\mathsf{argenc}(x_1),\dots,\mathsf{argenc}(x_n)))$. This can then be lifted to arbitrary rewrite steps. Moreover, there is no infinite path in $\mathfrak{T}$ that only consists of rewrite steps with $\mathcal{G}(\mathcal{P})$.
Otherwise, let $\varphi(x)E = \{y_1,\dots,y_k\}$ be the set of successors of $\varphi(x)$ in $\mathfrak{T}$, and we have $t^{\mathfrak{T}}_{\varphi(x)} \xrightarrow{\mathtt{f}}_{\mathcal{P}} \{^{p^{\mathfrak{T}}_{y_1}/p^{\mathfrak{T}}_{\varphi(x)}}t^{\mathfrak{T}}_{y_1},\dots,{}^{p^{\mathfrak{T}}_{y_k}/p^{\mathfrak{T}}_{\varphi(x)}}t^{\mathfrak{T}}_{y_k}\}$. Since $t^{\mathfrak{T}'}_{x} = \mathrm{dv}(t^{\mathfrak{T}}_{\varphi(x)})$ and $p^{\mathfrak{T}'}_{x} = p^{\mathfrak{T}}_{\varphi(x)}$ by (4.1), we also have $t^{\mathfrak{T}'}_{x} \xrightarrow{\mathtt{f}}_{\mathcal{P}} \{^{p^{\mathfrak{T}'}_{y_1}/p^{\mathfrak{T}'}_{x}}t^{\mathfrak{T}'}_{y_1},\dots,{}^{p^{\mathfrak{T}'}_{y_k}/p^{\mathfrak{T}'}_{x}}t^{\mathfrak{T}'}_{y_k}\}$ for terms $t^{\mathfrak{T}'}_{y_j}$ with $t^{\mathfrak{T}'}_{y_j} = \mathrm{dv}(t^{\mathfrak{T}}_{y_j})$ and $p^{\mathfrak{T}'}_{y_j} = p^{\mathfrak{T}}_{y_j}$. Thus, when defining $\varphi(y_j) = y_j$ for all $1 \le j \le k$, (4.1) is satisfied for the new leaves $y_1,\dots,y_k$ of $\mathfrak{T}'$.

▶ *Proof*($\Longleftarrow$, $s = \mathtt{i}$): Let $\mathfrak{T} = (N, E)$ be an $\xrightarrow{\mathtt{i}}_{\mathcal{P}}$-RT that starts with $t \in \mathcal{T}$. We use the same construction as for full rewriting, so it only remains to show that the resulting RT $\mathfrak{T}'$ is again innermost. The initial generator steps are also innermost w.r.t. the whole PTRS $\mathcal{P} \cup \mathcal{G}(\mathcal{P})$, as shown in [Fuh19]. For the mirrored steps, note that $\delta_t$ only introduces subterms of the form $\mathsf{argenc}(x)$ for variables $x$, which are normal forms w.r.t. $\mathcal{P} \cup \mathcal{G}(\mathcal{P})$, and that the symbols from $\Sigma_{\mathcal{G}(\mathcal{P})}$ do not occur in the rules of $\mathcal{P}$. Hence, innermost rewrite steps in $\mathfrak{T}$ correspond to innermost rewrite steps in $\mathfrak{T}'$.

▶ *Proof*($\Longrightarrow$, $s = \mathtt{i}$): Let $\mathfrak{T} = (N, E)$ be an $\xrightarrow{\mathtt{i}}_{\mathcal{P}\cup\mathcal{G}(\mathcal{P})}$-RT that starts with $t \in \mathcal{T}_{\mathcal{B}}$. We distinguish two cases depending on the root symbol of the basic term $t$.
First, consider the case where the root of $t$ is a defined symbol from $\mathcal{D}$. Then the symbols $\mathsf{enc}_f$ and $\mathsf{argenc}$ do not occur in $t$, as the arguments of the basic term $t$ are constructor terms. Since every left-hand side of $\mathcal{G}(\mathcal{P})$ has root $\mathsf{enc}_f$ or $\mathsf{argenc}$, no generator rule is ever applicable in $\mathfrak{T}$, and hence these symbols do not occur in any other term of $\mathfrak{T}$ either. Thus, the only symbols from $\Sigma_{\mathcal{G}(\mathcal{P})}$ occurring in $\mathfrak{T}$ are the constructors $\mathsf{cons}_f$, and every maximal subterm with root symbol $\mathsf{cons}_f$ of a term in $\mathfrak{T}$ is a term from $\mathcal{T}(\mathcal{C} \cup \{\mathsf{cons}_f \mid f \in \mathcal{D}\}, \mathcal{V})$, hence a normal form that lies completely inside the used substitution whenever it occurs in a redex of a rule of $\mathcal{P}$ (as $\mathsf{cons}_f$ does not occur in the rules of $\mathcal{P}$). So if we replace every such maximal subterm by a fresh variable (where equal subterms are replaced by the same variable and different subterms by pairwise different variables), then every rewrite step of $\mathfrak{T}$ remains a valid innermost rewrite step with the same rule. In this way, we obtain an $\xrightarrow{\mathtt{i}}_{\mathcal{P}}$-RT $\mathfrak{T}'$ with the same tree structure and the same probabilities as $\mathfrak{T}$ that starts with a term from $\mathcal{T}$, so that $|\mathfrak{T}| = |\mathfrak{T}'|$.
Otherwise, the root of $t$ is $\mathsf{enc}_f$ or $\mathsf{argenc}$. In this case, we use the same construction as for full rewriting, so it only remains to show that the rewrite steps of the resulting RT $\mathfrak{T}'$ are again innermost. In $t$, the arguments of the root symbol $\mathsf{enc}_f$ and $\mathsf{argenc}$ are terms from $\mathcal{T}(\mathcal{C} \cup \{\mathsf{cons}_f \mid f \in \mathcal{D}\}, \mathcal{V})$. For all steps in $\mathfrak{T}$ with rules from $\mathcal{G}(\mathcal{P})$ at node $\varphi(x)$ with successor $y$ we have $\mathrm{dv}(t^{\mathfrak{T}}_{\varphi(x)}) = \mathrm{dv}(t^{\mathfrak{T}}_{y})$, so no rewrite step is needed in $\mathfrak{T}'$. Whenever $\mathfrak{T}$ performs a step with a rule of $\mathcal{P}$, innermost rewriting ensures that the redex contains no $\mathsf{cons}_f$ and $\mathsf{argenc}$ only in the form $\mathsf{argenc}(x)$ with $x \in \mathcal{V}$, which are normal forms. Hence, we can copy all these rewrite steps and all of them remain innermost rewrite steps. *QED*

If one considers $\mathcal{G}(\mathcal{P})$ as *relative rules*, i.e., as rules whose rewrite steps do not count towards the derivation length, then similar results should also be possible for $\mathtt{PAST}(\xrightarrow{s}_{\mathcal{P}})$, $\mathtt{SAST}(\xrightarrow{s}_{\mathcal{P}})$, and both expected derivational and runtime complexity. When considering the additional generator rules as ordinary instead of relative rewrite rules, we obtain

an upper bound on the expected complexity and sufficient conditions for $\mathtt{PAST}(\xrightarrow{s}_{\mathcal{P}})$ and $\mathtt{SAST}(\xrightarrow{s}_{\mathcal{P}})$.

Theorem 4.4.5 – From All Terms to Basic Terms (2).
For any PTRS $\mathcal{P}$ and $s \in \{\mathbf{f}, \mathbf{i}\}$ we have

$$\begin{aligned} \mathtt{PAST}(\xrightarrow{s}_{\mathcal{P}}) &\impliedby \mathtt{PAST}(\xrightarrow{s}_{\mathcal{P}\cup\mathcal{G}(\mathcal{P})}) \text{ starting in } \mathcal{T}_{\mathcal{B}}, \\ \mathtt{SAST}(\xrightarrow{s}_{\mathcal{P}}) &\impliedby \mathtt{SAST}(\xrightarrow{s}_{\mathcal{P}\cup\mathcal{G}(\mathcal{P})}) \text{ starting in } \mathcal{T}_{\mathcal{B}}, \text{ and} \\ \operatorname{edc}_{\xrightarrow{s}_{\mathcal{P}}} &\leq \operatorname{erc}_{\xrightarrow{s}_{\mathcal{P}\cup\mathcal{G}(\mathcal{P})}}. \end{aligned}$$

*Proof.* Let $\mathfrak{T}$ be an $\xrightarrow{s}_{\mathcal{P}}$-RT that starts with $t \in \mathcal{T}$. If $t \in \mathcal{V}$, then $\operatorname{edl}(\mathfrak{T}) = 0$, so let $t \notin \mathcal{V}$. The construction from the proof of Theorem 4.4.3 yields an $\xrightarrow{s}_{\mathcal{P}\cup\mathcal{G}(\mathcal{P})}$-RT $\mathfrak{T}'$ that starts with $\operatorname{bv}(t)$ where $|\mathfrak{T}'| = |\mathfrak{T}|$. Since the tree structure and the probabilities are the same, and we only add rewrite steps in the beginning, we get $\operatorname{edl}(\mathfrak{T}') \geq \operatorname{edl}(\mathfrak{T})$. For the bound on the complexity, note that $\operatorname{bv}(t) \in \mathcal{T}_{\mathcal{B}}$ and $|\operatorname{bv}(t)| = |t|$, since cv and bv only replace function symbols by function symbols of the same arity. Hence, $\operatorname{edh}_{\xrightarrow{s}_{\mathcal{P}}}(t) \leq \operatorname{edh}_{\xrightarrow{s}_{\mathcal{P}\cup\mathcal{G}(\mathcal{P})}}(\operatorname{bv}(t))$ for all $t \notin \mathcal{V}$ implies $\operatorname{edc}_{\xrightarrow{s}_{\mathcal{P}}} \leq \operatorname{erc}_{\xrightarrow{s}_{\mathcal{P}\cup\mathcal{G}(\mathcal{P})}}$. *Proof* *QED*

However, this transformation does not help when checking the criteria mentioned in Theorem 4.3.3. While for $\mathcal{P} \cup \mathcal{G}(\mathcal{P})$ it suffices to consider only basic start terms, the PTRS $\mathcal{P} \cup \mathcal{G}(\mathcal{P})$ is not guaranteed to be spare, even if $\mathcal{P}$ is spare. Although the generator rules $\mathcal{G}(\mathcal{P})$ are right-linear, the problem is that the generator rules include a rule like $\mathsf{enc_d}(x_1) \to \{^1\mathsf{d}(\mathsf{argenc}(x_1))\}$ where a defined symbol $\mathsf{argenc}$ occurs below the duplicating symbol $\mathsf{d}$ on the right-hand side. Indeed, while $\mathcal{P}_6$ is spare, $\mathcal{P}_6 \cup \mathcal{G}(\mathcal{P}_6)$ is not. For example, when starting with the basic term $\mathsf{enc_d}(\mathsf{a})$, we have

$$\begin{aligned} \{^1\mathsf{enc_d}(\mathsf{a})\} &\xrightarrow[\mathcal{G}(\mathcal{P}_6)]{\mathbf{f}} \{^1\mathsf{d}(\mathsf{argenc}(\mathsf{a}))\} \\ &\xrightarrow[\mathcal{P}_6]{\mathbf{f}} \{^1\mathsf{c}(\mathsf{argenc}(\mathsf{a}), \mathsf{argenc}(\mathsf{a}))\} \end{aligned}$$

where the last step is not spare. In general, $\mathcal{P} \cup \mathcal{G}(\mathcal{P})$ is guaranteed to be spare if $\mathcal{P}$ is right-linear. So we could modify Theorem 4.3.3 into a theorem which states that $\mathtt{AST}(\xrightarrow{\mathbf{f}}_{\mathcal{P}})$ holds for all terms iff $\mathtt{AST}(\xrightarrow{\mathbf{i}}_{\mathcal{P}\cup\mathcal{G}(\mathcal{P})})$ holds when starting in $\mathcal{T}_{\mathcal{B}}$ (which, by Theorem 4.4.3, is equivalent to $\mathtt{AST}(\xrightarrow{\mathbf{i}}_{\mathcal{P}})$ on all terms) for orthogonal and right-linear PTRSs $\mathcal{P}$. However, this theorem would be subsumed by Theorem 4.2.2, where we already showed the equivalence of $\mathtt{AST}(\xrightarrow{\mathbf{f}}_{\mathcal{P}})$ and $\mathtt{AST}(\xrightarrow{\mathbf{i}}_{\mathcal{P}})$ if $\mathcal{P}$ is orthogonal and right-linear. Indeed, our goal in Theorem 4.3.3 was to find a weaker requirement than right-linearity. Hence, such a transformational approach to move from $\mathtt{AST}(\xrightarrow{\mathbf{f}}_{\mathcal{P}})$ on all start terms to $\mathtt{AST}(\xrightarrow{\mathbf{f}}_{\mathcal{P}})$ starting in $\mathcal{T}_{\mathcal{B}}$ does not seem to help using Theorem 4.3.3.

Nevertheless, this transformation is very useful to lift bounds on the expected runtime complexity to bounds on the expected derivational complexity of a PTRS, for both full and innermost evaluation.

### 4.4.2 From Outermost to Innermost Evaluation

The transformation from $\mathcal{P}$ to $\mathcal{P} \cup \mathcal{G}(\mathcal{P})$ for basic start terms uses generator rules that we apply before copying the original rewrite sequence. Since all generator rules only have trivial probabilities, the adaptation to the probabilistic setting is straightforward. Next, we present an adaptation of another transformation that uses additional rules not only before mirroring the rewrite sequence, but before and after *each* original

rewrite step. This illustrates that most transformational approaches should work in the probabilistic setting similar to the transformations in the non-probabilistic setting.

We present an adaptation of the transformation from [Thi09], which can be lifted to the probabilistic setting to analyze outermost termination via innermost termination, which is substantially easier to analyze. Unfortunately, the PTRSs that it produces are hard to handle for automated termination techniques.

The idea is to transform a given TRS $\mathcal{R}$ into a new TRS $\mathrm{trans}_{\mathbf{o}}(\mathcal{R})$ such that innermost termination of $\mathrm{trans}_{\mathbf{o}}(\mathcal{R})$ is equivalent to outermost termination of $\mathcal{R}$. To adapt this transformation for AST of PTRSs, as in [Thi09] we extend the signature by *marker* symbols. We use a symbol $\mathsf{top}$ to mark the root of the original term, and two symbols $\mathsf{reduce}$ and $\mathsf{go_up}$ to indicate the current position of the reduction. An outermost rewrite step $s \xrightarrow{\mathbf{o}}_{\mathcal{P}} \{^{p_1}t_1, \dots, ^{p_k}t_k\}$ is then performed in 3 different phases: First, we start with the term $\mathsf{top}(\mathsf{reduce}(s))$ and move the symbol $\mathsf{reduce}$ through the term until it reaches the root of an outermost redex. Then we perform the actual rewrite step at this redex. Finally, the symbol $\mathsf{go_up}$ is used to move the marker back to the root, resulting in terms of the form $\mathsf{top}(\mathsf{reduce}(t_j))$, for all $1 \leq j \leq k$.

Definition 4.4.6 – Transformation for Analyzing Outermost Rewriting.

*Let $\mathcal{P}$ be a PTRS over the signature $\Sigma$. We define the extended signature $\Sigma' = \Sigma \uplus \{\mathsf{top}, \mathsf{reduce}, \mathsf{go_up}, \mathsf{result}\} \uplus \{\mathsf{check}_f, \mathsf{redex}_f \mid f \in \Sigma\} \uplus \{\mathsf{in}_{f,i} \mid f \in \Sigma_m, 1 \leq i \leq m\}$. Additionally, we define the transformed PTRS $\mathrm{trans}_{\mathbf{o}}(\mathcal{P})$ over $\Sigma'$ to contain the rules*

$$\mathsf{reduce}(f(x_1, \dots, x_m)) \to \{^1\mathsf{check}_f(\mathsf{redex}_f(x_1, \dots, x_m))\} \tag{4.2}$$
$$\mathsf{check}_f(\mathsf{redex}_f(x_1, \dots, x_m)) \to \{^1\mathsf{in}_{f,i}(x_1, \dots, \mathsf{reduce}(x_i), \dots, x_m)\} \tag{4.3}$$
$$\mathsf{redex}_f(\ell_1, \dots, \ell_m) \to \{^{p_1}\mathsf{result}(r_1), \dots, ^{p_k}\mathsf{result}(r_k)\} \tag{4.4}$$
$$\mathsf{check}_f(\mathsf{result}(x)) \to \{^1\mathsf{go_up}(x)\} \tag{4.5}$$
$$\mathsf{in}_{f,i}(x_1, \dots, \mathsf{go_up}(x_i), \dots, x_m) \to \{^1\mathsf{go_up}(f(x_1, \dots, x_m))\} \tag{4.6}$$
$$\mathsf{top}(\mathsf{go_up}(x)) \to \{^1\mathsf{top}(\mathsf{reduce}(x))\} \tag{4.7}$$

*where Rules (4.2) and (4.5) are generated for all $f \in \Sigma$, Rules (4.3) and (4.6) for all $f \in \Sigma_m$ and $1 \leq i \leq m$, and Rule (4.4) for all $f(\ell_1, \dots, \ell_m) \to \{^{p_1}r_1, \dots, ^{p_k}r_k\} \in \mathcal{P}$.*

Example 4.4.7 (Outermost Transformation). Consider again the PTRS $\mathcal{P}_{\mathsf{dup}}$ from Counterexample 4.2.1 with the rules $\mathsf{g} \to \{^{3/4}\mathsf{d}(\mathsf{g}), ^{1/4}0\}$ and $\mathsf{d}(x) \to \{^1\mathsf{c}(x, x)\}$. The outermost rewrite step

$$\{^{1/2}\mathsf{g},\ ^{1/2}\mathsf{d}(\mathsf{g})\} \xrightarrow{\mathbf{o}}_{\mathcal{P}_{\mathsf{dup}}} \{^{3/8}\mathsf{d}(\mathsf{g}),\ ^{1/8}0,\ ^{1/2}\mathsf{c}(\mathsf{g}, \mathsf{g})\}$$

can be simulated with the following innermost rewrite sequence of $\mathrm{trans}_{\mathbf{o}}(\mathcal{P}_{\mathsf{dup}})$:

$$\begin{aligned}
&\{^{1/2}\mathsf{top}(\mathsf{reduce}(\mathsf{g})),\ ^{1/2}\mathsf{top}(\mathsf{reduce}(\mathsf{d}(\mathsf{g})))\}\\
\xrightarrow{\mathbf{i}}_{\mathrm{trans}_{\mathbf{o}}(\mathcal{P}_{\mathsf{dup}})} &\{^{1/2}\mathsf{top}(\mathsf{check}_{\mathsf{g}}(\mathsf{redex}_{\mathsf{g}})),\ ^{1/2}\mathsf{top}(\mathsf{check}_{\mathsf{d}}(\mathsf{redex}_{\mathsf{d}}(\mathsf{g})))\}\\
\xrightarrow{\mathbf{i}}_{\mathrm{trans}_{\mathbf{o}}(\mathcal{P}_{\mathsf{dup}})} &\{^{3/8}\mathsf{top}(\mathsf{check}_{\mathsf{g}}(\mathsf{result}(\mathsf{d}(\mathsf{g})))),\ ^{1/8}\mathsf{top}(\mathsf{check}_{\mathsf{g}}(\mathsf{result}(0))),\\
&\quad ^{1/2}\mathsf{top}(\mathsf{check}_{\mathsf{d}}(\mathsf{result}(\mathsf{c}(\mathsf{g}, \mathsf{g}))))\}\\
\xrightarrow{\mathbf{i}}_{\mathrm{trans}_{\mathbf{o}}(\mathcal{P}_{\mathsf{dup}})} &\{^{3/8}\mathsf{top}(\mathsf{go_up}(\mathsf{d}(\mathsf{g}))),\ ^{1/8}\mathsf{top}(\mathsf{go_up}(0)),\ ^{1/2}\mathsf{top}(\mathsf{go_up}(\mathsf{c}(\mathsf{g}, \mathsf{g})))\}\\
\xrightarrow{\mathbf{i}}_{\mathrm{trans}_{\mathbf{o}}(\mathcal{P}_{\mathsf{dup}})} &\{^{3/8}\mathsf{top}(\mathsf{reduce}(\mathsf{d}(\mathsf{g}))),\ ^{1/8}\mathsf{top}(\mathsf{reduce}(0)),\ ^{1/2}\mathsf{top}(\mathsf{reduce}(\mathsf{c}(\mathsf{g}, \mathsf{g})))\}.
\end{aligned}$$

Here, Phase 1, i.e., the search for a redex, consists of the first rewrite step (in general, this is done by Rules (4.2) and (4.3)). Then Phase 2, i.e., the actual reduction, is done in the second rewrite step (Rule (4.4)). Note that the innermost evaluation strategy for $\mathrm{trans}_{\mathbf{o}}(\mathcal{P}_{\mathsf{dup}})$ ensures that (4.3) is only applied if the subterm $\mathsf{redex}_f(\ldots)$ is in normal form. This ensures that one only applies (4.4) to terms that correspond to outermost redexes w.r.t. $\mathcal{P}$. Finally, the marker is moved back to the root in the last two rewrite steps (by Rules (4.5)–(4.7)), i.e., these steps correspond to Phase 3.

Theorem 4.4.8 – Soundness of the Outermost Transformation.
Let $\mathcal{P}$ be a PTRS. Then $\mathtt{AST}(\xrightarrow{\mathbf{o}}_{\mathcal{P}}) \impliedby \mathtt{AST}(\xrightarrow{\mathbf{i}}_{\mathrm{trans}_{\mathbf{o}}(\mathcal{P})})$.

*Proof.* The proof is analogous to the proof for the non-probabilistic setting in [Thi09]. We show that for every $t \xrightarrow{\mathbf{o}}_{\mathcal{P}} \{^{p_1}s_1, \ldots, ^{p_k}s_k\}$ we have $\{^1\mathsf{reduce}(t)\} \xrightarrow{\mathbf{i}}{}^{+}_{\mathrm{trans}_{\mathbf{o}}(\mathcal{P})} \{^{p_1}\mathsf{go_up}(s_1), \ldots, ^{p_k}\mathsf{go_up}(s_k)\}$. Then we can prove the claim by induction.
Let $t \xrightarrow{\mathbf{o}}_{\mathcal{P}} \{^{p_1}s_1, \ldots, ^{p_k}s_k\}$. Then we have $t = f(t_1, \ldots, t_m)$. We perform induction on the position $\pi$ where the rewrite step takes place.
If $\pi = \varepsilon$, then $t = f(t_1, \ldots, t_m) = \ell\sigma$ for some $\ell \to \{^{p_1}r_1, \ldots, ^{p_k}r_k\} \in \mathcal{P}$ and $s_i = r_i\sigma$ for all $1 \leq i \leq k$. We can build the innermost reduction

$$\begin{aligned}
&\{^1\mathsf{reduce}(f(t_1, \ldots, t_m))\}\\
\xrightarrow{\mathbf{i}}_{\mathrm{trans}_{\mathbf{o}}(\mathcal{P})} &\{^1\mathsf{check}_f(\mathsf{redex}_f(t_1, \ldots, t_m))\}\\
\xrightarrow{\mathbf{i}}_{\mathrm{trans}_{\mathbf{o}}(\mathcal{P})} &\{^{p_1}\mathsf{check}_f(\mathsf{result}(s_1)), \ldots, ^{p_k}\mathsf{check}_f(\mathsf{result}(s_k))\}\\
\xrightarrow{\mathbf{i}}_{\mathrm{trans}_{\mathbf{o}}(\mathcal{P})} &\{^{p_1}\mathsf{go_up}(s_1), \ldots, ^{p_k}\mathsf{go_up}(s_k)\}.
\end{aligned}$$

If $\pi = i.\tau$, then $t = f(t_1, \ldots, t_i, \ldots, t_m)$, $\{^1t_i\} \xrightarrow{\mathbf{o}}_{\mathcal{P}} \{^{p_1}s_{i,1}, \ldots, ^{p_k}s_{i,k}\}$, and $s_j = f(t_1, \ldots, s_{i,j}, \ldots, t_m)$ for all $1 \leq j \leq k$. By the induction hypothesis we have $\{^1\mathsf{reduce}(t_i)\} \xrightarrow{\mathbf{i}}{}^{+}_{\mathrm{trans}_{\mathbf{o}}(\mathcal{P})} \{^{p_1}\mathsf{go_up}(s_{i,1}), \ldots, ^{p_k}\mathsf{go_up}(s_{i,k})\}$. Therefore, we can build the innermost reduction

$$\begin{aligned}
&\{^1\mathsf{reduce}(f(t_1, \ldots, t_m))\}\\
\xrightarrow{\mathbf{i}}_{\mathrm{trans}_{\mathbf{o}}(\mathcal{P})} &\{^1\mathsf{check}_f(\mathsf{redex}_f(t_1, \ldots, t_m))\}\\
\xrightarrow{\mathbf{i}}_{\mathrm{trans}_{\mathbf{o}}(\mathcal{P})} &\{^1\mathsf{in}_{f,i}(t_1, \ldots, \mathsf{reduce}(t_i), \ldots, t_m)\}\\
\xrightarrow{\mathbf{i}}{}^{+}_{\mathrm{trans}_{\mathbf{o}}(\mathcal{P})} &\{^{p_1}\mathsf{in}_{f,i}(t_1, \ldots, \mathsf{go_up}(s_{i,1}), \ldots, t_m), \ldots, ^{p_k}\mathsf{in}_{f,i}(t_1, \ldots, \mathsf{go_up}(s_{i,k}), \ldots, t_m)\}\\
\xrightarrow{\mathbf{i}}_{\mathrm{trans}_{\mathbf{o}}(\mathcal{P})} &\{^{p_1}\mathsf{go_up}(f(t_1, \ldots, s_{i,1}, \ldots, t_m)), \ldots, ^{p_k}\mathsf{go_up}(f(t_1, \ldots, s_{i,k}, \ldots, t_m))\}\\
= &\{^{p_1}\mathsf{go_up}(s_1), \ldots, ^{p_k}\mathsf{go_up}(s_k)\}.
\end{aligned}$$

With this we can prove the claim by induction. Assume that $\neg\mathtt{AST}(\xrightarrow{\mathbf{o}}_{\mathcal{P}})$. Then there exists a $\xrightarrow{\mathbf{o}}_{\mathcal{P}}$-RT $\mathfrak{T}$ that starts with $t \in \mathcal{T}$ such that $|\mathfrak{T}| < 1$. We can create a $\xrightarrow{\mathbf{i}}_{\mathrm{trans}_{\mathbf{o}}(\mathcal{P})}$-RT $\mathfrak{T}'$ that starts with $\mathsf{top}(\mathsf{reduce}(t))$ such that $|\mathfrak{T}'| = |\mathfrak{T}| < 1$, and thus, $\neg\mathtt{AST}(\xrightarrow{\mathbf{i}}_{\mathrm{trans}_{\mathbf{o}}(\mathcal{P})})$.
For the construction of $\mathfrak{T}'$ we replace each rewrite step $t \xrightarrow{\mathbf{o}}_{\mathcal{P}} \{^{p_1}s_1, \ldots, ^{p_k}s_k\}$ of $\mathfrak{T}$ by the tree corresponding to the rewrite sequence

$$\begin{aligned}
&\{^1\mathsf{top}(\mathsf{reduce}(t))\}\\
\xrightarrow{\mathbf{i}}{}^{+}_{\mathrm{trans}_{\mathbf{o}}(\mathcal{P})} &\{^{p_1}\mathsf{top}(\mathsf{go_up}(s_1)), \ldots, ^{p_k}\mathsf{top}(\mathsf{go_up}(s_k))\}\\
\xrightarrow{\mathbf{i}}_{\mathrm{trans}_{\mathbf{o}}(\mathcal{P})} &\{^{p_1}\mathsf{top}(\mathsf{reduce}(s_1)), \ldots, ^{p_k}\mathsf{top}(\mathsf{reduce}(s_k))\}.
\end{aligned}$$

*Proof* *QED*

In addition to soundness, we conjecture that this transformation is complete as well. However, a corresponding proof for completeness would require more sophisticated tools for analyzing PTRSs that are not available at this point of the thesis. Therefore, we leave completeness open. In [Thi09] it was also shown that the transformation can be improved in order to generate fewer rules. These improvements should carry over to the transformation for PTRSs as well.

SECTION 4.5

## Evaluation

To evaluate the practical relevance of all of our contributions, we implemented them in the tool AProVE, see Chapter 10 for further details. Throughout the thesis, we evaluate different parts of AProVE that correspond to the theory we introduce in the respective chapters. For this evaluation we analyze all 199 PTRSs from the *Termination Problem Database* [TPDB], i.e., the benchmarks used for the annual *Termination Competition* [GRS+19]. These are typical probabilistic programs, including examples with complicated probabilistic structure and probabilistic algorithms on lists and trees. All experiments were performed on a machine with 16 GB of RAM and a timeout of 30 seconds for each example. Throughout all evaluations in this thesis, we do not report execution times, since the most important factor is currently the ability to prove a property at all, especially as we use a fairly short timeout. (On average, the proofs take less than five seconds.) Moreover, we do not evaluate every possible combination of the techniques presented in this thesis, as this would lead to an exponential blow-up in the number of configurations. Instead, we only present the most important evaluations that illustrate both the advantages and the disadvantages of the respective techniques.

To illustrate the power of the relations between the different evaluation strategies, we first measure how many of these PTRSs fall into the syntactic classes for which our relations guarantee that the different evaluation strategies coincide, see Table 4.1. Recall that innermost and full `AST` (and likewise `SAST`) coincide for PTRSs that are non-overlapping (`NO`) and linear (`L`), where linearity means both left-linearity (`LL`) and right-linearity (`RL`). In Section 4.3 we weakened right-linearity to spareness (`SP`) for rewrite sequences that start with basic terms. Finally, outermost and full `AST` coincide for PTRSs that are additionally non-erasing (`NE`). One can see that for a substantial part of the benchmark set, an analysis of the (typically much easier) innermost evaluation strategy already suffices to conclude the result for full or even outermost evaluation.

| Coinciding strategies | Sufficient criterion | # PTRSs (of 199) |
|---|---|---|
| innermost = full (`S`)`AST` | `NO`, `LL`, `RL` | 79 (39.7%) |
| improved for basic terms | `NO`, `LL`, `SP` | 113 (56.8%) |
| outermost = full (`S`)`AST` | `NO`, `LL`, `RL`, `NE` | 64 (32.2%) |

Table 4.1: How many PTRSs have the desired properties?

To show experimentally how well these relations perform in practice, we start by giving the total number of `AST` and `SAST` proofs for AProVE using all of the techniques later presented in this thesis (and also additional techniques that are not explained in this thesis, e.g., matrix interpretations) for the class that is easiest to prove: innermost evaluation with basic start terms, see Table 4.2. Afterwards, we show how many

of these AST and SAST proofs can be lifted using the relations between evaluation strategies in Table 4.3.

| Innermost AST w.r.t. basic start terms | Proved | Disproved |
|---|---|---|
| AProVE | 153 (76.9%) | 21 (10.6%) |

Table 4.2: The baseline on all 199 benchmarks of [TPDB] for basic start terms.

All percentages are relative to the 199 benchmarks. For basic start terms we can use the relations between innermost and the other strategies to get the following results. Here, we show both how many proofs can be lifted using the relations and how many examples can be proved without the relations. This illustrates both the power of the technique on its own and the power within a fully fledged tool like AProVE that contains multiple other techniques as well. Note that for outermost evaluation (and likewise for innermost evaluation), *disproving* AST is possible *only* via the relations: the loop-finding techniques of Chapter 9 are sound for full rewriting only, and the outermost-to-innermost transformation of Section 4.4 is sound only for proving. Hence, without the relations, no outermost AST disproofs can be obtained at all, and the table entry "Outermost evaluation, only lifted proofs" for disproving refers to the lifted disproofs from full evaluation. For *full* evaluation, the situation is reversed: there, the disproving techniques apply directly, and the innermost AST disproofs of the baseline are themselves obtained by moving the problem to full rewriting via the relations whenever the strategies coincide. Lifting innermost disproofs to full evaluation can therefore not yield any new results, so the "only lifted proofs" rows for full evaluation contain no disproofs.

| …AST w.r.t. basic start terms | Proved | Disproved |
|---|---|---|
| Full evaluation, only lifted proofs | 83 (41.7%) | 0 (0.0%) |
| Full evaluation, without relations | 99 (49.7%) | 37 (18.6%) |
| Full evaluation, with relations | 117 (58.8%) | 37 (18.6%) |
| Outermost evaluation, only lifted proofs | 47 (23.6%) | 14 (7.0%) |
| Outermost evaluation, without relations | 62 (31.2%) | 0 (0.0%) |
| Outermost evaluation, with relations | 78 (39.2%) | 14 (7.0%) |

Table 4.3: Evaluation using the theorems that allow us to switch the evaluation strategy to innermost (and full) evaluation for basic start terms.

Next, we do the same evaluation for arbitrary start terms. We first present the baseline in Table 4.4, and the results when using the relations are shown in Table 4.5.

| Innermost AST | Proved | Disproved |
|---|---|---|
| AProVE | 153 (76.9%) | 18 (9.0%) |

Table 4.4: The baseline on all 199 benchmarks of [TPDB] for arbitrary start terms.

| …AST | Proved | Disproved |
|---|---|---|
| Full evaluation, only lifted proofs | 58 (29.1%) | 0 (0.0%) |
| Full evaluation, without relations | 77 (38.7%) | 38 (19.1%) |
| Full evaluation, with relations | 90 (45.2%) | 38 (19.1%) |
| Outermost evaluation, only lifted proofs | 47 (23.6%) | 14 (7.0%) |
| Outermost evaluation, without relations | 62 (31.2%) | 0 (0.0%) |
| Outermost evaluation, with relations | 78 (39.2%) | 14 (7.0%) |

Table 4.5: Evaluation using the theorems that allow us to switch the evaluation strategy to innermost evaluation.

We can carry out the analogous evaluation for SAST and expected complexity. Here, we also illustrate the power of the transformational approach to move from derivational complexity analysis to runtime complexity analysis. Note that the modular framework for expected complexity that we present in Chapter 8 applies only to *innermost* evaluation with *basic* start terms, i.e., to innermost *runtime* complexity. The relations of this chapter including the transformations make this framework available to the other proof goals as well. Table 4.6 reports the number of SAST proofs (out of 199) obtained via finite expected complexity bounds as we switch each of these two liftings on individually and together. Recall that a finite bound on the expected derivational complexity proves SAST w.r.t. arbitrary start terms, whereas a finite bound on the expected runtime complexity proves SAST w.r.t. basic start terms, in both cases for the considered evaluation strategy (full or innermost).

| | derivational | | runtime | |
|---|---|---|---|---|
| Enabled liftings | full | innermost | full | innermost |
| none | 38 | 38 | 38 | 69 |
| innermost-to-full relation | 38 | 38 | 52 | 69 |
| derivational-to-runtime transf. | 38 | 54 | 38 | 69 |
| both | 43 | 54 | 52 | 69 |

Table 4.6: Evaluation of SAST.

The innermost runtime setting, i.e., innermost SAST w.r.t. basic start terms, is the one the framework handles natively, and it is the strongest throughout (69 proofs). Enabling the strategy relations lets full evaluation reuse the innermost analysis, e.g., raising the number of SAST proofs w.r.t. basic start terms for full evaluation from 38 to 52. Enabling the derivational-to-runtime transformation lets the derivational complexity analysis reuse the runtime complexity analysis, raising the number of innermost SAST proofs w.r.t. arbitrary start terms from 38 to 54. Only with *both* liftings does the number of SAST proofs w.r.t. arbitrary start terms for full evaluation reach its best value of 43. Thus, as for AST, the relations and the transformation substantially improve our analysis.

We additionally evaluate the *outermost*-to-innermost transformation of Section 4.4, which lifts an innermost AST proof of the transformed system to an outermost AST proof of the original PTRS. We compare AProVE on AST for outermost evaluation with

and without this transformation. In both configurations, all remaining techniques are enabled, including the relations of this chapter, so that the comparison isolates the contribution of the transformation itself.

| Outermost AST | Proved | Disproved |
|---|---|---|
| Without outermost → innermost | 78 (39.2%) | 14 (7.0%) |
| With outermost → innermost | 78 (39.2%) | 14 (7.0%) |

Table 4.7: Evaluation of the outermost-to-innermost transformation.

Unfortunately, on this benchmark set, the transformation does not yield additional results: every PTRS that AProVE proves outermost AST via the transformation can also be handled by applying interpretations directly or by the relations of this chapter. Moreover, the transformation cannot be used for disproving, as Theorem 4.4.8 is only sound in one direction. However, in contrast to the relations, the transformation is applicable to *every* PTRS without any syntactic requirements. In particular, it is the only technique that can potentially prove outermost AST for a PTRS that is not AST: applying interpretations directly proves AST, and the relations are only applicable if outermost and full AST coincide.

SECTION 4.6

## Related Work & Open Problems

Most of the related work that this chapter builds on has already been discussed in the previous sections. In the probabilistic rewriting setting, strategies were already considered in [BK02], and criteria for proving PAST of probabilistic rewrite systems under strategies were developed in [BG05, BG06, Gna07]. For ordinary TRSs, relations between evaluation strategies were established in a long line of work, including criteria for deriving termination from innermost termination [O'D77, Gra95], relations between leftmost-innermost and innermost rewriting [KR00], and correspondences between full and innermost runtime complexity [FG17a]. Regarding transformational approaches, apart from [Thi09, Fuh19] there exist further transformational approaches for TRSs, e.g., [RZ09, EH10].

In general, evaluation strategies are of interest for many properties of rewriting, e.g., for normalization, confluence, fairness, optimality, etc., see [BKd03, Chapter 9] and [Kvv07].

Another related line of research concerns probabilistic confluence or confluence of systems with infinitary behavior, and unique normal forms in the limit [DCM18, KC18, Fag22, FG22a]. These works indicate that classical proof patterns from ordinary rewriting, in particular those relying on well-founded induction, cannot be transferred directly and require new proof techniques. Beyond term rewriting, questions regarding evaluation strategies like efficiency are central in functional probabilistic formalisms, e.g., call-by-name [DLZ12, DE11] versus call-by-value [DL20] in probabilistic $\lambda$-calculi [FDR19], or even randomized evaluation strategies for the ordinary $\lambda$-calculus [DLV20].

The two most important open questions mentioned in this chapter mostly target confluence of probabilistic systems. It has been argued in [DCM18, Fag22] that solutions (at least to Open Problem 4.6.2) are likely to be very hard to find.

Open Problem 4.6.1 – Local Confluence Criteria for PTRSs.

Is every linear and non-overlapping PTRS locally confluent or even confluent, where both notions are defined via the lifting $\overset{\mathbf{f}}{\Rightarrow}_{\mathcal{P}}$ as in [DCM18] (see Section 4.2.1)?

Open Problem 4.6.2 – A Newman-Style Principle for AST.

Develop a probabilistic analogue of Newman's Lemma that yields a local condition for almost surely terminating PTRSs to ensure the global condition of confluence (in the sense of Open Problem 4.6.1).

Another interesting open problem is adapting Theorem 4.1.4 to the probabilistic setting.

Open Problem 4.6.3 – Relations for Overlay Systems.

Can we replace non-overlappingness in Theorem 4.2.2 by the requirement of locally confluent overlay systems?

Finally, one can adapt further transformational approaches from TRSs to PTRSs, e.g., [RZ09, EH10]. As already mentioned, it seems likely that most of the transformations for TRSs can be adapted in a straightforward way.

Open Problem 4.6.4 – Adapt Further Transformations.

Extend additional transformational approaches like [RZ09] from TRSs to PTRSs.

# 5 Structural Properties and Composability of PTRSs

There are many properties of a program that seem straightforward and intuitive. For example, consider the sequential execution "$P_1; P_2$" of two imperative programs $P_1$ and $P_2$. If both programs $P_1$ and $P_2$ are universally terminating, i.e., they terminate on all possible inputs, then so is $P_1; P_2$. The idea for a proof of this property is rather simple. After termination of the first program $P_1$, we start the second program $P_2$ with the output of $P_1$, but since both programs terminate on every possible input, $P_2$ terminates on the output of $P_1$ as well. In the probabilistic setting, even such a seemingly simple property is hard to prove formally, if it can be proved at all. In fact, only AST has such a composability result for sequential execution. This no longer holds for PAST or SAST, i.e., there exist probabilistic programs $P_1$ and $P_2$ that are both SAST, but the sequential execution $P_1; P_2$ is no longer SAST. Sequential execution is not only an interesting property for imperative programs, but can be used in the rewriting setting as well. Given two TRSs $\mathcal{R}^1$ and $\mathcal{R}^2$ that are terminating, the sequential execution of $\mathcal{R}^1$ and $\mathcal{R}^2$ is terminating as well, i.e., there exists no infinite sequence $t_0 \to_{\mathcal{R}^1} \cdots \to_{\mathcal{R}^1} t_i \to_{\mathcal{R}^2} t_{i+1} \to_{\mathcal{R}^2} \cdots$. We show that in the setting of probabilistic rewriting we have the same results as for imperative programs, i.e., only AST is closed under sequential execution but neither PAST nor SAST are.

Another interesting possible combination of programs is parallel execution. Again in terms of two imperative programs $P_1$ and $P_2$, the parallel execution $(P_1, P_2)$ runs both programs in parallel where we assume that there is no interaction between both of them. Again, universal termination of $(P_1, P_2)$ seems to follow trivially from universal termination of $P_1$ and $P_2$. Such a parallel combination of executions is omnipresent in the setting of term rewriting, where two executions starting in terms $t_1$ and $t_2$ are executed in parallel if they occur at parallel positions in a term, e.g., in a term like $\mathsf{c}(t_1, t_2)$ for some binary function symbol $\mathsf{c}$. We show that both AST and SAST are closed under parallel execution, while PAST is not.

Both of the aforementioned results regarding sequential execution and parallel execution are key properties used in many soundness proofs of different termination techniques. For example, we use both of them to obtain a witness theorem for non-AST, analogous to the non-termination witness results discussed in Section 3.2.4.

Compared to imperative programs, PTRSs have no fixed control flow. Therefore, another interesting combination of two systems is simply taking the union of all rewrite rules. A property $P$ of PTRSs is called *modular* if $P[\mathcal{P}] \iff P[\mathcal{P}^1] \wedge P[\mathcal{P}^2]$ for every union $\mathcal{P} = \mathcal{P}^1 \uplus \mathcal{P}^2$ of the considered kind. We present the first modularity results

(both positive and negative) for AST, PAST, and SAST of PTRSs. We will see that as in the non-probabilistic setting [Toy87a, Gra95], full probabilistic rewriting is not modular in general. Instead, we present results on the modularity under an innermost evaluation strategy, and we can utilize our results on the relation between innermost and full rewriting from Chapter 4 to obtain modularity results for full rewriting as well.

Modularity is not only interesting from a theoretical point of view, but also important in practice. One would like to know whether one can split a huge PTRS into smaller parts such that the property of interest is preserved. Then one can analyze these parts independently of each other. This is one of the main ideas of the DP framework for termination of TRSs [AG00, GAO02, GTS04, GTSF06, HM05, HM07] that we adapt to prove AST in Chapter 7, and to prove SAST and to infer upper bounds on the expected runtime complexity in Chapter 8. In both frameworks, we develop two processors, namely the *dependency graph processor* and the *usable rules processor*, that together subsume all the modularity results from this chapter. This is also the reason why we omit a detailed evaluation section for this chapter. However, the modularity results of this chapter yield theoretical insights on how powerful a corresponding DP framework is expected to be.

We consider three different types of unions $\mathcal{P} = \mathcal{P}^1 \uplus \mathcal{P}^2$, namely unions where $\mathcal{P}^1$ and $\mathcal{P}^2$ do not have any function symbol in common (*disjoint unions*), unions where $\mathcal{P}^1$ and $\mathcal{P}^2$ only share constructor symbols (*shared constructor unions*), and unions where defined symbols of $\mathcal{P}^2$ may be used as constructors in $\mathcal{P}^1$, but not vice versa (*hierarchical unions*), see [Ohl94, Gra95, Rao94, Rao95]. We show that for innermost evaluation, both AST and SAST are modular under disjoint unions, while only AST is modular under shared constructor unions, and that AST is preserved under hierarchical unions if the PTRSs satisfy additional requirements. All other modularity results, e.g., modularity of PAST under disjoint unions, are disproven by concrete counterexamples.

In the end, we consider signature extensions in Section 5.5. A property $P$ is closed under signature extensions if for every PTRS $\mathcal{P}$ we have $P[\mathcal{P}]$ over the signature $\Sigma$ if and only if $P[\mathcal{P}]$ over the signature $\Sigma' \supseteq \Sigma$. Here, again we show that both AST and SAST are closed under signature extensions, while PAST is not. These results are closely related to the results regarding the parallel execution, because as mentioned above we can add a fresh binary function symbol $\mathsf{c}$ and start with the term $\mathsf{c}(t_1, t_2)$.

Throughout this chapter, we only consider finite PTRSs, since several of our proofs rely on the fact that the rewrite relation of a finite PTRS is finitely branching. All the results regarding composability of this chapter are depicted in Table 5.1. Here, Yes means that the respective termination property is closed under the respective type of combination, No means that the termination property is not closed under the type of combination, and (Yes) means that it is not closed in general but with some additional requirements on the PTRS it is closed.

| Closed Under … | iAST | AST | iPAST | PAST | iSAST | SAST |
|---|---|---|---|---|---|---|
| Sequential Execution | Yes | Yes | No | No | No | No |
| Parallel Execution | Yes | Yes | No | No | Yes | Yes |
| Disjoint Unions | Yes | (Yes) | No | No | Yes | (Yes) |
| Shared Constructor Unions | Yes | (Yes) | No | No | No | No |
| Hierarchical Unions | (Yes) | (Yes) | No | No | No | No |
| Signature Extensions | Yes | Yes | No | No | Yes | Yes |

Table 5.1: Summary of all composability results of this chapter.

### Contributions

We present the following contributions in this chapter. The results marked with !⋆! are novel, while the remaining contributions can be found in the following peer-reviewed publication: ⋆ [KG25]. However, all the results from [KG25] are supplemented by several new examples and presented in a novel way.

- ⋆ The first probabilistic composability result for sequential execution of PTRSs: AST is closed under sequential composition (Lemma 5.1.3), while PAST and SAST are not (Counterexample 5.1.7).
- ⋆ The first probabilistic composability result for parallel execution of PTRSs: both AST and SAST are closed under parallel composition (Lemmas 5.2.2 and 5.2.4), while PAST is not (Counterexample 5.2.6).
- ⋆ A detailed witness analysis for non-AST, non-PAST, and non-SAST showing that minimality fails in the probabilistic setting (Counterexample 5.3.1), but for non-AST witnesses the start term $t$ still initially exhibits a root-step (Theorem 5.3.2) and is in argument normal form for innermost rewriting (Theorem 5.3.3). Compared to the non-probabilistic setting, this witness analysis requires the results for sequential and parallel execution.
- ⋆ Showing that both AST and SAST are modular for disjoint unions under innermost evaluation (Theorems 5.4.2 and 5.4.8), while PAST is not (Counterexample 5.4.3).
- ⋆ Showing that AST is modular for constructor sharing unions under innermost evaluation (Theorem 5.4.21), while PAST and SAST are not (Counterexamples 5.4.3 and 5.4.20).
- !⋆! Showing that AST is preserved under hierarchical unions under innermost evaluation and some additional requirements (Theorem 5.4.26), while PAST and SAST are not (Counterexamples 5.4.3 and 5.4.20).
- ⋆ Integration of results from Chapter 4 such that modularity of full AST and SAST can be recovered via the relations between innermost and full rewriting (Corollaries 5.4.18, 5.4.22 and 5.4.28).
- ⋆ Preservation of AST and SAST w.r.t. signature extensions for both full and innermost rewriting (Theorem 5.5.2), while PAST is not closed under signature extensions (Theorem 5.5.1).

## Structure

First, we investigate sequential execution in Section 5.1, followed by an investigation of parallel execution in Section 5.2. We use both of these results to analyze witnesses for non-AST, non-PAST, and non-SAST in Section 5.3, analogous to the non-termination witness results in Section 3.2.4. Afterwards, we present the modularity results for AST, PAST, and SAST in Section 5.4. In the end, we consider signature extensions in Section 5.5. We end this chapter by discussing related work and giving important open problems for future work in Section 5.6.

SECTION 5.1

## Sequential Execution

The most important combination of imperative programs is the sequential execution. In this section, we consider the sequential execution $P_1; P_2$ of two programs $P_1$ and $P_2$, and we analyze whether AST, PAST, or SAST are closed under such a combination. It is not hard to see that if both programs $P_1$ and $P_2$ are non-probabilistic and universally terminating, then so is $P_1; P_2$. We show that the same holds for AST but not for PAST and SAST, i.e., AST is closed under sequential execution, but PAST and SAST are not. While this was already known for imperative programs (see, e.g., [KKMO16] for the counterexample for PAST), we adapt these results to the setting of probabilistic term rewriting, where no fixed control structures exist.

We start by investigating AST and showing that it is indeed closed under sequential execution.

**Algorithm 3:** First program $P_1$.

**while** $x > 0$ **do**
  $x \leftarrow x - 1 \oplus_{1/2} x \leftarrow x + 1$;
  $y \leftarrow y + 1$;

**Algorithm 4:** Second program $P_2$.

**while** $y > 0$ **do**
  $y \leftarrow y - 1 \oplus_{1/2} y \leftarrow y + 1$;

Example 5.1.1 (Sequential Execution in Imperative Programs). Consider the two pGCL programs $P_1$ and $P_2$ from Algorithm 3 and Algorithm 4, respectively. Program $P_1$ models a symmetric random walk on the value of $x$. Additionally, during every loop iteration it increases the value of $y$ by 1. So after termination of $P_1$, the value of $y$ is increased by the number of loop iterations performed. Program $P_2$ models another symmetric random walk, but this time on the value of $y$. Both programs are AST as they model symmetric random walks. In fact, program $P_2$ terminates with probability 1 no matter what the input value of $y$ is. Therefore, also the sequential execution $P_1; P_2$ terminates with probability 1 and is AST.

Sequential execution in terms of rewriting means that we are given two TRSs $\mathcal{R}^1$ and $\mathcal{R}^2$ and we say that $\mathcal{R}^1$ followed by $\mathcal{R}^2$ is sequentially terminating if there exists no infinite sequence of the form $t_0 \to_{\mathcal{R}^1} \cdots \to_{\mathcal{R}^1} t_i \to_{\mathcal{R}^2} t_{i+1} \to_{\mathcal{R}^2} \cdots$, where we also allow infinite sequences that only use $\to_{\mathcal{R}^1}$. As for imperative programs, sequential termination of $\mathcal{R}^1$ followed by $\mathcal{R}^2$ is equivalent to termination of $\mathcal{R}^1$ and $\mathcal{R}^2$. Likewise, the sequential execution of $\mathcal{P}^1$ followed by $\mathcal{P}^2$ should be AST whenever both of the two PTRSs $\mathcal{P}^1$ and $\mathcal{P}^2$ are AST. Here, for sequential execution we consider $\to_{\mathcal{P}^1 \cup \mathcal{P}^2}$-RTs

where we are only allowed to use the relation $\to_{\mathcal{P}^2}$ once we reach a normal form w.r.t. $\mathcal{P}^1$, and after reaching a normal form w.r.t. $\mathcal{P}^1$ we are not allowed to use $\to_{\mathcal{P}^1}$ anymore. The resulting $\to_{\mathcal{P}^1 \cup \mathcal{P}^2}$-RTs still converge with probability 1. This composability of AST is the backbone of many techniques we present in the remainder of this thesis, and it is represented by the following *Sequential Execution Lemma*.

Example 5.1.2. We can model the imperative programs from Example 5.1.1 via the two PTRSs $\mathcal{P}^1$ and $\mathcal{P}^2$, where $\mathcal{P}^1$ contains the two rules $\mathsf{loop1}(0, y) \to \{^1\mathsf{loop2}(y)\}$ and $\mathsf{loop1}(\mathsf{s}(x), y) \to \{^{1/2}\mathsf{loop1}(x, \mathsf{s}(y)), {}^{1/2}\mathsf{loop1}(\mathsf{s}(\mathsf{s}(x)), \mathsf{s}(y))\}$, and $\mathcal{P}^2$ contains the rule $\mathsf{loop2}(\mathsf{s}(y)) \to \{^{1/2}\mathsf{loop2}(\mathsf{s}(\mathsf{s}(y))), {}^{1/2}\mathsf{loop2}(y)\}$. In Figure 5.1 one can see a $\to_{\mathcal{P}^1 \cup \mathcal{P}^2}$-RT that starts with $\mathsf{loop1}(\mathsf{s}(0), 0)$ and only uses the rule from $\mathcal{P}^2$ once we have already seen a normal form w.r.t. $\mathcal{P}^1$. Since all $\to_{\mathcal{P}^1}$-RTs and all $\to_{\mathcal{P}^2}$-RTs converge with probability 1, we can be sure that the $\to_{\mathcal{P}^1 \cup \mathcal{P}^2}$-RT in Figure 5.1 converges with probability 1 as well.

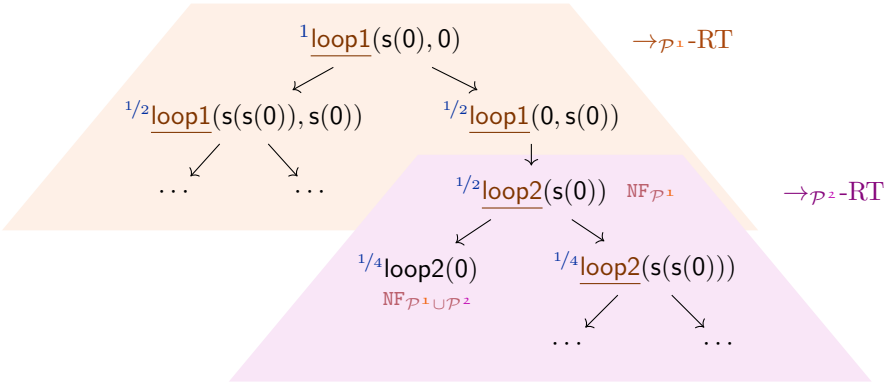



Figure 5.1: $\to_{\mathcal{P}^1 \cup \mathcal{P}^2}$-RT that only uses rules from $\mathcal{P}^2$ at nodes whose path from the root contains a normal form w.r.t. $\mathcal{P}^1$.

The *Sequential Execution Lemma* formalizes the property mentioned in Example 5.1.2 and generalizes it even further to allow an arbitrary number of nested sequential executions. To be precise, it states that if there exists an $\to$-RT $\mathfrak{T}$ that converges with probability $< 1$ and a partitioning of its inner nodes into $N_1 \uplus N_2$ such that every subtree of $\mathfrak{T}$ that only contains inner nodes from $N_1$ converges with probability 1, then we can create a subtree of $\mathfrak{T}$ that converges with probability $< 1$ such that every infinite path contains an infinite number of $N_2$ nodes (i.e., it does not contain any infinite path with eventually only nodes from $N_1$).

Let us explain why this lemma resembles sequential execution. For this consider the case where all nodes from $N_2$ come before the nodes from $N_1$ in $\mathfrak{T}$, i.e., all paths in $\mathfrak{T}$ have the form $v_0 v_1 v_2 \ldots \in N_2^* N_1^* \uplus N_2^* N_1^\omega \uplus N_2^\omega$. The path prefixes in $N_2^* \uplus N_2^\omega$ correspond to the execution of the first program $P_2$ and the path suffixes in $N_1^* \uplus N_1^\omega$ correspond to the execution of the second program $P_1$ in a sequential execution $P_2; P_1$. Then all the subtrees $\mathfrak{T}_1$ of $\mathfrak{T}$ which only contain inner nodes from $N_1$ are the executions of $P_1$. The Sequential Execution Lemma intuitively states in this case that if all the trees $\mathfrak{T}_1$ converge with probability 1 ($P_1$ is AST) and the sequential execution of both, namely $\mathfrak{T}$, does not ($P_2; P_1$ is not AST), then the first part of the program has to be the reason for that and is not AST either ($P_2$ is not AST).

**Lemma 5.1.3** – Sequential Execution Lemma.

*Let $(A, \to)$ be a PARS, and let $\mathfrak{T}$ be an $\to$-RT with $|\mathfrak{T}| < 1$. Suppose we can partition its inner nodes into $N_1 \uplus N_2$ such that $|\mathfrak{T}_1| = 1$ holds for every subtree $\mathfrak{T}_1$ of $\mathfrak{T}$ which only contains inner nodes from $N_1$. Then there exists a grounded subtree $\mathfrak{T}'$ of $\mathfrak{T}$ with $|\mathfrak{T}'| < 1$ such that every infinite path of $\mathfrak{T}'$ has an infinite number of nodes from $N_2$.*

*Proof.* Let $\mathfrak{T} = (N, E)$ be an $\to$-RT with $|\mathfrak{T}| = c < 1$ for some $c \in \mathbb{R}$. Since we have $0 \leq c < 1$, there is an $\epsilon > 0$ such that $c + \epsilon < 1$. Remember that the formula for the geometric series is

$$\sum_{n=1}^{\infty} \left(\tfrac{1}{d}\right)^n = \tfrac{1}{d-1}, \text{ for all } d \in \mathbb{R} \text{ such that } \tfrac{1}{|d|} < 1.$$

Let $d = \frac{1}{\epsilon} + 2$. Then we have $\frac{1}{d} = \frac{1}{\frac{1}{\epsilon}+2} < 1$ and

$$\tfrac{1}{\epsilon} + 1 < \tfrac{1}{\epsilon} + 2 \Leftrightarrow \tfrac{1}{\epsilon} + 1 < d \Leftrightarrow \tfrac{1}{\epsilon} < d - 1 \Leftrightarrow \tfrac{1}{d-1} < \epsilon \Leftrightarrow \sum_{n=1}^{\infty} \left(\tfrac{1}{d}\right)^n < \epsilon. \tag{5.1}$$

We will now construct a subtree $\mathfrak{T}' = (N', E')$ such that every infinite path has an infinite number of $N_2$ nodes and such that

$$|\mathfrak{T}'| \leq |\mathfrak{T}| + \sum_{n=1}^{\infty} \left(\tfrac{1}{d}\right)^n, \tag{5.2}$$

and then, we finally have

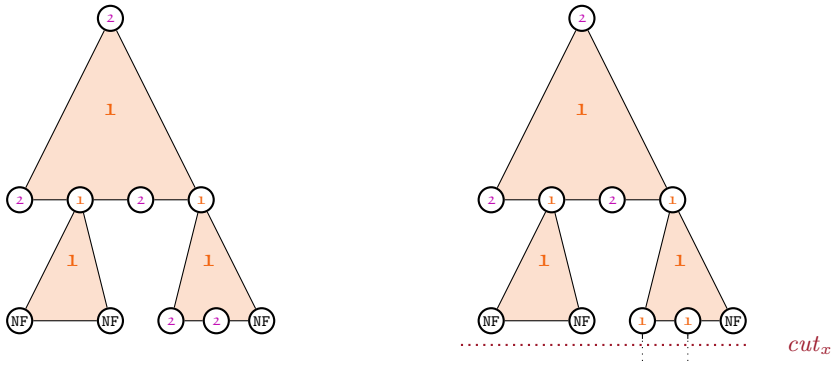
$$|\mathfrak{T}'| \overset{(5.2)}{\leq} |\mathfrak{T}| + \sum_{n=1}^{\infty} \left(\tfrac{1}{d}\right)^n = c + \sum_{n=1}^{\infty} \left(\tfrac{1}{d}\right)^n \overset{(5.1)}{<} c + \epsilon < 1.$$

The idea of this construction is that we cut infinite subtrees of pure $N_1$ nodes as soon as the probability for normal forms is high enough. In this way, one obtains paths where after finitely many $N_1$ nodes, there is an $N_2$ node, or we reach a leaf.
The construction works as follows. For any node $x \in N$, let $l(x)$ be the number of $N_2$ nodes in the path from the root to $x$. Furthermore, for any set $W \subseteq N$ and $k \in \mathbb{N}$, let $L(W, k) = \{x \in W \mid l(x) \leq k \vee (x \in N_2 \wedge l(x) \leq k + 1)\}$ be the set of all nodes in $W$ that have at most $k$ nodes from $N_2$ in the path from the root to their predecessor. So if $x \in L(W, k)$ is not in $N_2$, then we have at most $k$ nodes from $N_2$ in the path from the root to $x$ and if $x \in L(W, k)$ is in $N_2$, then we have at most $k + 1$ nodes from $N_2$ in the path from the root to $x$. We will inductively define a set $U_k \subseteq N$ such that $U_k \subseteq L(N, k)$ and then define the desired subtree as $\mathfrak{T}' = \mathfrak{T}[\bigcup_{k \in \mathbb{N}} U_k]$.
We start by considering the subtree $\mathfrak{T}_0 = \mathfrak{T}[L(N, 0)]$. This tree only contains inner nodes from $N_1$. While the node set $L(N, 0)$ itself may contain nodes from $N_2$, they can only occur at the leaves of $\mathfrak{T}_0$. Using the prerequisite of the Sequential Execution Lemma, we get $|\mathfrak{T}_0| = 1$. There are two possible cases for $\mathfrak{T}_0$, see Figure 5.2. Either $\mathfrak{T}_0$ is finite or $\mathfrak{T}_0$ is infinite. In the first case, we can add all the nodes to $U_0$ since there is no infinite path of pure $N_1$ nodes. Hence, we define $U_0 = L(N, 0)$. In the second case, we have to cut the tree at a specific depth once the probability of leaves is high enough. Let $\mathrm{d}_0(y)$ be the depth of the node $y$ in the tree $\mathfrak{T}_0$. Moreover, let $D_0(k) = \{y \in L(N, 0) \mid \mathrm{d}_0(y) \leq k\}$ be the set of nodes in $\mathfrak{T}_0$ that have depth at most

$k$. Since $|\mathfrak{T}_0| = 1$ and the total probability of the leaves of $\mathfrak{T}_0$ up to depth $k$ converges monotonically to $|\mathfrak{T}_0|$ for $k \to \infty$, we can find a $cut_0 \in \mathbb{N}$ such that

$$\sum_{x \in \mathrm{Leaf}^{\mathfrak{T}_0}, \mathrm{d}_0(x) \leq cut_0} p_x^{\mathfrak{T}_0} \geq 1 - \frac{1}{d}.$$

We include all nodes from $D_0(cut_0)$ in $U_0$ and delete every other node of $\mathfrak{T}_0$. In other words, we cut the tree after depth $cut_0 \in \mathbb{N}$. This cut is indicated by the dotted line in Figure 5.2. We now know that this cut may increase the probability of leaves by at most $\frac{1}{d}$. Therefore, we define $U_0 = D_0(cut_0)$ in this case.




$\mathfrak{T}_x$ is finite. $\mathfrak{T}_x$ is infinite.

Figure 5.2: Possibilities for $\mathfrak{T}_x$, which consists of its root $x \in N_2$ and the nodes below $x$ up to the next nodes from $N_2$: either $\mathfrak{T}_x$ is finite (left) and we can simply keep all of its nodes, or $\mathfrak{T}_x$ is infinite (right) and we have to cut the tree once the probability of reaching leaves is high enough.

For the induction step, assume that we have already defined a subset $U_i \subseteq L(N, i)$. Let $H_i = \{x \in U_i \mid x \in N_2, l(x) = i + 1\}$ be the set of leaves in $\mathfrak{T}[U_i]$ that are in $N_2$. For each $x \in H_i$, we consider the subtree that starts at $x$ until we reach the next node from $N_2$, including the node itself. Everything below such a node will be cut. To be precise, we regard the tree $\mathfrak{T}_x = (N_x, E_x) = \mathfrak{T}[L(xE^*, i + 1)]$. Here, $xE^*$ is the set of all nodes that are reachable from $x$ by arbitrarily many steps.
First, we show that $|\mathfrak{T}_x| = 1$. For every direct successor $y$ of $x$, the subtree $\mathfrak{T}_y = \mathfrak{T}_x[yE_x^*]$ of $\mathfrak{T}_x$ that starts at $y$ does not contain any inner nodes from $N_2$. Hence, we have $|\mathfrak{T}_y| = 1$ by the prerequisite of the Sequential Execution Lemma, and hence

$$|\mathfrak{T}_x| = \sum_{y \in xE} p_y \cdot |\mathfrak{T}_y| = \sum_{y \in xE} p_y \cdot 1 = \sum_{y \in xE} p_y = 1.$$

For the construction of $U_{i+1}$, we have the same cases as before, see Figure 5.2. Either $\mathfrak{T}_x$ is finite or $\mathfrak{T}_x$ is infinite. Let $Z_x$ be the set of nodes that we want to add to our node set $U_{i+1}$ from the tree $\mathfrak{T}_x$. In the first case, we can add all the nodes again and set $Z_x = N_x$. In the second case, we once again cut the tree at a specific depth once the probability for leaves is high enough. Let $\mathrm{d}_x(z)$ be the depth of the node $z$ in the tree $\mathfrak{T}_x$. Moreover, let $D_x(k) = \{z \in N_x \mid \mathrm{d}_x(z) \leq k\}$ be the set of nodes in $\mathfrak{T}_x$ that

have a depth of at most $k$. Since $|\mathfrak{T}_x| = 1$ and the total probability of the leaves of $\mathfrak{T}_x$ up to depth $k$ converges monotonically to $|\mathfrak{T}_x|$ for $k \to \infty$, we can find a $cut_x \in \mathbb{N}$ such that

$$\sum_{y \in \mathrm{Leaf}^{\mathfrak{T}_x}, \mathrm{d}_x(y) \leq cut_x} p_y^{\mathfrak{T}_x} \geq 1 - \left(\frac{1}{d}\right)^{i+2} \cdot \frac{1}{|H_i|}.$$

We include all nodes from $D_x(cut_x)$ in $U_{i+1}$ and delete every other node of $\mathfrak{T}_x$. In other words, we cut the tree after depth $cut_x \in \mathbb{N}$. We now know that this cut may increase the probability of leaves by at most $\left(\frac{1}{d}\right)^{i+2} \cdot \frac{1}{|H_i|}$. Therefore, we set $Z_x = D_x(cut_x)$.
We do this for each $x \in H_i$ and in the end, we set $U_{i+1} = U_i \cup \bigcup_{x \in H_i} Z_x$.
It is straightforward to see that $\bigcup_{k \in \mathbb{N}} U_k$ satisfies the conditions of Definition 3.3.12, as we only cut after certain nodes in our construction. Hence, $\bigcup_{k \in \mathbb{N}} U_k$ is non-empty and weakly connected, and for each of its nodes, it either contains no or all successors. Moreover, $U_0$ contains the root of $\mathfrak{T}$, so the resulting subtree is grounded. Furthermore, $\mathfrak{T}' = \mathfrak{T}[\bigcup_{k \in \mathbb{N}} U_k]$ is a subtree which does not contain an infinite path of pure $N_1$ nodes as we cut every such path after a finite depth.
It remains to prove that $|\mathfrak{T}'| \leq |\mathfrak{T}| + \sum_{n=1}^{\infty} \left(\frac{1}{d}\right)^n$ holds. During the $i$-th iteration of the construction, the cuts may increase the total probability of the leaves (compared to $|\mathfrak{T}|$) by the sum of all probabilities corresponding to the new leaves resulting from the cuts. As we cut at most $|H_i|$ trees in the $i$-th iteration and for each such tree, we added at most a total probability of $\left(\frac{1}{d}\right)^{i+2} \cdot \frac{1}{|H_i|}$ for the new leaves, the total probability of the leaves might increase by

$$|H_i| \cdot \left(\frac{1}{d}\right)^{i+2} \cdot \frac{1}{|H_i|} = \left(\frac{1}{d}\right)^{i+2}$$

in the $i$-th iteration, and hence in total, we then obtain

$$|\mathfrak{T}'| \leq |\mathfrak{T}| + \sum_{n=1}^{\infty} \left(\frac{1}{d}\right)^n,$$

as desired (see (5.2)). *QED*

The following corollary describes a single sequential execution of two PTRSs.

> Corollary 5.1.4 – Sequential Execution.
>
> *Let $\mathcal{P}^1$ and $\mathcal{P}^2$ be two finite PTRSs such that $\mathtt{AST}(\to_{\mathcal{P}^1})$ and $\mathtt{AST}(\to_{\mathcal{P}^2})$ both hold. Then, for every $\to_{\mathcal{P}^1 \cup \mathcal{P}^2}$-RT $\mathfrak{T}$ that uses only rules from $\mathcal{P}^1$ at nodes whose path from the root contains no normal form w.r.t. $\mathcal{P}^1$, and only rules from $\mathcal{P}^2$ at all other nodes, we have $|\mathfrak{T}| = 1$.*

*Proof.* Assume for a contradiction that there exists a $\to_{\mathcal{P}^1 \cup \mathcal{P}^2}$-RT $\mathfrak{T}$ with $|\mathfrak{T}| < 1$ that uses only rules from $\mathcal{P}^1$ at nodes whose path from the root contains no normal form w.r.t. $\mathcal{P}^1$, and only rules from $\mathcal{P}^2$ at all other nodes. We can partition the inner nodes from $\mathfrak{T}$ into the sets *Proof*

- $N_1 = \left\{ x \in N^{\mathfrak{T}} \setminus \mathrm{Leaf}^{\mathfrak{T}} \;\middle|\; \begin{array}{l} \text{there is no normal form w.r.t. } \mathcal{P}^1 \\ \text{in the path from the root to } x \end{array} \right\}$
- $N_2 = (N^{\mathfrak{T}} \setminus \mathrm{Leaf}^{\mathfrak{T}}) \setminus N_1$

By Theorem 5.5.2, $\mathtt{AST}(\to_{\mathcal{P}_1})$ and $\mathtt{AST}(\to_{\mathcal{P}_2})$ also hold for terms that contain function symbols of the other PTRS. Note that every connected component that only contains nodes from $N_2$ can be seen as a $\to_{\mathcal{P}_2}$-RT, and thus, terminates with probability 1. By the Sequential Execution Lemma (Lemma 5.1.3) (applied with the roles of $N_1$ and $N_2$ exchanged) there exists a subtree $\mathfrak{T}'$ of $\mathfrak{T}$ that contains infinitely many nodes from $N_1$ in every infinite path such that $|\mathfrak{T}'| < 1$. Since all inner nodes below a node from $N_2$ are from $N_2$ as well, no infinite path of $\mathfrak{T}'$ contains a node from $N_2$. So all subtrees of $\mathfrak{T}'$ that start in a node from $N_2$ are finite (by KÖNIG's Lemma, as $\mathfrak{T}'$ is finitely branching), and thus, they converge with probability 1. Hence, if we turn the topmost nodes from $N_2$ in $\mathfrak{T}'$ into leaves, we obtain a $\to_{\mathcal{P}_1}$-RT $\mathfrak{T}_1$ such that $|\mathfrak{T}_1| = |\mathfrak{T}'| < 1$, which is a contradiction. Therefore, such a $\to_{\mathcal{P}_1 \cup \mathcal{P}_2}$-RT $\mathfrak{T}$ has to converge with probability 1. *QED*

Example 5.1.5. Consider the PTRS $\mathcal{P}_1$ with the two rules $\mathsf{loop1}(\mathsf{s}(x), y) \to \{{}^{1/3}\mathsf{loop1}(x, \mathsf{s}(y)), {}^{2/3}\mathsf{loop1}(\mathsf{s}(\mathsf{s}(x)), \mathsf{s}(y))\}$ and $\mathsf{loop1}(0, y) \to \{{}^{1}\mathsf{loop2}(y)\}$, and the PTRS $\mathcal{P}_2$ with the rule $\mathsf{loop2}(\mathsf{s}(y)) \to \{{}^{1/2}\mathsf{loop2}(\mathsf{s}(\mathsf{s}(y))), {}^{1/2}\mathsf{loop2}(y)\}$. While $\mathcal{P}_2$ is the same as in Example 5.1.2, the PTRS $\mathcal{P}_1$ now models a positively drifting random walk. Consider the fully evaluated $\to_{\mathcal{P}_1 \cup \mathcal{P}_2}$-RT $\mathfrak{T}$ starting with $\mathsf{loop1}(\mathsf{s}(0), 0)$ again. This time we have $|\mathfrak{T}| < 1$. Since we know that $\mathcal{P}_2$ is AST, we can use the Sequential Execution Lemma to detect that the non-AST part of the reduction must be in the beginning, when we only use $\mathcal{P}_1$. Thus, $\mathcal{P}_1$ cannot be AST. In fact, since $\mathcal{P}_1$ models a positively drifting random walk it cannot be AST by Corollary 2.3.33.

Note that the Sequential Execution Lemma is stated for arbitrary PARSs, so it also applies to other evaluation strategies, e.g., to the PARS $(\mathcal{T}, \xrightarrow{\mathrm{i}}_{\mathcal{P}})$.

Next, we consider PAST and SAST and show why such a closedness result regarding sequential execution does not hold. We start by recapitulating the imperative case again. Note that while the termination probability of $P_2$ in a sequential execution $P_1; P_2$ is not influenced by the execution of $P_1$ if $P_2$ is AST, the runtime might be influenced, since we can change the input variables of $P_2$. More importantly, this influence on the runtime can even be exponential.

**Algorithm 5:** First program $P_1$.

**while** $x > 0$ **do**
  $x \leftarrow 0 \oplus_{1/2} x \leftarrow 1$;
  $y \leftarrow 2 \cdot y$;

**Algorithm 6:** Second program $P_2$.

**while** $y > 0$ **do**
  $y \leftarrow y - 1$;

Counterexample 5.1.6. Consider the two pGCL programs $P_1$ and $P_2$ from Algorithm 5 and Algorithm 6, respectively. The program $P_1$ models a coin flip on the value of $x$ and stops if we flip tails ($x = 0$) during the loop iteration. Additionally, the value of $y$ is doubled in each loop iteration. If we start initially with $y = 1$, then after termination of $P_1$ the value of $y$ is $2^n$ where $n$ is the number of loop iterations. The program $P_2$ simply counts down the value of $y$ by one in each loop iteration until it reaches 0. Both programs $P_1$ and $P_2$ are PAST and SAST. The second program $P_2$ even certainly terminates. However, if we start with values $x = 1$ and $y = 1$, then the expected runtime of $P_1; P_2$ is at least $1/2 \cdot 2 + (1/2)^2 \cdot 4 + (1/2)^3 \cdot 8 + \cdots = \sum_{i=1}^{\infty} (1/2)^i \cdot 2^i = \sum_{i=1}^{\infty} 1 = \infty$. The reason for this infinite expected runtime is that if we leave the first loop ($P_1$)

after $n$ iterations (which happens with probability $(1/2)^n$), then $y = 2^n$, and we have to perform exponentially many steps in the second loop ($P_2$). Thus, we have neither PAST nor SAST for $P_1; P_2$. This example is adapted from [KKMO16].

Due to the problem shown in Counterexample 5.1.6, an adaptation of the Sequential Execution Lemma would fail for PAST and SAST as shown in the following counterexample.

Counterexample 5.1.7. We can adapt Counterexample 5.1.6 to term rewriting using the PTRSs $\mathcal{P}^1$ and $\mathcal{P}^2$, where $\mathcal{P}^1$ contains the two rewrite rules $\mathsf{loop1}(\mathsf{s}(x), y) \to \{^{1/2}\mathsf{loop1}(0, \mathsf{double}(y)), {}^{1/2}\mathsf{loop1}(\mathsf{s}(x), \mathsf{double}(y))\}$ and $\mathsf{loop1}(0, y) \to \{^1\mathsf{loop2}(y)\}$, and $\mathcal{P}^2$ contains the only rule $\mathsf{loop2}(\mathsf{double}(y)) \to \{^1\mathsf{c}(\mathsf{loop2}(y), \mathsf{loop2}(y))\}$. Note that double is a constructor here, i.e., $\mathcal{P}^1$ only "stores" the doublings instead of computing them. Since doubling a unary number cannot be performed in a constant number of rewrite steps, computing the doublings would already make $\mathcal{P}^1$ itself non-PAST. Instead, $\mathcal{P}^2$ unfolds the stored doublings, so that evaluating $\mathsf{loop2}(\mathsf{double}^n(0))$ needs $2^n - 1$ steps, which corresponds to counting down the value $2^n$ of the variable $y$ in $P_2$. Here, we have both PAST and SAST for $\to_{\mathcal{P}^1}$ and for $\to_{\mathcal{P}^2}$. However, by the same argumentation as in Counterexample 5.1.6, the sequential execution is not PAST anymore: starting with $\mathsf{loop1}(\mathsf{s}(0), 0)$, with probability $(1/2)^n$ we reach the term $\mathsf{loop2}(\mathsf{double}^n(0))$, whose evaluation needs $2^n - 1$ steps, leading to an infinite expected derivation length. Therefore, the analogue of Corollary 5.1.4 fails for PAST (and analogously for SAST).

In this chapter we always end the sections by recapitulating the most important results. In this case, we have shown the following.

| Closed Under … | AST | PAST | SAST |
|---|---|---|---|
| Sequential Execution | Yes | No | No |

Table 5.2: Closure of AST, PAST, and SAST under sequential execution.

SECTION 5.2

## Parallel Execution

Parallel execution becomes interesting when a computer has multiple cores and can execute programs in parallel. If we only have a single CPU core, then running such a parallel program means that the CPU switches between both program executions and always runs one command after another sequentially. In this section, we investigate such parallel execution on a single core where both programs do not influence each other. Such a parallel combination of executions is omnipresent in the setting of term rewriting, where two executions starting in terms $t_1$ and $t_2$ are executed in parallel if they occur at parallel positions in a term. We show that both AST and SAST are closed under parallel execution, while PAST is not.

To be precise, we consider the parallel execution $(P_1, P_2)$ of two programs $P_1$ and $P_2$, and we analyze whether AST, PAST, or SAST are closed under such a combination. Compared to sequential execution, it may look even more immediate that universal termination of $P_1$ and $P_2$ is equivalent to universal termination of $(P_1, P_2)$ if the programs are completely independent of each other. However, this does not hold for

PAST, and it turns out that proving such a closure result for AST and SAST requires a rather complicated counting argument. For example, the main difficulty when considering AST is that we are interested in a *quantitative* measure on how likely it is to terminate, while in the non-probabilistic setting a *qualitative* analysis on whether the program terminates was enough. This quantitative measure requires a thorough and complex analysis of the termination probability of $(P_1, P_2)$.

Parallel execution in terms of term rewriting means that we are interested in the termination probability of $\mathsf{c}(q_1, \dots, q_n)$ assuming that all the terms $q_1, \dots, q_n \in \mathcal{T}$ are AST and where $\mathsf{c} \notin \Sigma \cup \mathcal{V}$ is a fresh function symbol of arity $n$. Note that if we rewrite a term $q_i \to \{{}^{p_1}q_{i,1}, \dots, {}^{p_k}q_{i,k}\}$, then we obtain a distribution $\{{}^{p_1}\mathsf{c}(q_1, \dots, q_{i,1}, \dots, q_n), \dots, {}^{p_k}\mathsf{c}(q_1, \dots, q_{i,k}, \dots, q_n)\}$. Now, the terms $q_j$ with $j \neq i$ have been "split" and occur multiple times in this distribution, and we may use different rules to rewrite them. Hence, the order in which we rewrite the different $q_i$ matters and cannot be chosen arbitrarily as seen in the following example.

Example 5.2.1. Reconsider the PTRS $\mathcal{P} = \mathcal{P}_4$ from Counterexample 4.2.11 with the five rules

$$\begin{aligned} \mathsf{a} &\to \{{}^{1}\mathsf{c}_1\} & \mathsf{b} &\to \{{}^{1/2}\mathsf{d}_1, {}^{1/2}\mathsf{d}_2\} \\ \mathsf{a} &\to \{{}^{1}\mathsf{c}_2\} & \mathsf{f}(\mathsf{c}_1, \mathsf{d}_1) &\to \{{}^{1}\mathsf{f}(\mathsf{a}, \mathsf{b})\} \\ & & \mathsf{f}(\mathsf{c}_2, \mathsf{d}_2) &\to \{{}^{1}\mathsf{f}(\mathsf{a}, \mathsf{b})\} \end{aligned}$$

We have $\neg\mathtt{AST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}})$, since there exists the infinite rewrite sequence $\{{}^{1}\mathsf{f}(\mathsf{a}, \mathsf{b})\} \xrightarrow{\mathtt{i}}_{\mathcal{P}} \{{}^{1/2}\mathsf{f}(\mathsf{a}, \mathsf{d}_1), {}^{1/2}\mathsf{f}(\mathsf{a}, \mathsf{d}_2)\} \xrightarrow{\mathtt{i}}_{\mathcal{P}} \{{}^{1/2}\mathsf{f}(\mathsf{c}_1, \mathsf{d}_1), {}^{1/2}\mathsf{f}(\mathsf{c}_2, \mathsf{d}_2)\} \xrightarrow{\mathtt{i}}_{\mathcal{P}} \{{}^{1/2}\mathsf{f}(\mathsf{a}, \mathsf{b}), {}^{1/2}\mathsf{f}(\mathsf{a}, \mathsf{b})\} \xrightarrow{\mathtt{i}}_{\mathcal{P}} \cdots$, which converges with probability 0. It first "splits" the term $\mathsf{f}(\mathsf{a}, \mathsf{b})$ with the b-rule, and then applies one of the two different a-rules to each of the resulting terms. In contrast, when applying a leftmost-innermost evaluation strategy, we have to decide which a-rule to use before we split the term with the b-rule. For example, we have $\{{}^{1}\mathsf{f}(\mathsf{a}, \mathsf{b})\} \xrightarrow{\mathtt{li}}_{\mathcal{P}} \{{}^{1}\mathsf{f}(\mathsf{c}_1, \mathsf{b})\} \xrightarrow{\mathtt{li}}_{\mathcal{P}} \{{}^{1/2}\mathsf{f}(\mathsf{c}_1, \mathsf{d}_1), {}^{1/2}\mathsf{f}(\mathsf{c}_1, \mathsf{d}_2)\}$. Here, the second term $\mathsf{f}(\mathsf{c}_1, \mathsf{d}_2)$ is a normal form. Since all leftmost-innermost rewrite sequences keep on adding such normal forms after a certain number of steps for each start term, we have $\mathtt{AST}(\xrightarrow{\mathtt{li}}_{\mathcal{P}})$. All in all, the "split" of the initial term is required for creating a rewrite sequence that does not converge with probability 1.

Since we cannot arbitrarily fix the evaluation strategy during the proof of the following lemma, the proof is much more complex than in the non-probabilistic setting for termination. Nevertheless, AST remains closed under parallel execution similar to termination in the non-probabilistic setting.

Lemma 5.2.2 – Parallel Execution Lemma for AST.
*Let $\mathcal{P}$ be a finite PTRS, and let $q_1, \dots, q_n \in \mathcal{T}$ be terms such that for every $1 \leq i \leq n$ and every $\to_{\mathcal{P}}$-RT $\mathfrak{T}_i$ that starts with $q_i$ we have $|\mathfrak{T}_i| = 1$. Then, every $\to_{\mathcal{P}}$-RT $\mathfrak{T}$ that starts with $\mathsf{c}(q_1, \dots, q_n)$ for some fresh symbol $\mathsf{c}$ converges with probability 1.*

*Proof.* By $\xrightarrow{\neg\varepsilon}_{\mathcal{P}}$ we denote the restriction of $\to_{\mathcal{P}}$ that does not perform any rewrite steps at the root position. We prove that every $\to_{\mathcal{P}}$-RT which starts with $\mathsf{c}(q_1, \dots, q_n)$ for some fresh symbol $\mathsf{c}$ converges with probability 1. Since $\mathsf{c}$ is fresh, rewriting with $\xrightarrow{\neg\varepsilon}_{\mathcal{P}}$ or $\to_{\mathcal{P}}$ yields the same results. Let $\overset{e}{\hookrightarrow}_{\mathcal{P}}$ be the restriction of $\xrightarrow{\neg\varepsilon}_{\mathcal{P}}$ where we can only rewrite on or below a position $1 \leq i \leq e$. Note that $\overset{n}{\hookrightarrow}_{\mathcal{P}} = \xrightarrow{\neg\varepsilon}_{\mathcal{P}}$. By induction on $e$, we prove that every $\overset{e}{\hookrightarrow}_{\mathcal{P}}$-RT which starts with $\mathsf{c}(q_1, \dots, q_n)$ converges

Proof

with probability 1.
In the base case we have $e = 1$ and only allow rewrite steps on or below position 1. Since every $\to_{\mathcal{P}}$-RT that starts with $q_1$ converges with probability 1 by our assumption, so does every $\overset{1}{\hookrightarrow}_{\mathcal{P}}$-RT that starts with $\mathsf{c}(q_1, \ldots, q_n)$.
In the induction step, we assume that the statement holds for $e-1$, where $1 < e \leq n$, i.e., every $\overset{e-1}{\hookrightarrow}_{\mathcal{P}}$-RT which starts with $\mathsf{c}(q_1, \ldots, q_n)$ converges with probability 1. Now consider an arbitrary $\overset{e}{\hookrightarrow}_{\mathcal{P}}$-RT $\mathfrak{T}$ that starts with $\mathsf{c}(q_1, \ldots, q_n)$. Assume for a contradiction that it converges with a probability $< 1$. The rest of the induction step has the following structure.

1. We first use the Sequential Execution Lemma (Lemma 5.1.3) to get rid of infinite paths that only use rewrite steps on or below position $e$, resulting in an $\overset{e}{\hookrightarrow}_{\mathcal{P}}$-RT $\mathfrak{T}'$, which uses infinitely many $\overset{e-1}{\hookrightarrow}_{\mathcal{P}}$-steps in each infinite path.
2. Then, for each finite height $H \in \mathbb{N}$ we split the tree $\mathfrak{T}'_H$, which consists of all nodes of the tree $\mathfrak{T}'$ with depth at most $H$, into multiple (finitely many) $\overset{e-1}{\hookrightarrow}_{\mathcal{P}}$-RTs of height at most $H$ that all start with $\mathsf{c}(q_1, \ldots, q_n)$. Furthermore, we show that for each $H \in \mathbb{N}$, at least one of these $\overset{e-1}{\hookrightarrow}_{\mathcal{P}}$-RTs has a low enough total probability of those leaves that are also leaves of $\mathfrak{T}'$.
3. We create an infinite, finitely branching tree $\mathbf{T}$ whose nodes are labeled with such $\overset{e-1}{\hookrightarrow}_{\mathcal{P}}$-RTs.
4. Finally, we use König's Lemma to obtain an infinite path in $\mathbf{T}$, which corresponds to an infinite $\overset{e-1}{\hookrightarrow}_{\mathcal{P}}$-RT that starts with $\mathsf{c}(q_1, \ldots, q_n)$ and converges with probability $< 1$, which is our desired contradiction.

**Step 1** Remove infinite paths that only rewrite on or below position $e$

We can partition the inner nodes of our $\overset{e}{\hookrightarrow}_{\mathcal{P}}$-RT $\mathfrak{T}$ into the sets

- $N_1 = \{x \in N^{\mathfrak{T}} \setminus \mathrm{Leaf}^{\mathfrak{T}} \mid \text{the rewrite step at } x \text{ is on or below position } e\}$
- $N_2 = (N^{\mathfrak{T}} \setminus \mathrm{Leaf}^{\mathfrak{T}}) \setminus N_1$

Note that in $N_2$ we rewrite on or below a position $1 \leq i \leq e-1$. We know that every $\to_{\mathcal{P}}$-RT that starts with $q_e$ converges with probability 1 by assumption. Hence, also every subtree of $\mathfrak{T}$ that only contains nodes from $N_1$ as inner nodes converges with probability 1. Thus, we can apply the Sequential Execution Lemma (Lemma 5.1.3) and obtain a subtree $\mathfrak{T}'$ of $\mathfrak{T}$ that converges with probability $< 1$ and contains infinitely many nodes of $N_2$ in each infinite path.

**Step 2** Split the tree $\mathfrak{T}'_H$ for every $H \in \mathbb{N}$

Let $H \in \mathbb{N}$ and let $\mathfrak{T}'_H$ be the finite tree consisting of all nodes of $\mathfrak{T}'$ with depth at most $H$. Since $\mathfrak{T}'$ converges with probability $< 1$, there exists an $\alpha$ with $0 < \alpha < 1 - |\mathfrak{T}'|$ such that $|\mathfrak{T}'| = \lim_{H\to\infty} \sum_{x \in \mathrm{Leaf}^{\mathfrak{T}'} \wedge \mathrm{d}(x) \leq H} p_x^{\mathfrak{T}'} < 1 - \alpha$, and hence

$$\sum_{x \in \mathrm{Leaf}^{\mathfrak{T}'} \wedge \mathrm{d}(x) \leq H} p_x^{\mathfrak{T}'} = \sum_{x \in \mathrm{Leaf}^{\mathfrak{T}'} \wedge x \in \mathrm{Leaf}^{\mathfrak{T}'_H}} p_x^{\mathfrak{T}'_H} < 1 - \alpha \tag{5.3}$$

for every $H \in \mathbb{N}$. Note that $\mathfrak{T}'_H$ is not a valid $\xhookrightarrow{e-1}_{\mathcal{P}}$-RT, as we still have steps where we perform a rewrite step on or below position $e$.
From $\mathfrak{T}'_H$ we generate a set $\mathbb{T}_H$ of pairs $(p_\mathfrak{t}, \mathfrak{t})$ where $0 < p_\mathfrak{t} \leq 1$ is a probability and $\mathfrak{t}$ is an $\xhookrightarrow{e-1}_{\mathcal{P}}$-RT of height at most $H$ (i.e., we do not perform any rewrite steps on or below position $e$ anymore). The set $\mathbb{T}_H$ contains all $\xhookrightarrow{e-1}_{\mathcal{P}}$-RTs $\mathfrak{t}$ resulting from $\mathfrak{T}'_H$ when skipping the nodes of $N_1$ (where the rewrite step is on or below the position $e$). Instead, one uses one of its child nodes directly. Here, every node of $\mathfrak{t}$ is labeled with the term of the corresponding node of $\mathfrak{T}'_H$ where the argument at position $e$ is replaced by $q_e$ (as all rewrite steps on or below position $e$ are skipped), so that $\mathfrak{t}$ indeed starts with $\mathsf{c}(q_1, \ldots, q_n)$. Moreover, $p_\mathfrak{t}$ is the probability for choosing the $\xhookrightarrow{e-1}_{\mathcal{P}}$-RT $\mathfrak{t}$. We will give more intuition on $\mathbb{T}_H$ below. Since a rewrite step on or below the position $e$ may create several children (one for each term in the support of the multi-distribution on the right-hand side of the applied rewrite rule), the set $\mathbb{T}_H$ may contain several pairs. $\mathbb{T}_H$ will satisfy the following two properties.

(Prop-1) We have $\sum_{(p_\mathfrak{t},\mathfrak{t}) \in \mathbb{T}_H} p_\mathfrak{t} = 1$. This means that the sum of the probabilities for all trees in $\mathbb{T}_H$ sums up to one.

(Prop-2) For all $x \in \mathrm{Leaf}^{\mathfrak{T}'_H} \cup (N_2 \cap N^{\mathfrak{T}'_H})$ we have $p_x^{\mathfrak{T}'_H} = \sum_{(p_\mathfrak{t},\mathfrak{t}) \in \mathbb{T}_H} p_\mathfrak{t} \cdot p_x^\mathfrak{t}$. As mentioned above, $p_\mathfrak{t}$ represents the probability that the $\xhookrightarrow{e-1}_{\mathcal{P}}$-RT $\mathfrak{t}$ is chosen and $p_x^\mathfrak{t}$ is the probability of the node $x$ in the $\xhookrightarrow{e-1}_{\mathcal{P}}$-RT $\mathfrak{t}$. Whenever $x$ is not a node of $\mathfrak{t}$ (i.e., $x \notin N^\mathfrak{t}$), then we define $p_x^\mathfrak{t} = 0$. This means that the probability for a leaf $x \in \mathrm{Leaf}^{\mathfrak{T}'_H}$ or an inner node $x \in N_2$ in our cut tree $\mathfrak{T}'_H$ is equal to the sum over all trees $\mathfrak{t}$ that contain $x$, where we multiply the probability $p_\mathfrak{t}$ of the tree $\mathfrak{t}$ by the probability of node $x$ in $\mathfrak{t}$.

Assume that for every $(p_\mathfrak{t}, \mathfrak{t}) \in \mathbb{T}_H$ we have $\sum_{x \in \mathrm{Leaf}^\mathfrak{t} \wedge x \in \mathrm{Leaf}^{\mathfrak{T}'}} p_x^\mathfrak{t} \geq 1 - \alpha$. Then, with the two properties (Prop-1) and (Prop-2) we obtain

$$
\begin{aligned}
& \textstyle\sum_{x \in \mathrm{Leaf}^{\mathfrak{T}'_H} \wedge x \in \mathrm{Leaf}^{\mathfrak{T}'}} p_x^{\mathfrak{T}'_H} \qquad (5.4)\\
& \quad \downarrow \ \text{(by (Prop-2))} \\
&= \textstyle\sum_{x \in \mathrm{Leaf}^{\mathfrak{T}'_H} \wedge x \in \mathrm{Leaf}^{\mathfrak{T}'}} \sum_{(p_\mathfrak{t},\mathfrak{t}) \in \mathbb{T}_H} p_\mathfrak{t} \cdot p_x^\mathfrak{t} \\
&= \textstyle\sum_{(p_\mathfrak{t},\mathfrak{t}) \in \mathbb{T}_H} \sum_{x \in \mathrm{Leaf}^{\mathfrak{T}'_H} \wedge x \in \mathrm{Leaf}^{\mathfrak{T}'}} p_\mathfrak{t} \cdot p_x^\mathfrak{t} \\
&= \textstyle\sum_{(p_\mathfrak{t},\mathfrak{t}) \in \mathbb{T}_H} p_\mathfrak{t} \cdot \sum_{x \in \mathrm{Leaf}^{\mathfrak{T}'_H} \wedge x \in \mathrm{Leaf}^{\mathfrak{T}'}} p_x^\mathfrak{t} \\
&= \textstyle\sum_{(p_\mathfrak{t},\mathfrak{t}) \in \mathbb{T}_H} p_\mathfrak{t} \cdot \sum_{x \in \mathrm{Leaf}^\mathfrak{t} \wedge x \in \mathrm{Leaf}^{\mathfrak{T}'}} p_x^\mathfrak{t} \\
& \quad \downarrow \ \text{(by assumption)} \\
&\geq \textstyle\sum_{(p_\mathfrak{t},\mathfrak{t}) \in \mathbb{T}_H} p_\mathfrak{t} \cdot (1 - \alpha) \\
&= (1 - \alpha) \cdot \textstyle\sum_{(p_\mathfrak{t},\mathfrak{t}) \in \mathbb{T}_H} p_\mathfrak{t} \\
& \quad \downarrow \ \text{(by (Prop-1))} \\
&= (1 - \alpha) \cdot 1 \\
&= 1 - \alpha,
\end{aligned}
$$

which is a contradiction to (5.3). Hence, for each $H \in \mathbb{N}$ there exists a $(p_\mathfrak{t}, \mathfrak{t}) \in \mathbb{T}_H$

such that

$$\sum_{x \in \mathrm{Leaf}^{\mathfrak{t}} \wedge x \in \mathrm{Leaf}^{\mathfrak{T}'}} p_x^{\mathfrak{t}} < 1 - \alpha. \tag{5.5}$$

We will use this in Steps 3 and 4 to generate our infinite $\xhookrightarrow{e-1}_{\mathcal{P}}$-RT that converges with probability $< 1$. But first we have to define the set $\mathbb{T}_H$ that satisfies (Prop-1) and (Prop-2).

**Step 2.1** Idea of $\mathbb{T}_H$

The idea of the split can be seen in Figures 5.3 and 5.4. There, the probabilities of the nodes are indicated by the small numbers and the probabilities for the edges are indicated by the big numbers. The probabilities of the edges below the node with a step on or below position $e$ are purple, since they become the probabilities of the resulting trees in Figure 5.4. Whenever we encounter a node from $N_1$ with successors $y_1, \ldots, y_m$, meaning that we rewrite on or below position $e$, then we split the tree into $m$ different trees, since the terms below positions 1 to $e-1$ did not change, and we may use different rules on the different resulting terms afterwards.

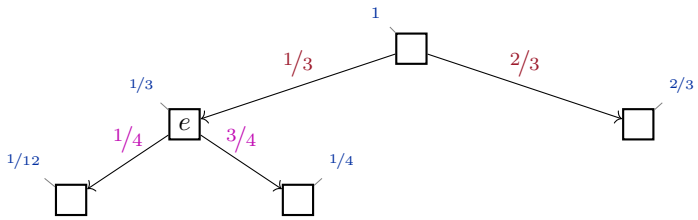



Figure 5.3: Example tree $\mathfrak{T}'_2$ with a step on or below position $e$ in the left node.

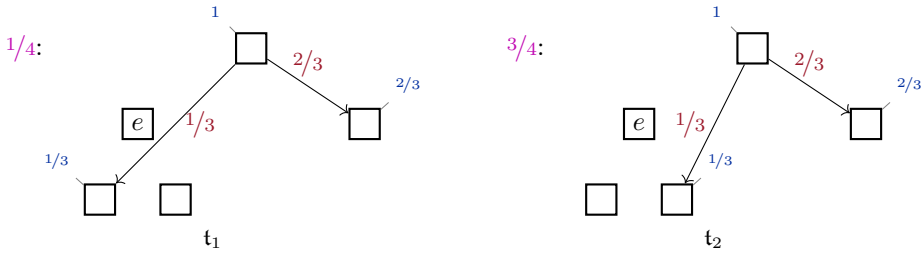



Figure 5.4: The set $\mathbb{T}_2 = \{(1/4, \mathfrak{t}_1), (3/4, \mathfrak{t}_2)\}$.

**Step 2.2** Constructing $\mathbb{T}_H$

To construct $\mathbb{T}_H$, we repeatedly remove all nodes of $N_1$ from the tree $\mathfrak{T}'_H$ and create a set $M$ inductively that satisfies the following properties.

(Ind-1) $\sum_{(p_{\mathfrak{t}}, \mathfrak{t}) \in M} p_{\mathfrak{t}} = 1$.

(Ind-2) For all $(p_{\mathfrak{t}}, \mathfrak{t}) \in M$ and all inner nodes $x \in N^{\mathfrak{t}}$ with successors $y_1, \ldots, y_k$ in $\mathfrak{t}$, the edge relation either represents a valid rewrite step on or below a position $1 \leq j \leq e-1$, or the subterms on or below the positions $1 \leq j \leq e-1$ remain the same, but the probabilities of the successors still sum up to the probability of node $x$, i.e., $\sum_{j=1}^{k} p_{y_j}^{\mathfrak{t}} = p_x^{\mathfrak{t}}$.

(Ind-3) For all $x \in \mathrm{Leaf}^{\mathfrak{T}'_H} \cup (N_2 \cap N^{\mathfrak{T}'_H})$ we have $p_x^{\mathfrak{T}'_H} = \sum_{(p_\mathfrak{t},\mathfrak{t}) \in M} p_\mathfrak{t} \cdot p_x^\mathfrak{t}$.

We stop the construction once every $\mathfrak{t} \in M$ is a valid $\xhookrightarrow{e-1}_{\mathcal{P}}$-RT, i.e., once we have removed all nodes that correspond to rewrite steps on or below position $e$. In the end, we obtain a set $M$ that satisfies (Ind-1), (Ind-2), and (Ind-3), and $\mathfrak{t}$ is a valid $\xhookrightarrow{e-1}_{\mathcal{P}}$-RT for every $(p_\mathfrak{t}, \mathfrak{t}) \in M$. Then $M$ is our desired set, and we define $\mathbb{T}_H = M$. Moreover, in the end, (Prop-1) and (Prop-2) follow from (Ind-1) and (Ind-3).
We start with $M = \{(1, \mathfrak{T}'_H)\}$. Here, all three properties (Ind-1)–(Ind-3) are satisfied. Next, assume that there is still a pair $(p_\mathfrak{t}, \mathfrak{l}) \in M$ such that $\mathfrak{l}$ contains a node $v \in N_1$ that is not a leaf in $\mathfrak{l}$. We will now split $\mathfrak{l}$ into multiple trees that do not contain $v$ anymore but move directly to one of its children, as illustrated in Figure 5.5.
First, assume that $v$ is not the root of $\mathfrak{l}$. Let $vE^\mathfrak{l} = \{w_1, \dots, w_m\}$ be the direct successors of $v$ in $\mathfrak{l}$ and let $z$ be the predecessor of $v$ in $\mathfrak{l}$. Instead of one tree $\mathfrak{l}$ with the edges $(z, v), (v, w_1), \dots, (v, w_m)$, we split the tree into $m$ different trees $\mathfrak{l}_1, \dots, \mathfrak{l}_m$ such that for every $1 \le h \le m$, the tree $\mathfrak{l}_h$ contains a direct edge from $z$ to $w_h$. In addition to that, the unreachable nodes are removed, and we also have to adjust the probabilities of all (not necessarily direct) successors of $w_h$ (including $w_h$ itself) in $\mathfrak{l}_h$.

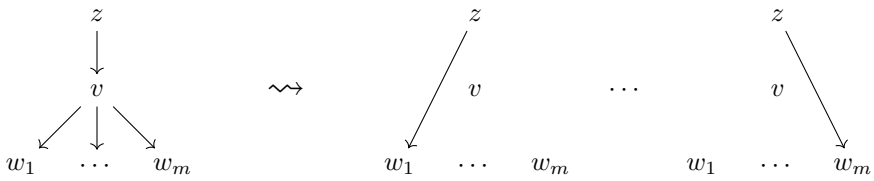



Figure 5.5: Skipping inner node $v \in N_1$ to create $m$ different trees.

More precisely, we set $\mathfrak{l}_h = (N^{\mathfrak{l}_h}, E^{\mathfrak{l}_h})$, with

$$N^{\mathfrak{l}_h} = \big(N^\mathfrak{l} \setminus v(E^\mathfrak{l})^*\big) \cup w_h(E^\mathfrak{l})^*$$
$$E^{\mathfrak{l}_h} = \big(E^\mathfrak{l} \setminus (N^\mathfrak{l} \times v(E^\mathfrak{l})^*)\big) \cup \{(z, w_h)\} \cup \big(E^\mathfrak{l} \cap (w_h(E^\mathfrak{l})^* \times w_h(E^\mathfrak{l})^*)\big).$$

Furthermore, let $p_h = \frac{p^\mathfrak{l}_{w_h}}{p^\mathfrak{l}_v}$. Then, the labeling $L^{\mathfrak{l}_h}(x) = (p_x^{\mathfrak{l}_h}, t_x^{\mathfrak{l}_h})$ is defined by

$$L^{\mathfrak{l}_h}(x) = \begin{cases} (\frac{1}{p_h} \cdot p_x^\mathfrak{l}, t_x^\mathfrak{l}), & \text{if } x \in w_h(E^\mathfrak{l})^*, \\ (p_x^\mathfrak{l}, t_x^\mathfrak{l}), & \text{otherwise.} \end{cases}$$

Note that

$$\sum_{1 \le h \le m} p_h = \sum_{1 \le h \le m} \frac{p^\mathfrak{l}_{w_h}}{p^\mathfrak{l}_v} = \frac{1}{p^\mathfrak{l}_v} \cdot \sum_{1 \le h \le m} p^\mathfrak{l}_{w_h} \overset{\text{(Ind-2)}}{=} \frac{1}{p^\mathfrak{l}_v} \cdot p^\mathfrak{l}_v = 1. \tag{5.6}$$

If $v$ is the root of $\mathfrak{l}$, then we use the same construction, but we have no predecessor $z$ of $v$ and directly start with the node $w_h$ as the new root. Hence, we have to use the node set $N^{\mathfrak{l}_h} = w_h(E^\mathfrak{l})^*$ and edge relation $E^{\mathfrak{l}_h} = (E^\mathfrak{l} \cap (w_h(E^\mathfrak{l})^* \times w_h(E^\mathfrak{l})^*))$ and the rest stays the same.
In the end, we set

$$M' = M \setminus \{(p_\mathfrak{t}, \mathfrak{l})\} \cup \{(p_\mathfrak{t} \cdot p_1, \mathfrak{l}_1), \dots, (p_\mathfrak{t} \cdot p_m, \mathfrak{l}_m)\}.$$

It remains to prove that (Ind-1), (Ind-2), and (Ind-3) are still satisfied for $M'$.

▶ Proof of (Ind-1): We have

$$\begin{aligned}
&\textstyle\sum_{(p_\mathfrak{t},\mathfrak{t})\in M'} p_\mathfrak{t}\\
&= \textstyle\sum_{(p_\mathfrak{t},\mathfrak{t})\in M\setminus\{(p_\mathfrak{l},\mathfrak{l})\}} p_\mathfrak{t} + \sum_{(p_\mathfrak{t},\mathfrak{t})\in\{(p_\mathfrak{l}\cdot p_1,\mathfrak{l}_1),\dots,(p_\mathfrak{l}\cdot p_m,\mathfrak{l}_m)\}} p_\mathfrak{t}\\
&= \textstyle\sum_{(p_\mathfrak{t},\mathfrak{t})\in M\setminus\{(p_\mathfrak{l},\mathfrak{l})\}} p_\mathfrak{t} + \sum_{1\le h\le m} p_\mathfrak{l}\cdot p_h\\
&= \textstyle\sum_{(p_\mathfrak{t},\mathfrak{t})\in M\setminus\{(p_\mathfrak{l},\mathfrak{l})\}} p_\mathfrak{t} + p_\mathfrak{l}\cdot\sum_{1\le h\le m} p_h\\
&\quad\downarrow \quad \text{(by (5.6))}\\
&= \textstyle\sum_{(p_\mathfrak{t},\mathfrak{t})\in M\setminus\{(p_\mathfrak{l},\mathfrak{l})\}} p_\mathfrak{t} + p_\mathfrak{l}\cdot 1\\
&= \textstyle\sum_{(p_\mathfrak{t},\mathfrak{t})\in M\setminus\{(p_\mathfrak{l},\mathfrak{l})\}} p_\mathfrak{t} + p_\mathfrak{l}\\
&= \textstyle\sum_{(p_\mathfrak{t},\mathfrak{t})\in M} p_\mathfrak{t}\\
&\overset{\text{IH}}{=} 1.
\end{aligned}$$

▶ Proof of (Ind-2): Let $1 \le h \le m$. We only split nodes whenever we would have rewritten on or below position $e$. Hence, we only have to prove that in this case the probabilities of the successors for a node add up to the probability for the node itself with our new labeling. We constructed $\mathfrak{l}_h$ by skipping the node $v$ and directly moving from $z$ to $w_h$ (or starting with $w_h$ if $v$ was the root node). Hence, $(N^{\mathfrak{l}_h}, E^{\mathfrak{l}_h})$ is still a finitely branching tree. Let $x \in N^{\mathfrak{l}_h}$ with $xE^{\mathfrak{l}_h} \neq \varnothing$. If $x \in w_h(E^\mathfrak{l})^*$, then $xE^{\mathfrak{l}_h} = xE^\mathfrak{l}$ and thus

$$\sum_{y\in xE^{\mathfrak{l}_h}} p_y^{\mathfrak{l}_h} = \sum_{y\in xE^{\mathfrak{l}_h}} \tfrac{1}{p_h}\cdot p_y^\mathfrak{l} = \tfrac{1}{p_h}\cdot\sum_{y\in xE^{\mathfrak{l}_h}} p_y^\mathfrak{l} = \tfrac{1}{p_h}\cdot\sum_{y\in xE^\mathfrak{l}} p_y^\mathfrak{l} = \tfrac{1}{p_h}\cdot p_x^\mathfrak{l} = p_x^{\mathfrak{l}_h}.$$

If $v$ was not the root and $x = z$, then $zE^{\mathfrak{l}_h} = (zE^\mathfrak{l}\setminus\{v\})\cup\{w_h\}$ and thus

$$\begin{aligned}
\sum_{y\in zE^{\mathfrak{l}_h}} p_y^{\mathfrak{l}_h} &= \sum_{y\in(zE^\mathfrak{l}\setminus\{v\})\cup\{w_h\}} p_y^{\mathfrak{l}_h} = \sum_{y\in(zE^\mathfrak{l}\setminus\{v\})} p_y^{\mathfrak{l}_h} + p_{w_h}^{\mathfrak{l}_h} = \sum_{y\in(zE^\mathfrak{l}\setminus\{v\})} p_y^\mathfrak{l} + \tfrac{1}{p_h}\cdot p_{w_h}^\mathfrak{l}\\
&= \sum_{y\in(zE^\mathfrak{l}\setminus\{v\})} p_y^\mathfrak{l} + \tfrac{p_v^\mathfrak{l}}{p_{w_h}^\mathfrak{l}}\cdot p_{w_h}^\mathfrak{l} = \sum_{y\in(zE^\mathfrak{l}\setminus\{v\})} p_y^\mathfrak{l} + p_v^\mathfrak{l} = \sum_{y\in zE^\mathfrak{l}} p_y^\mathfrak{l} = p_z^\mathfrak{l} = p_z^{\mathfrak{l}_h}.
\end{aligned}$$

Otherwise, we have $x \in N^\mathfrak{l}\setminus(v(E^\mathfrak{l})^*\cup\{z\})$. This means $p_y^{\mathfrak{l}_h} = p_y^\mathfrak{l}$ for all $y \in xE^{\mathfrak{l}_h}$ and $xE^{\mathfrak{l}_h} = xE^\mathfrak{l}$, and thus

$$\sum_{y\in xE^{\mathfrak{l}_h}} p_y^{\mathfrak{l}_h} = \sum_{y\in xE^{\mathfrak{l}_h}} p_y^\mathfrak{l} = \sum_{y\in xE^\mathfrak{l}} p_y^\mathfrak{l} = p_x^\mathfrak{l} = p_x^{\mathfrak{l}_h}.$$

Note that if $v$ is not the root in $\mathfrak{l}$, then the root and its labeling did not change, so that we have $p_{\mathfrak{r}^{\mathfrak{l}_h}}^{\mathfrak{l}_h} = p_{\mathfrak{r}^\mathfrak{l}}^\mathfrak{l} = 1$, where $\mathfrak{r}^{\mathfrak{l}_h} = \mathfrak{r}^\mathfrak{l}$ is the root of $\mathfrak{l}_h$ and $\mathfrak{l}$. If $v$ was the root, then $w_h$ is the new root with

$$p_{w_h}^{\mathfrak{l}_h} = \frac{1}{p_h}\cdot p_{w_h}^\mathfrak{l} = \frac{p_v^\mathfrak{l}}{p_{w_h}^\mathfrak{l}}\cdot p_{w_h}^\mathfrak{l} = p_v^\mathfrak{l} = 1.$$

▶ Proof of (Ind-3): Let $x \in \text{Leaf}^{\mathfrak{T}'_H} \cup (N_2 \cap N^{\mathfrak{T}'_H})$. If we have $x \notin N^{\mathfrak{l}}$, then also $x \notin N^{\mathfrak{l}_h}$ for all $1 \leq h \leq m$ and thus

$$\begin{aligned}
& \textstyle\sum_{(p_\mathfrak{t},\mathfrak{t})\in M'} p_\mathfrak{t} \cdot p_x^\mathfrak{t} \\
= \; & \textstyle\sum_{(p_\mathfrak{t},\mathfrak{t})\in M\setminus\{(p_\mathfrak{l},\mathfrak{l})\}\cup\{(p_\mathfrak{l}\cdot p_1,\mathfrak{l}_1),\ldots,(p_\mathfrak{l}\cdot p_m,\mathfrak{l}_m)\}} p_\mathfrak{t} \cdot p_x^\mathfrak{t} \\
= \; & \textstyle\sum_{(p_\mathfrak{t},\mathfrak{t})\in M\setminus\{(p_\mathfrak{l},\mathfrak{l})\}} p_\mathfrak{t} \cdot p_x^\mathfrak{t} + \sum_{(p_\mathfrak{t},\mathfrak{t})\in\{(p_\mathfrak{l}\cdot p_1,\mathfrak{l}_1),\ldots,(p_\mathfrak{l}\cdot p_m,\mathfrak{l}_m)\}} p_\mathfrak{t} \cdot p_x^\mathfrak{t} \\
= \; & \textstyle\sum_{(p_\mathfrak{t},\mathfrak{t})\in M\setminus\{(p_\mathfrak{l},\mathfrak{l})\}} p_\mathfrak{t} \cdot p_x^\mathfrak{t} + \sum_{(p_\mathfrak{t},\mathfrak{t})\in\{(p_\mathfrak{l}\cdot p_1,\mathfrak{l}_1),\ldots,(p_\mathfrak{l}\cdot p_m,\mathfrak{l}_m)\}} p_\mathfrak{t} \cdot 0 \\
= \; & \textstyle\sum_{(p_\mathfrak{t},\mathfrak{t})\in M\setminus\{(p_\mathfrak{l},\mathfrak{l})\}} p_\mathfrak{t} \cdot p_x^\mathfrak{t} \\
= \; & \textstyle\sum_{(p_\mathfrak{t},\mathfrak{t})\in M\setminus\{(p_\mathfrak{l},\mathfrak{l})\}} p_\mathfrak{t} \cdot p_x^\mathfrak{t} + p_\mathfrak{l} \cdot 0 \\
= \; & \textstyle\sum_{(p_\mathfrak{t},\mathfrak{t})\in M\setminus\{(p_\mathfrak{l},\mathfrak{l})\}} p_\mathfrak{t} \cdot p_x^\mathfrak{t} + p_\mathfrak{l} \cdot p_x^\mathfrak{l} \\
= \; & \textstyle\sum_{(p_\mathfrak{t},\mathfrak{t})\in M} p_\mathfrak{t} \cdot p_x^\mathfrak{t} \\
\overset{\text{IH}}{=} \; & p_x^{\mathfrak{T}'_H}.
\end{aligned}$$

If we have $x \in N^{\mathfrak{l}}$ and $x \notin v(E^{\mathfrak{l}})^*$, then $x \in N^{\mathfrak{l}_h}$ for all $1 \leq h \leq m$ and $p_x^{\mathfrak{l}} = p_x^{\mathfrak{l}_h}$. Hence

$$\begin{aligned}
& \textstyle\sum_{(p_\mathfrak{t},\mathfrak{t})\in M'} p_\mathfrak{t} \cdot p_x^\mathfrak{t} \\
= \; & \textstyle\sum_{(p_\mathfrak{t},\mathfrak{t})\in M\setminus\{(p_\mathfrak{l},\mathfrak{l})\}\cup\{(p_\mathfrak{l}\cdot p_1,\mathfrak{l}_1),\ldots,(p_\mathfrak{l}\cdot p_m,\mathfrak{l}_m)\}} p_\mathfrak{t} \cdot p_x^\mathfrak{t} \\
= \; & \textstyle\sum_{(p_\mathfrak{t},\mathfrak{t})\in M\setminus\{(p_\mathfrak{l},\mathfrak{l})\}} p_\mathfrak{t} \cdot p_x^\mathfrak{t} + \sum_{(p_\mathfrak{t},\mathfrak{t})\in\{(p_\mathfrak{l}\cdot p_1,\mathfrak{l}_1),\ldots,(p_\mathfrak{l}\cdot p_m,\mathfrak{l}_m)\}} p_\mathfrak{t} \cdot p_x^\mathfrak{t} \\
= \; & \textstyle\sum_{(p_\mathfrak{t},\mathfrak{t})\in M\setminus\{(p_\mathfrak{l},\mathfrak{l})\}} p_\mathfrak{t} \cdot p_x^\mathfrak{t} + \sum_{1\leq h\leq m} p_\mathfrak{l} \cdot p_h \cdot p_x^{\mathfrak{l}_h} \\
= \; & \textstyle\sum_{(p_\mathfrak{t},\mathfrak{t})\in M\setminus\{(p_\mathfrak{l},\mathfrak{l})\}} p_\mathfrak{t} \cdot p_x^\mathfrak{t} + \sum_{1\leq h\leq m} p_\mathfrak{l} \cdot p_h \cdot p_x^{\mathfrak{l}} \\
= \; & \textstyle\sum_{(p_\mathfrak{t},\mathfrak{t})\in M\setminus\{(p_\mathfrak{l},\mathfrak{l})\}} p_\mathfrak{t} \cdot p_x^\mathfrak{t} + p_\mathfrak{l} \cdot p_x^{\mathfrak{l}} \cdot \sum_{1\leq h\leq m} p_h \\
= \; & \textstyle\sum_{(p_\mathfrak{t},\mathfrak{t})\in M\setminus\{(p_\mathfrak{l},\mathfrak{l})\}} p_\mathfrak{t} \cdot p_x^\mathfrak{t} + p_\mathfrak{l} \cdot p_x^{\mathfrak{l}} \cdot 1 \\
= \; & \textstyle\sum_{(p_\mathfrak{t},\mathfrak{t})\in M} p_\mathfrak{t} \cdot p_x^\mathfrak{t} \\
\overset{\text{IH}}{=} \; & p_x^{\mathfrak{T}'_H}.
\end{aligned}$$

Otherwise we have $x \in N^{\mathfrak{l}}$ and $x \in v(E^{\mathfrak{l}})^*$. This means that we have $x \in N^{\mathfrak{l}_h}$ and $x \in w_h(E^{\mathfrak{l}})^*$ for some $1 \leq h \leq m$ and $x \notin N^{\mathfrak{l}_{h'}}$ for all $h' \neq h$. Furthermore, we have $p_x^{\mathfrak{l}_h} = \frac{1}{p_h} \cdot p_x^{\mathfrak{l}}$, and hence

$$\begin{aligned}
& \textstyle\sum_{(p_\mathfrak{t},\mathfrak{t})\in M'} p_\mathfrak{t} \cdot p_x^\mathfrak{t} \\
= \; & \textstyle\sum_{(p_\mathfrak{t},\mathfrak{t})\in M\setminus\{(p_\mathfrak{l},\mathfrak{l})\}\cup\{(p_\mathfrak{l}\cdot p_1,\mathfrak{l}_1),\ldots,(p_\mathfrak{l}\cdot p_m,\mathfrak{l}_m)\}} p_\mathfrak{t} \cdot p_x^\mathfrak{t} \\
= \; & \textstyle\sum_{(p_\mathfrak{t},\mathfrak{t})\in M\setminus\{(p_\mathfrak{l},\mathfrak{l})\}} p_\mathfrak{t} \cdot p_x^\mathfrak{t} + \sum_{(p_\mathfrak{t},\mathfrak{t})\in\{(p_\mathfrak{l}\cdot p_1,\mathfrak{l}_1),\ldots,(p_\mathfrak{l}\cdot p_m,\mathfrak{l}_m)\}} p_\mathfrak{t} \cdot p_x^\mathfrak{t} \\
= \; & \textstyle\sum_{(p_\mathfrak{t},\mathfrak{t})\in M\setminus\{(p_\mathfrak{l},\mathfrak{l})\}} p_\mathfrak{t} \cdot p_x^\mathfrak{t} + p_\mathfrak{l} \cdot p_h \cdot p_x^{\mathfrak{l}_h} \\
= \; & \textstyle\sum_{(p_\mathfrak{t},\mathfrak{t})\in M\setminus\{(p_\mathfrak{l},\mathfrak{l})\}} p_\mathfrak{t} \cdot p_x^\mathfrak{t} + p_\mathfrak{l} \cdot p_h \cdot \frac{1}{p_h} \cdot p_x^{\mathfrak{l}} \\
= \; & \textstyle\sum_{(p_\mathfrak{t},\mathfrak{t})\in M\setminus\{(p_\mathfrak{l},\mathfrak{l})\}} p_\mathfrak{t} \cdot p_x^\mathfrak{t} + p_\mathfrak{l} \cdot p_x^{\mathfrak{l}} \\
= \; & \textstyle\sum_{(p_\mathfrak{t},\mathfrak{t})\in M} p_\mathfrak{t} \cdot p_x^\mathfrak{t} \\
\overset{\text{IH}}{=} \; & p_x^{\mathfrak{T}'_H}.
\end{aligned}$$

**Step 3** Create the finitely branching infinite tree $\mathbf{T}$ of RTs

Next, we create a tree $\mathbf{T}$ whose nodes are labeled with RTs. More precisely, its nodes at depth $H$ represent $\xrightarrow{e-1}_{\mathcal{P}}$-RTs $\mathfrak{t}$ with $(p_\mathfrak{t}, \mathfrak{t}) \in \mathbb{T}_H$ where the leaves that are also leaves of $\mathfrak{T}'$ have a small enough total probability, i.e., $\sum_{x\in\text{Leaf}^\mathfrak{t} \wedge x\in\text{Leaf}^{\mathfrak{T}'}} p_x^\mathfrak{t} < 1-\alpha$. The root of $\mathbf{T}$ is labeled with the subtree $\mathfrak{T}'_0$ of $\mathfrak{T}'$ that only consists of its root. $\mathfrak{T}'_0$ is a finite subtree and $\sum_{x\in\text{Leaf}^{\mathfrak{T}'_0} \wedge x\in\text{Leaf}^{\mathfrak{T}'}} p_x^{\mathfrak{T}'_0} = 0 < 1-\alpha$, since the root of $\mathfrak{T}'$ cannot be a leaf in $\mathfrak{T}'$.

Let $H \in \mathbb{N}$ with $H \geq 1$. For the tree $\mathbf{T}$, we have a node at depth $H$ for every pair $(p_\mathfrak{t}, \mathfrak{t}) \in \mathbb{T}_H$ such that $\sum_{x \in \mathrm{Leaf}^\mathfrak{t} \wedge x \in \mathrm{Leaf}^{\mathfrak{T}'}} p_x^\mathfrak{t} < 1 - \alpha$. We draw an edge from a node $X$ at depth $H-1$ to the node $Y$ at depth $H$ if the corresponding RTs in the labels are the same, or if the RT for node $Y$ is an extension of the RT for node $X$ (i.e., we evaluate the leaves in the RT of $X$ further to obtain $Y$).
Each $\mathbb{T}_H$ is finite, so $\mathbf{T}$ is finitely branching. Furthermore, there is a node in each layer of the tree $\mathbf{T}$, since for every $H \in \mathbb{N}$, we can find an $(p_\mathfrak{t}, \mathfrak{t}) \in \mathbb{T}_H$ such that $\sum_{x \in \mathrm{Leaf}^\mathfrak{t} \wedge x \in \mathrm{Leaf}^{\mathfrak{T}'}} p_x^\mathfrak{t} < 1 - \alpha$, see (5.5).

**Step 4** Use König's Lemma

Now we have an infinite tree $\mathbf{T}$ that is finitely branching, which means that the tree has an infinite path by König's Lemma. (As a reminder, the lemma of König states that every finitely branching tree that is infinite must contain an infinite path.) This path represents an $\xhookrightarrow{e-1}_\mathcal{P}$-RT that does not converge with probability 1, which is our desired contradiction. To see this, let $\mathfrak{t}_1, \mathfrak{t}_2, \ldots$ be the finite $\xhookrightarrow{e-1}_\mathcal{P}$-RTs in the labels of the nodes in the infinite path in $\mathbf{T}$, where $\mathfrak{t}_i$ is a prefix of $\mathfrak{t}_{i+1}$ for all $i \geq 1$. Hence, we can define the tree $\mathfrak{t}_{\lim} = \lim_{i \to \infty} \mathfrak{t}_i$. Here, a leaf of $\mathfrak{t}_i$ that is an inner node of $\mathfrak{T}'$ from $N_1$ is replaced by one of its successors (with the same label and probability) in $\mathfrak{t}_{i+1}$. Since there are no infinite paths in $\mathfrak{T}'$ that eventually only contain nodes from $N_1$ (due to the application of the Sequential Execution Lemma to create $\mathfrak{T}'$), this happens only finitely often for every leaf. Hence, $\mathfrak{t}_{\lim}$ is a well-defined $\xhookrightarrow{e-1}_\mathcal{P}$-RT, and its leaves are precisely those nodes that are leaves of $\mathfrak{T}'$ and of some $\mathfrak{t}_i$ (and then of all $\mathfrak{t}_j$ with $j \geq i$, with the same probability). Finally, we have $|\mathfrak{t}_{\lim}| = \lim_{i \to \infty} \sum_{x \in \mathrm{Leaf}^{\mathfrak{t}_i} \wedge x \in \mathrm{Leaf}^{\mathfrak{T}'}} p_x^{\mathfrak{t}_i} \leq 1 - \alpha < 1$ since $\sum_{x \in \mathrm{Leaf}^{\mathfrak{t}_i} \wedge x \in \mathrm{Leaf}^{\mathfrak{T}'}} p_x^{\mathfrak{t}_i} < 1 - \alpha$ holds for all $i \in \mathbb{N}$, which is our desired contradiction. *QED*

Example 5.2.3. Consider the PTRS $\mathcal{P} = \mathcal{P}_{\mathsf{coin}}$ from Example 3.4.2 with the only rule $\mathsf{g}(x) \to \{1/2 x, 1/2 \mathsf{g}(x)\}$ that models a fair coin flip. The $\rightrightarrows_\mathcal{P}$-sequence $\{1\mathsf{g}(x)\} \rightrightarrows_\mathcal{P} \{1/2 x, 1/2 \mathsf{g}(x)\} \rightrightarrows_\mathcal{P} \{1/2 x, 1/4 x, 1/4 \mathsf{g}(x)\} \rightrightarrows_\mathcal{P} \cdots$ that starts with $\mathsf{g}(x)$ converges with probability 1. For example, after a single rewrite step, there is a probability of $1/2$ that we reach a normal form.

Let us now consider the start term $\mathsf{c}(\mathsf{g}(x), \mathsf{g}(x))$ for a fresh symbol $\mathsf{c}$ and the $\rightrightarrows_\mathcal{P}$-sequence $\{1\mathsf{c}(\mathsf{g}(x), \mathsf{g}(x))\} \rightrightarrows_\mathcal{P} \{1/2 \mathsf{c}(x, \mathsf{g}(x)), 1/2 \mathsf{c}(\mathsf{g}(x), \mathsf{g}(x))\} \rightrightarrows_\mathcal{P} \{1/4 \mathsf{c}(x, x), 1/4 \mathsf{c}(x, \mathsf{g}(x)), 1/4 \mathsf{c}(x, \mathsf{g}(x)), 1/4 \mathsf{c}(\mathsf{g}(x), \mathsf{g}(x))\} \rightrightarrows_\mathcal{P} \cdots$. After a single $\rightrightarrows_\mathcal{P}$-step, we do not reach a normal form anymore. Instead, we have to perform two $\rightrightarrows_\mathcal{P}$-steps to reach a normal form and this only happens with a probability of $1/4$. If we want to get a chance of $1/2$ to terminate, then we have to perform a total of three $\rightrightarrows_\mathcal{P}$-steps. Even though the expected runtime increases (because we have to terminate in both executions), the Parallel Execution Lemma shows that the $\rightrightarrows_\mathcal{P}$-sequence converges with probability 1 again.

While the termination probability remains 1 if we combine multiple almost surely terminating terms $q_i$ in a single term $\mathsf{c}(q_1, \ldots, q_n)$, the expected derivation height may increase. If all the terms $q_i$ have an expected derivation height of $C_i$, then the sum $\sum_{i=1}^n C_i$ over all of them yields an upper bound on the expected derivation height of $\mathsf{c}(q_1, \ldots, q_n)$. Thus, we obtain a Parallel Execution Lemma for SAST as well.

Lemma 5.2.4 – Parallel Execution Lemma for SAST.
*Let $\mathcal{P}$ be a finite PTRS, and let $q_1, \dots, q_n \in \mathcal{T}$ be terms and $C_1, \dots, C_n \in \mathbb{R}$ constants such that for every $1 \leq i \leq n$ and every $\to_{\mathcal{P}}$-RT $\mathfrak{T}_i$ that starts with $q_i$ we have $\operatorname{edl}(\mathfrak{T}_i) \leq C_i$. Then, every $\to_{\mathcal{P}}$-RT $\mathfrak{T}$ that starts with $\mathtt{c}(q_1, \dots, q_n)$ for a fresh symbol $\mathtt{c}$ has a finite expected derivation length, which is bounded by $\sum_{i=1}^{n} C_i \in \mathbb{R}$, i.e., $\operatorname{edl}(\mathfrak{T}) \leq \sum_{i=1}^{n} C_i$.*

*Proof.* We use the same induction on $e$ as in the proof for the Parallel Execution Lemma for AST (Lemma 5.2.2). Note that we now need to prove an upper bound (the expected derivation length is at most $\sum_{i=1}^{n} C_i$), while for AST we had to prove a lower bound (probability of convergence is at least 1). Therefore, we do not use the Sequential Execution Lemma and the proof is a bit different in the induction step, while the base case is again trivial.
In the induction step, we assume that the statement holds for $e-1$, i.e., there exists a bound $C'_{e-1} = \sum_{i=1}^{e-1} C_i \in \mathbb{R}$ such that for every $\xhookrightarrow{e-1}_{\mathcal{P}}$-RT $\mathfrak{T}$ that starts with $\mathtt{c}(q_1, \dots, q_n)$ we have $\operatorname{edl}(\mathfrak{T}) \leq C'_{e-1}$. Now consider an arbitrary $\xhookrightarrow{e}_{\mathcal{P}}$-RT $\mathfrak{T}$ that starts with $\mathtt{c}(q_1, \dots, q_n)$. We prove that its expected derivation length is bounded by $C'_{e-1} + C_e$.
Let $H \in \mathbb{N}$ and let $\mathfrak{T}_H$ be the tree consisting of all nodes of $\mathfrak{T}$ with depth at most $H$. We partition the inner nodes of $\mathfrak{T}_H$ into $N_1$ and $N_2$, analogous to the proof of the Parallel Execution Lemma for AST (Lemma 5.2.2). As in that proof, for each $H \in \mathbb{N}$ we split the tree $\mathfrak{T}_H$ into multiple (finitely many) $\xhookrightarrow{e-1}_{\mathcal{P}}$-RTs of height at most $H$ that all start with $\mathtt{c}(q_1, \dots, q_n)$. This again leads to a set of pairs $\mathbb{T}_H$ (using the same notation as before) with the following two properties.

(PropS-1) $\sum_{(p_\mathfrak{t}, \mathfrak{t}) \in \mathbb{T}_H} p_\mathfrak{t} = 1$.

(PropS-2) For all $x \in N_2$ we have $p_x^{\mathfrak{T}_H} = \sum_{(p_\mathfrak{t}, \mathfrak{t}) \in \mathbb{T}_H} p_\mathfrak{t} \cdot p_x^\mathfrak{t}$.

Furthermore, we can use the same construction to create a set $\mathbb{T}_H^e$ containing (finitely many) pairs of probabilities and $\to_{\mathcal{P}}$-RTs that start with $q_e$ of height at most $H$, by simply switching when to split the tree and when to perform the rewrite step. To be precise, we split the tree if we encounter a rewrite step on or below a position $1 \leq j \leq e-1$, and perform the rewrite step if it is on or below position $e$. Again, we get

(PropS-1-e) $\sum_{(p_\mathfrak{t}, \mathfrak{t}) \in \mathbb{T}_H^e} p_\mathfrak{t} = 1$.

(PropS-2-e) For all $x \in N_1$ we have $p_x^{\mathfrak{T}_H} = \sum_{(p_\mathfrak{t}, \mathfrak{t}) \in \mathbb{T}_H^e} p_\mathfrak{t} \cdot p_x^\mathfrak{t}$.

We can bound the expected derivation length after height $H$ by $C'_{e-1} + C_e$ for every

$H \in \mathbb{N}$, as

$$
\begin{aligned}
& \mathrm{edl}(\mathfrak{T}_H) \\
= \;& \textstyle\sum_{x \in N^{\mathfrak{T}_H} \setminus \mathrm{Leaf}^{\mathfrak{T}_H}} p_x^{\mathfrak{T}_H} \\
& \quad \downarrow \quad \text{(since } N_1 \uplus N_2 = N^{\mathfrak{T}_H} \setminus \mathrm{Leaf}^{\mathfrak{T}_H}\text{)} \\
= \;& \textstyle\sum_{x \in N_1} p_x^{\mathfrak{T}_H} \quad + \quad \sum_{x \in N_2} p_x^{\mathfrak{T}_H} \\
& \quad \downarrow \quad \text{(by (PropS-2-e) and (PropS-2))} \\
= \;& \textstyle\sum_{x \in N_1} \sum_{(p_\mathfrak{t}, \mathfrak{t}) \in \mathbb{T}_H^e} p_\mathfrak{t} \cdot p_x^\mathfrak{t} \quad + \quad \sum_{x \in N_2} \sum_{(p_\mathfrak{t}, \mathfrak{t}) \in \mathbb{T}_H} p_\mathfrak{t} \cdot p_x^\mathfrak{t} \\
& \quad \downarrow \quad \text{(as in (5.4))} \\
= \;& \textstyle\sum_{(p_\mathfrak{t}, \mathfrak{t}) \in \mathbb{T}_H^e} p_\mathfrak{t} \cdot \sum_{x \in N^\mathfrak{t} \setminus \mathrm{Leaf}^\mathfrak{t}} p_x^\mathfrak{t} \quad + \quad \sum_{(p_\mathfrak{t}, \mathfrak{t}) \in \mathbb{T}_H} p_\mathfrak{t} \cdot \sum_{x \in N^\mathfrak{t} \setminus \mathrm{Leaf}^\mathfrak{t}} p_x^\mathfrak{t} \\
& \quad \downarrow \quad \text{(by assumption and induction hypothesis)} \\
\leq \;& \textstyle\sum_{(p_\mathfrak{t}, \mathfrak{t}) \in \mathbb{T}_H^e} p_\mathfrak{t} \cdot C_e \quad + \quad \sum_{(p_\mathfrak{t}, \mathfrak{t}) \in \mathbb{T}_H} p_\mathfrak{t} \cdot C'_{e-1} \\
= \;& C_e \cdot \textstyle\sum_{(p_\mathfrak{t}, \mathfrak{t}) \in \mathbb{T}_H^e} p_\mathfrak{t} \quad + \quad C'_{e-1} \cdot \sum_{(p_\mathfrak{t}, \mathfrak{t}) \in \mathbb{T}_H} p_\mathfrak{t} \\
& \quad \downarrow \quad \text{(by (PropS-1-e) and (PropS-1))} \\
= \;& C_e \cdot 1 + C'_{e-1} \cdot 1 \\
= \;& C'_{e-1} + C_e.
\end{aligned}
$$

This gives us $\mathrm{edl}(\mathfrak{T}) = \lim_{H \to \infty} \mathrm{edl}(\mathfrak{T}_H) \leq C'_{e-1} + C_e$, as desired. *QED*

Example 5.2.5. Reconsider the PTRS $\mathcal{P} = \mathcal{P}_{\mathsf{coin}}$ with the rule $\mathsf{g}(x) \to \{{}^1\!/\!_2\, x, {}^1\!/\!_2\, \mathsf{g}(x)\}$ that models a coin flip from Example 5.2.3. In the only $\rightrightarrows_\mathcal{P}$-sequence that starts with $\mathsf{g}(x)$ (see Example 5.2.3), we reach a normal form in each step with probability $1/2$, so we have $\mathrm{edh}_{\to_\mathcal{P}}(\mathsf{g}(x)) = \sum_{n=0}^{\infty} (1/2)^n = 2$. Using the Parallel Execution Lemma for SAST we obtain a bound on the expected derivation height of $\mathsf{c}(\mathsf{g}(x), \mathsf{g}(x))$ by $\mathrm{edh}_{\to_\mathcal{P}}(\mathsf{c}(\mathsf{g}(x), \mathsf{g}(x))) \leq \mathrm{edh}_{\to_\mathcal{P}}(\mathsf{g}(x)) + \mathrm{edh}_{\to_\mathcal{P}}(\mathsf{g}(x)) = 2 + 2 = 4$. More generally, we obtain the bound $2n$ on the expected derivation height of the term $\mathsf{c}(\underbrace{\mathsf{g}(x), \ldots, \mathsf{g}(x)}_{n \text{ times}})$.

Once again, the Parallel Execution Lemma for both AST and SAST can easily be adapted to other evaluation strategies like innermost or leftmost-innermost rewriting. Moreover, since $\mathsf{c}$ is fresh, the $\to_\mathcal{P}$-RTs that start with $\mathsf{c}(q_1, \ldots, q_n)$ are precisely the $\xrightarrow{\neg\varepsilon}_\mathcal{P}$-RTs that start with $\mathsf{c}(q_1, \ldots, q_n)$. As the root symbol does not influence rewrite steps below the root, both lemmas (and their innermost variants) also hold for all $\xrightarrow{\neg\varepsilon}_\mathcal{P}$-RTs ($\xrightarrow{\mathrm{i}\neg\varepsilon}_\mathcal{P}$-RTs) that start with $f(q_1, \ldots, q_n)$ for an arbitrary function symbol $f$ of arity $n$.

Lastly, we want to mention that the Parallel Execution Lemma does not hold for PAST. If we take a term $t$ that has infinite expected derivation height, but the expected derivation length of every rewrite sequence starting with $t$ is finite, then (as $\mathcal{P}$ is finite) we are able to construct a single rewrite sequence starting with $\mathsf{c}(t, t)$ that has an infinite expected derivation length (as in the proofs of Theorems 3.4.23 and 3.4.24). This is due to the non-determinism that arises with such a parallel execution.

Counterexample 5.2.6. Reconsider the term $\mathsf{f}(\mathsf{0})$ and the PTRS $\mathcal{P}_{\mathsf{unary}}$ from Counterexamples 3.4.18 and 3.4.19. Counterexample 3.4.18 shows that $\mathsf{f}(\mathsf{0})$ is PAST, and Counterexample 3.4.19 shows that the term $\mathsf{c}(\mathsf{f}(\mathsf{0}), \mathsf{f}(\mathsf{0}))$ is not PAST.

| Closed Under … | AST | PAST | SAST |
|---|---|---|---|
| Parallel Execution | Yes | No | Yes |

Table 5.3: Closure of AST, PAST, and SAST under parallel execution.

SECTION 5.3

## Non-Termination Witnesses

Before we continue investigating different ways of combining programs and rewrite systems, we take a look at the witnesses for non-AST, non-PAST, and non-SAST. We deliberately omitted the analysis of such witnesses in Section 3.4, because the results rely on the Sequential Execution Lemma (Lemma 5.1.3) and the Parallel Execution Lemma (Lemma 5.2.2).

First of all, note that a witness for non-AST or non-PAST of a PTRS $\mathcal{P}$ is a single infinite $\rightrightarrows_{\mathcal{P}}$-sequence $\vec{\mu}$ such that $|\vec{\mu}| < 1$ or $\operatorname{edl}(\vec{\mu}) = \infty$, respectively. A witness for non-SAST is an infinite sequence of $\rightrightarrows_{\mathcal{P}}$-sequences $(\vec{\mu}_n)_{n\in\mathbb{N}}$ such that $\lim_{n\to\infty} \operatorname{edl}(\vec{\mu}_n) = \infty$ and all the $\vec{\mu}_n$ start with the same term. However, we show that most witness theorems only work for AST (if they still work at all in the probabilistic setting, see, e.g., Counterexample 5.3.4), and witnesses for the maximal expected derivation height are much more difficult to analyze. Note that already in the non-probabilistic setting, the witness theorems only deal with termination and not with complexity. For complexity, one instead restricts the set of start terms explicitly, as in the runtime complexity, which is often much lower and easier to analyze than the derivational complexity.

First we show that one of the most important properties for termination witnesses of TRSs does not hold in the probabilistic setting anymore, namely the *minimality criterion*. In the non-probabilistic setting, whenever a term $t$ starts an infinite rewrite sequence, then there exists a *minimal* infinite rewrite sequence beginning with $t$, where one only reduces redexes whose proper subterms are strongly normalizing. Such minimal infinite sequences do not always exist in the probabilistic setting for PTRSs that are not AST, not PAST, or not SAST.

Counterexample 5.3.1 (No Minimal Witnesses for non-AST, non-PAST, and non-SAST). Consider the PTRS $\mathcal{P}_{\mathsf{min}}$ containing the two rules $\mathsf{g} \to \{^{3/4}\mathsf{d}(\mathsf{g}), {}^{1/4}\mathsf{a}\}$ and $\mathsf{d}(\mathsf{g}) \to \{^{1}\mathsf{c}(\mathsf{g},\mathsf{g})\}$. We have $\neg\mathtt{AST}(\to_{\mathcal{P}_{\mathsf{min}}})$ (and hence $\neg\mathtt{PAST}(\to_{\mathcal{P}_{\mathsf{min}}})$ and $\neg\mathtt{SAST}(\to_{\mathcal{P}_{\mathsf{min}}})$), because if we always rewrite the outer $\mathsf{d}$ after rewriting the $\mathsf{g}$ we obtain $\{^{1}\mathsf{g}\} \rightrightarrows^{2}_{\mathcal{P}_{\mathsf{min}}} \{^{3/4}\mathsf{c}(\mathsf{g},\mathsf{g}), {}^{1/4}\mathsf{a}\}$, which corresponds to the positively drifting random walk program from Figure 5.6.

However, the $\mathsf{d}$-rule is never applicable in innermost evaluations, since the argument $\mathsf{g}$ of its left-hand side is a redex. To see that we have $\mathtt{SAST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}_{\mathsf{min}}})$ (and hence $\mathtt{PAST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}_{\mathsf{min}}})$ and $\mathtt{AST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}_{\mathsf{min}}})$), consider the only possible innermost rewrite sequence $\vec{\mu}$ starting with $\mathsf{g}$, namely

$$\{^{1}\mathsf{g}\} \overset{\mathtt{i}}{\rightrightarrows}_{\mathcal{P}_{\mathsf{min}}} \{^{3/4}\mathsf{d}(\mathsf{g}), {}^{1/4}\mathsf{a}\} \overset{\mathtt{i}}{\rightrightarrows}_{\mathcal{P}_{\mathsf{min}}} \{^{(3/4)^2}\mathsf{d}(\mathsf{d}(\mathsf{g})), {}^{1/4\cdot 3/4}\mathsf{d}(\mathsf{a}), {}^{1/4}\mathsf{a}\} \overset{\mathtt{i}}{\rightrightarrows}_{\mathcal{P}_{\mathsf{min}}} \cdots .$$

We can view this rewrite sequence as an $\xrightarrow{\mathtt{i}}_{\mathcal{P}_{\mathsf{min}}}$-RT, see Figure 5.7. The branches to the right that start with $\mathsf{a}$, $\mathsf{d}(\mathsf{a})$, $\mathsf{d}(\mathsf{d}(\mathsf{a}))$, etc., stop after 0 innermost steps, as the terms are normal forms. Thus, we have $\operatorname{edl}(\vec{\mu}) = \sum_{n=0}^{\infty}(1 - |\mu_n|_{\mathcal{P}_{\mathsf{min}}}) = \sum_{k\in\mathbb{N}} (3/4)^k = 4$. Due to innermost rewriting, there is no other sequence possible. Thus, we have $\operatorname{edh}_{\xrightarrow{\mathtt{i}}_{\mathcal{P}_{\mathsf{min}}}}(\mathsf{g}) = 4$. Analogously, in all other innermost rewrite sequences, the $\mathsf{d}$-rule is never applicable either. Thus, all terms have finite expected derivation height w.r.t. innermost rewriting. Therefore, we have $\mathtt{SAST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}_{\mathsf{min}}})$, and thus, also $\mathtt{PAST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}_{\mathsf{min}}})$ and $\mathtt{AST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}_{\mathsf{min}}})$.

While there exists a $\rightrightarrows_{\mathcal{P}_{\mathsf{min}}}$-sequence with convergence probability $< 1$, there is no minimal rewrite sequence with convergence probability $< 1$. Note that the term $\mathsf{g}$ is

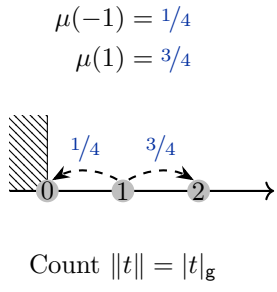



Figure 5.6: Random walk for $\to_{\mathcal{P}_{\mathsf{min}}}$.

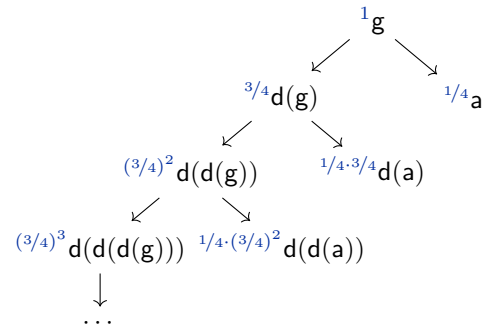



Figure 5.7: $\overset{\mathsf{i}}{\to}_{\mathcal{P}_{\mathsf{min}}}$-RT.

not almost surely terminating, and starts an infinite $\rightrightarrows_{\mathcal{P}_{\mathsf{min}}}$-sequence which converges with probability $< 1$. However, every rewrite sequence that does not use the $\mathsf{d}$-rule converges with probability 1, since then the number of $\mathsf{g}$-symbols never increases, and every rewrite step removes a $\mathsf{g}$-symbol with probability $1/4$. Therefore, a rewrite sequence $\vec{\mu}$ with convergence probability $< 1$ would have to eventually use the $\mathsf{d}$-rule on the redex $\mathsf{d}(\mathsf{g})$. But then $\vec{\mu}$ is not minimal since $\mathsf{g}$ itself starts a rewrite sequence with convergence probability $< 1$.

The same argumentation as for non-$\mathtt{AST}$ above shows that there is also no minimal non-$\mathtt{PAST}$ witness, i.e., no minimal rewrite sequence $\vec{\mu}$ with $\mathrm{edl}(\vec{\mu}) = \infty$. Again, every rewrite sequence that does not use the $\mathsf{d}$-rule has a finite expected derivation length, since then the number of $\mathsf{g}$-symbols never increases and every $\mathsf{g}$-symbol is rewritten at most 4 times in expectation. Thus, we have to rewrite the redex $\mathsf{d}(\mathsf{g})$ eventually, although its argument $\mathsf{g}$ starts a rewrite sequence with infinite expected derivation length. However, then the rewrite sequence is not minimal.

Many important termination techniques for TRSs are based on the minimality property. Therefore, already this seemingly minor counterexample yields hard boundaries for automatic termination techniques in the probabilistic setting.

Nevertheless, we can still create a witness theorem for non-$\mathtt{AST}$ sequences, which helps in fixing structural properties of such infinite sequences. Namely, we can again assume that the initial rewrite step is performed at the root.

Theorem 5.3.2 – Witness for Non-$\mathtt{AST}$.
Let $\mathcal{P}$ be a finite PTRS. If $\neg\mathtt{AST}(\to_{\mathcal{P}})$, then there exists an $\to_{\mathcal{P}}$-RT $\mathfrak{T}$ that converges with probability $< 1$ and starts with $t$ such that $t = \ell\sigma$ for some rule $\ell \to \mu \in \mathcal{P}$ and some substitution $\sigma$, and every proper subterm of $\ell\sigma$ is almost surely terminating w.r.t. $\mathcal{P}$. Moreover, the rewrite step at the root node of $\mathfrak{T}$ is performed at the root position of $t$.

*Proof of Theorem 5.3.2.* Let $\mathcal{T}_{goal}$ be the set of all terms $t$ such that $t = \ell\sigma$ for some rule $\ell \to \mu \in \mathcal{P}$ and some substitution $\sigma$, and every proper subterm of $\ell\sigma$ is almost surely terminating w.r.t. $\mathcal{P}$. We first prove that there exists a term $t \in \mathcal{T}_{goal}$ that starts a $\to_{\mathcal{P}}$-RT $\mathfrak{T}$ with $|\mathfrak{T}| < 1$. *Proof*

**Step 1** There exists $t \in \mathcal{T}_{goal}$ that starts a $\to_\mathcal{P}$-RT converging with probability $< 1$

Let $t \in \mathcal{T}$ be a term such that there exists a $\to_\mathcal{P}$-RT $\mathfrak{T}$ starting with $t$ with $|\mathfrak{T}| < 1$. Let $t' \trianglelefteq t$ be a minimal subterm of $t$ that still starts a $\to_\mathcal{P}$-RT $\mathfrak{T}'$ with $|\mathfrak{T}'| < 1$. Then, every proper subterm of $t'$ is almost surely terminating w.r.t. $\mathcal{P}$. From $t'$ we can reach a term $t'' \in \mathcal{T}_{goal}$ that starts a $\to_\mathcal{P}$-RT converging with probability $< 1$.
The proof is an application of both the Sequential Execution Lemma and the Parallel Execution Lemma. Let $X$ be the set of nodes in $\mathfrak{T}'$ such that $t_x^{\mathfrak{T}'} \in \mathcal{T}_{goal}$. We split the nodes of $\mathfrak{T}' = (N', E')$ into $N_1$ containing all nodes reachable from the root before reaching a node in $X$ and $N_2$ containing all the nodes reachable from a node in $X$. Note that the nodes $x \in N_2$ that have no predecessor in $N_2$ are precisely the first nodes on each path where we have $t_x \in \mathcal{T}_{goal}$.
Assume for a contradiction that no node $x \in N_2$ exists with $t_x \in \mathcal{T}_{goal}$ and $|\mathfrak{T}'[xE'^*]| < 1$. Since every node $y \in N_2$ is reachable from a node $x \in N_2$ with $t_x \in \mathcal{T}_{goal}$, we know that every subtree that only contains inner nodes from $N_2$ converges with probability 1. Applying the Sequential Execution Lemma (Lemma 5.1.3) to the partition $N_2 \uplus N_1$ yields a subtree of $\mathfrak{T}'$ that converges with probability $< 1$ and whose infinite paths contain infinitely many nodes from $N_1$. Since once we reach a node in $N_2$, there are only nodes from $N_2$ onwards, no infinite path of this subtree contains a node from $N_2$. So all its subtrees that start in a node from $N_2$ are finite (by KÖNIG's Lemma, as the tree is finitely branching), and if we turn the topmost nodes from $N_2$ into leaves, we obtain a subtree with the same convergence probability that only contains inner nodes from $N_1$. Note that such a subtree does not use a rewrite step at the root and can therefore be seen as a $\xrightarrow{\neg\varepsilon}_\mathcal{P}$-RT. However, by the Parallel Execution Lemma (Lemma 5.2.2), which also holds for $\xrightarrow{\neg\varepsilon}_\mathcal{P}$-RTs that start with $f(q_1, \ldots, q_n)$ for an arbitrary $f$, and since every proper subterm of $t'$ is $\mathtt{AST}$, this subtree must converge with probability 1. This is our desired contradiction. Therefore, there exists a node $x \in N_2$ with $t_x \in \mathcal{T}_{goal}$ and $|\mathfrak{T}'[xE'^*]| < 1$.

**Step 2** First rewrite step is performed at the root

Let $t \in \mathcal{T}_{goal}$ and $\mathfrak{T}$ be a $\to_\mathcal{P}$-RT starting with $t$ that converges with probability $< 1$. This proof is once again an application of the Sequential Execution Lemma. Let $X$ be the set of nodes in $\mathfrak{T}$ where a root rewrite step is performed. We split the nodes of $\mathfrak{T}$ into $N_1$ containing all nodes reachable from the root before reaching a node in $X$ and $N_2$ containing all the nodes reachable from a node in $X$. Note that the nodes $x \in N_2$ that have no predecessor in $N_2$ are precisely the nodes where we perform the first rewrite step at the root and we still have $t_x \in \mathcal{T}_{goal}$. The argumentation is now the same as above. *QED*

If we restrict ourselves to innermost rewriting, then it suffices to consider start terms $t$ of the form $t = \ell\sigma$ for some rule $\ell \to \mu \in \mathcal{P}$ and some substitution $\sigma$ such that $\ell\sigma \in \mathtt{ANF}_\mathcal{P}$.

Theorem 5.3.3 – Witness for Innermost Non-$\mathtt{AST}$.
Let $\mathcal{P}$ be a finite PTRS. If $\neg\mathtt{AST}(\xrightarrow{\mathtt{i}}_\mathcal{P})$, then there exists an $\xrightarrow{\mathtt{i}}_\mathcal{P}$-RT $\mathfrak{T}$ that converges with probability $< 1$ and starts with $t$ such that $t = \ell\sigma$ for some rule $\ell \to \mu \in \mathcal{P}$ and some substitution $\sigma$, and $\ell\sigma \in \mathtt{ANF}_\mathcal{P}$.

*Proof.* We prove the contrapositive.
Let $\mathcal{T}_{goal}$ be the set of all terms $t$ such that $t = \ell\sigma \in \mathtt{ANF}_\mathcal{P}$ for some rule $\ell \to \mu \in \mathcal{P}$ and some substitution $\sigma$. Assume that every $\xrightarrow{\mathtt{i}}_\mathcal{P}$-RT $\mathfrak{T}$ that starts with $t \in \mathcal{T}_{goal}$ converges with probability 1. We now prove that then also every $\xrightarrow{\mathtt{i}}_\mathcal{P}$-RT $\mathfrak{T}$ that starts with $t$ for some arbitrary $t \in \mathcal{T}$ converges with probability 1, and thus $\mathtt{AST}(\xrightarrow{\mathtt{i}}_\mathcal{P})$ holds. This will be proven by structural induction over the term $t$.
If $t$ is in normal form, then obviously every $\xrightarrow{\mathtt{i}}_\mathcal{P}$-RT $\mathfrak{T}$ that starts with $t$ converges with probability 1. If $t = \ell\sigma$ for some rule $\ell \to \mu \in \mathcal{P}$ and some substitution $\sigma$ such that $\ell\sigma \in \mathtt{ANF}_\mathcal{P}$ (i.e., $t \in \mathcal{T}_{goal}$), then we know by our assumption that every $\xrightarrow{\mathtt{i}}_\mathcal{P}$-RT $\mathfrak{T}$ that starts with $t$ converges with probability 1.
Now we regard the induction step, and assume that $t = f(q_1, \ldots, q_n)$. Here, our induction hypothesis is that every $\xrightarrow{\mathtt{i}}_\mathcal{P}$-RT $\mathfrak{T}$ that starts with $q_i$ for some $1 \leq i \leq n$ converges with probability 1. Due to the innermost evaluation strategy, we can only rewrite at the root position if every proper subterm is in normal form w.r.t. $\mathcal{P}$. However, since we know that $q_i$ is $\mathtt{AST}(\xrightarrow{\mathtt{i}}_\mathcal{P})$ for all $1 \leq i \leq n$, we can use the innermost variant of the Parallel Execution Lemma (Lemma 5.2.2) for $\xrightarrow{\mathtt{i}\neg\varepsilon}_\mathcal{P}$-RTs (see the end of Section 5.2) to conclude $\mathtt{AST}(\xrightarrow{\mathtt{i}\neg\varepsilon}_\mathcal{P})$ for $f(q_1, \ldots, q_n)$.
Next, we additionally allow rewrite steps at the root position. Assume for a contradiction that there exists an $\xrightarrow{\mathtt{i}}_\mathcal{P}$-RT $\mathfrak{T}$ that starts with $f(q_1, \ldots, q_n)$ and converges with probability $< 1$. Let $X$ be the set of nodes in $\mathfrak{T}$ such that $t_x^\mathfrak{T} \in \mathtt{ANF}_\mathcal{P}$. We split the nodes of $\mathfrak{T}$ into $N_1$ containing all nodes reachable from the root before reaching a node in $X$, and $N_2$ containing all the nodes reachable from a node in $X$. Note that the nodes $x \in N_2$ that have no predecessor in $N_2$ are precisely the first nodes on each path with $t_x \in \mathtt{ANF}_\mathcal{P}$, i.e., $t_x$ is a normal form or $t_x \in \mathcal{T}_{goal}$.
We can use the Sequential Execution Lemma and the Parallel Execution Lemma as in the proof of Theorem 5.3.2 to show that this yields a contradiction, proving the induction step.

*Proof* … *QED*

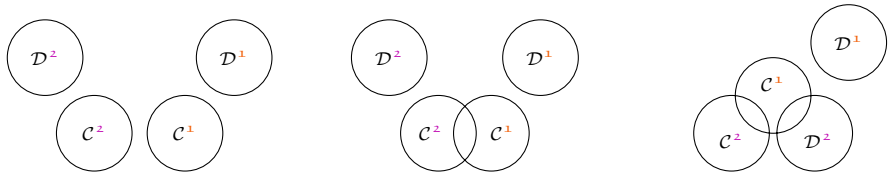
In contrast to $\mathtt{AST}(\xrightarrow{\mathtt{i}}_\mathcal{P})$, there is no analogous witness theorem for $\mathtt{PAST}(\xrightarrow{\mathtt{i}}_\mathcal{P})$ or $\mathtt{SAST}(\xrightarrow{\mathtt{i}}_\mathcal{P})$.

Counterexample 5.3.4. Consider the PTRS $\mathcal{P}$ with the two rules $\mathsf{a} \to \{{}^{1/2}\mathsf{0}, {}^{1/2}\mathsf{s}(\mathsf{a})\}$ and $\mathsf{f}(\mathsf{s}(x)) \to \{{}^{1}\mathsf{c}(\mathsf{f}(x), \mathsf{f}(x))\}$. Starting with $\mathsf{f}(\mathsf{a})$, innermost evaluation first rewrites $\mathsf{a}$ to $\mathsf{s}^n(\mathsf{0})$, which takes $n+1$ steps and happens with probability $2^{-(n+1)}$. Afterwards, $\mathsf{f}(\mathsf{s}^n(\mathsf{0}))$ needs $2^n - 1$ further steps. So the expected derivation length is $\sum_{n\in\mathbb{N}} 2^{-(n+1)} \cdot (n + 2^n) = \infty$, i.e., we have neither $\mathtt{PAST}(\xrightarrow{\mathtt{i}}_\mathcal{P})$ nor $\mathtt{SAST}(\xrightarrow{\mathtt{i}}_\mathcal{P})$. However, every term $\ell\sigma \in \mathtt{ANF}_\mathcal{P}$ for a rule $\ell \to \mu \in \mathcal{P}$ has finite expected derivation height.

SECTION 5.4

## Modularity

While the Sequential Execution Lemma and the Parallel Execution Lemma are powerful tools for the proofs of our techniques, normally one is more interested in a different notion of composability in term rewriting: *unions* of PTRSs. Since term rewriting can be seen as a functional programming language where no control structures exist, taking the union of two PTRSs $\mathcal{P}^1$ and $\mathcal{P}^2$ is the most natural combination and the most important in practice. A property is called *modular* for certain unions of PTRSs if a union has the property if and only if both of its parts have it. Modularity means that one can split a huge PTRS into smaller parts such that the property of interest is preserved. Then one can analyze these parts independently of each other. In this




(a) Disjoint union. (b) Shared constructor union. (c) Hierarchical union.

Figure 5.8: The three types of unions we investigate in this section [Ohl94].

section, we investigate modularity of AST, PAST, and SAST. We show that similar to the non-probabilistic setting [Toy87a, Gra95], most results regarding the modularity of termination only hold for innermost rewriting, and AST, PAST, and SAST are not modular w.r.t. full rewriting in general. However, our results on the relation between innermost and full probabilistic rewriting from Chapter 4 allow us to obtain modularity results for full probabilistic rewriting as well.

To distinguish the function symbols of different PTRSs $\mathcal{P}$, in the following we write $\mathcal{D}^{\mathcal{P}}$, $\mathcal{C}^{\mathcal{P}}$, and $\Sigma^{\mathcal{P}}$ for the defined symbols, constructor symbols, and all function symbols occurring in the rules of $\mathcal{P}$, respectively. Moreover, we distinguish between two PTRSs $\mathcal{P}^1$ and $\mathcal{P}^2$ that we want to combine into $\mathcal{P}^{1+2}$, i.e., $\mathcal{P}^{1+2} = \mathcal{P}^1 \uplus \mathcal{P}^2$. We may even simply write $\Sigma^1 = \Sigma^{\mathcal{P}^1}$, $\Sigma^2 = \Sigma^{\mathcal{P}^2}$, $\mathcal{D}^1 = \mathcal{D}^{\mathcal{P}^1}$, $\mathcal{C}^1 = \mathcal{C}^{\mathcal{P}^1}$, $\mathcal{D}^2 = \mathcal{D}^{\mathcal{P}^2}$, and $\mathcal{C}^2 = \mathcal{C}^{\mathcal{P}^2}$.

We present modularity results for the three most important forms of unions, namely *disjoint unions* of PTRSs (in Section 5.4.1), where both PTRSs do not share any function symbols; *shared constructor unions* of PTRSs (in Section 5.4.2), which may have common constructor symbols, but where no defined symbol of one PTRS occurs in the other one; and *hierarchical combinations* of PTRSs (in Section 5.4.3), which may have common constructor symbols, and additionally the first PTRS may contain defined symbols of the second one, but not vice versa. See Figure 5.8 for Venn diagrams of all three cases we investigate in this section. If one allows arbitrary unions where the defined symbols of each PTRS may occur arbitrarily in the rules of the other, then one cannot gain any modularity results. A counterexample for modularity in this case would be the two PTRSs $\mathcal{P}^1$ with the rule $\mathsf{a} \to \{^1\mathsf{b}\}$ and $\mathcal{P}^2$ with the rule $\mathsf{b} \to \{^1\mathsf{a}\}$. While both $\mathcal{P}^1$ and $\mathcal{P}^2$ are clearly AST, the union $\mathcal{P}^1 \uplus \mathcal{P}^2$ is not AST.

### 5.4.1 Disjoint Unions

We first consider unions of PTRSs that do not share any function symbols, i.e., we consider two PTRSs $\mathcal{P}^1$ and $\mathcal{P}^2$ such that $\Sigma^1 \cap \Sigma^2 = \varnothing$, see Figure 5.8a.

> Definition 5.4.1 – Disjoint Union.
> *We call $\mathcal{P}^{1+2} = \mathcal{P}^1 \uplus \mathcal{P}^2$ a disjoint union if $\Sigma^1 \cap \Sigma^2 = \varnothing$.*

In the non-probabilistic setting, [Gra95] showed that innermost termination is modular for disjoint unions.[1] This result can be lifted to $\mathtt{AST}(\xrightarrow{\mathsf{i}}_{\mathcal{P}})$ and $\mathtt{SAST}(\xrightarrow{\mathsf{i}}_{\mathcal{P}})$. We first investigate $\mathtt{AST}(\xrightarrow{\mathsf{i}}_{\mathcal{P}})$, where this result is an application of the witness theorem for $\mathtt{AST}(\xrightarrow{\mathsf{i}}_{\mathcal{P}})$ (Theorem 5.3.3). To be precise, due to the witness theorem it suffices to

[1] The proof in [Gra95] used an additional structural induction compared to our proof for $\mathtt{AST}(\xrightarrow{\mathsf{i}}_{\mathcal{P}})$. However, note that we apply the witness theorem for $\mathtt{AST}(\xrightarrow{\mathsf{i}}_{\mathcal{P}})$ that already used such a structural induction.

regard $\xrightarrow{\mathtt{i}}_{\mathcal{P}^{1+2}}$-RTs that start with a term $t$ such that $t = \ell\sigma \in \mathtt{ANF}_{\mathcal{P}^{1+2}}$ for some rule $\ell \to \mu \in \mathcal{P}^{1+2}$ and some substitution $\sigma$. Without loss of generality (otherwise simply swap $\mathcal{P}^1$ and $\mathcal{P}^2$), we use a rule from $\mathcal{P}^1$ at the root of such an $\xrightarrow{\mathtt{i}}_{\mathcal{P}^{1+2}}$-RT $\mathfrak{T}$. Then, there will never be a rule from $\mathcal{P}^2$ used in $\mathfrak{T}$. But this means that $\mathcal{P}^1$ already has to be not $\mathtt{AST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}})$.

We formalize this by the following theorem.

Theorem 5.4.2 – Modularity of $\mathtt{AST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}})$ for Disjoint Unions.

Let $\mathcal{P}^{1+2} = \mathcal{P}^1 \uplus \mathcal{P}^2$ be a disjoint union of finite PTRSs. Then we have

$$\mathtt{AST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}^{1+2}}) \iff \mathtt{AST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}^1}) \wedge \mathtt{AST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}^2}).$$

*Proof.* The direction "$\Longrightarrow$" is trivial and thus, we only prove "$\Longleftarrow$". Let $\mathcal{P}^{1+2} = \mathcal{P}^1 \uplus \mathcal{P}^2$ be a disjoint union such that both $\mathtt{AST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}^1})$ and $\mathtt{AST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}^2})$ hold.
Due to the witness theorem it suffices to regard $\xrightarrow{\mathtt{i}}_{\mathcal{P}^{1+2}}$-RTs that start with a term $t$ such that $t = \ell\sigma$ for some rule $\ell \to \mu \in \mathcal{P}^{1+2}$ and some substitution $\sigma$, and $\ell\sigma \in \mathtt{ANF}_{\mathcal{P}^{1+2}}$. Without loss of generality, let $\mathfrak{T}$ be such an $\xrightarrow{\mathtt{i}}_{\mathcal{P}^{1+2}}$-RT that converges with probability $< 1$ and uses a rule from $\mathcal{P}^1$ at the root. Let $\mathfrak{T}^1$ result from $\mathfrak{T}$ by starting with $t^1 \in \mathcal{T}(\Sigma^1, \mathcal{V})$, where $t^1$ results from $t$ by replacing all its maximal (i.e., topmost) subterms with root symbols from $\Sigma^2$ by fresh variables (using the same variable for the same subterm). Since both PTRSs have disjoint signatures and all proper subterms of $t$ are in normal form, we can still apply the same rules as in $\mathfrak{T}$, such that $\mathfrak{T}^1$ is an $\xrightarrow{\mathtt{i}}_{\mathcal{P}^1}$-RT with $|\mathfrak{T}^1| = |\mathfrak{T}|$. However, since $|\mathfrak{T}| < 1$, we would have $|\mathfrak{T}^1| < 1$, which is a contradiction to $\mathcal{P}^1$ being $\mathtt{AST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}})$. *Proof* *QED*

In contrast to $\mathtt{AST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}})$, $\mathtt{PAST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}})$ cannot be modular due to the potential extension of the signature (recall that by Theorem 3.4.20, $\mathtt{PAST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}})$ is not closed under signature extensions).

Counterexample 5.4.3. Consider the PTRS $\mathcal{P}_{\mathsf{unary}}$ from Counterexample 3.4.18, and a PTRS $\mathcal{P}_7$ containing a binary function symbol $\mathsf{c}$ such that both $\mathtt{PAST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}_7})$ and $\Sigma^{\mathcal{P}_{\mathsf{unary}}} \cap \Sigma^{\mathcal{P}_7} = \varnothing$ hold. For example, $\mathcal{P}_7$ could consist of the only rule $\mathsf{c}(\mathsf{d},\mathsf{d}) \to \{^1\mathsf{c}(\mathsf{e},\mathsf{e})\}$. As explained in Counterexample 3.4.19, due to the signature extension by a binary function symbol, we do not have $\mathtt{PAST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}_{\mathsf{unary}} \cup \mathcal{P}_7})$, while both $\mathtt{PAST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}_{\mathsf{unary}}})$ (over the signature $\Sigma^{\mathcal{P}_{\mathsf{unary}}}$) and $\mathtt{PAST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}_7})$ hold.

For $\mathtt{SAST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}})$ the problem is more difficult than for $\mathtt{AST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}})$. Note that there exists no witness theorem for $\mathtt{SAST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}})$ that we could utilize in the proof of modularity, see Counterexample 5.3.4. Therefore, if we want to prove that $\mathtt{SAST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}})$ is modular for disjoint unions, we have to come up with a completely new approach, without considering only terms that contain a single redex at the root in the beginning. The goal is to show that the expected derivation height of any term $t$ (with potentially nested redexes) is finite w.r.t. $\mathcal{P}^{1+2}$ assuming that this holds w.r.t. $\mathcal{P}^1$ and w.r.t. $\mathcal{P}^2$. However, after rewriting $t$'s proper subterms to normal forms, we may end up with infinitely many terms. All their expected derivation heights have to be considered in order to compute the expected derivation height of $t$.

Example 5.4.4. Consider the PTRSs $\mathcal{P}_8^{1}$ and $\mathcal{P}_8^{2}$ with

$$\begin{aligned} \mathcal{P}_8^{1} : \mathsf{f}(\mathsf{s}(x), y) &\to \{{}^{1}\mathsf{f}(x, y)\} \\ \mathsf{f}(x, \mathsf{s}(y)) &\to \{{}^{1}\mathsf{f}(x, y)\} \\ \mathsf{a} &\to \{{}^{1/2}\mathsf{0}, {}^{1/2}\mathsf{s}(\mathsf{a})\} \end{aligned} \qquad \mathcal{P}_8^{2} : \mathsf{g}(x) \to \{{}^{1}x\}$$

Clearly, we have both $\mathtt{SAST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}_8^{1}})$ and $\mathtt{SAST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}_8^{2}})$. Now consider the term $t = \mathsf{f}(\mathsf{g}(\mathsf{a}), \mathsf{g}(\mathsf{a}))$. Due to the innermost strategy, we have to rewrite its proper subterms first. A first idea would be to construct bounds on the expected derivation heights of the proper subterms, and then use them to obtain a bound on the expected derivation height of the whole term $t$. However, reducing $t$'s proper subterms can create infinitely many different terms, i.e., all terms of the form $\mathsf{f}(\mathsf{s}^n(\mathsf{0}), \mathsf{s}^m(\mathsf{0}))$ for any $n, m \in \mathbb{N}$ can be reached with a certain probability. Since there is no finite supremum on the derivation height of $\mathsf{f}(\mathsf{s}^n(\mathsf{0}), \mathsf{s}^m(\mathsf{0}))$ for all $n, m \in \mathbb{N}$, we have to take the individual probabilities for reaching the terms $\mathsf{f}(\mathsf{s}^n(\mathsf{0}), \mathsf{s}^m(\mathsf{0}))$ into account in order to prove that the expected derivation height of $t$ is indeed finite.

Instead of considering infinitely many different terms, we now present how to over-approximate this set of all reachable terms by a finite set. Since the root symbol $\mathsf{f}$ of $t = \mathsf{f}(\mathsf{g}(\mathsf{a}), \mathsf{g}(\mathsf{a}))$ is from $\Sigma^{\mathcal{P}_8^{1}}$, we have to over-approximate the normal forms reachable from the maximal subterms starting with a symbol from $\Sigma^{\mathcal{P}_8^{2}}$. For this over-approximation we have to consider the normal forms reachable from the argument $\mathsf{a}$ of $\mathsf{g} \in \Sigma^{\mathcal{P}_8^{2}}$ (because function symbols like $\mathsf{g}$ may have "collapsing rules" which return their arguments) or by fresh variables (which represent possible normal forms that start with symbols from $\Sigma^{\mathcal{P}_8^{2}}$). Thus, instead of considering the rewrite steps at symbols from $\Sigma^{\mathcal{P}_8^{1}}$ in $t$, we can consider all rewrite steps for the terms from the set $\{\mathsf{f}(\mathsf{a},\mathsf{a}), \mathsf{f}(x,\mathsf{a}), \mathsf{f}(\mathsf{a},y), \mathsf{f}(x,y), \mathsf{f}(y,x), \mathsf{f}(x,x), \mathsf{f}(y,y)\}$. This set is called the *disjoint union abstraction* of $t$ for $\Sigma^{\mathcal{P}_8^{1}}$. Note that all terms in this disjoint union abstraction are indeed from $\mathcal{T}\big(\Sigma^{\mathcal{P}_8^{1}}, \mathcal{V}\big)$. To also capture the possibility that the two occurrences of $\mathsf{g}(\mathsf{a})$ in $t$ might reach the same normal form that starts with a symbol from $\Sigma^{\mathcal{P}_8^{2}}$, the disjoint union abstraction of $t$ also contains $\mathsf{f}(x, x)$.

Similarly, instead of considering the rewrite steps at symbols from $\Sigma^{\mathcal{P}_8^{2}}$ in $t$, we consider the two arguments of $\mathsf{f}$ (i.e., the two occurrences of $\mathsf{g}(\mathsf{a})$) with roots from $\Sigma^{\mathcal{P}_8^{2}}$ where each occurrence of the subterm $\mathsf{a} \in \Sigma^{\mathcal{P}_8^{1}}$ is replaced by a fresh variable (to represent possible normal forms that start with symbols from $\Sigma^{\mathcal{P}_8^{1}}$). Thus, the disjoint union abstraction of $t$ for $\Sigma^{\mathcal{P}_8^{2}}$ contains $\mathsf{g}(x)$.

For the modularity proof of $\mathtt{SAST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}})$ we replace the start term $t$ by all terms in the disjoint union abstractions for both $\Sigma^{1}$ and $\Sigma^{2}$. This is a finite set of $K \in \mathbb{N}$ terms. Since every term in this abstraction is either from $\mathcal{T}(\Sigma^{1}, \mathcal{V})$ or from $\mathcal{T}(\Sigma^{2}, \mathcal{V})$, they all have a finite expected derivation height. Hence, if $C_{\max} \in \mathbb{R}$ is the maximal expected derivation height of all these terms, then $K \cdot C_{\max}$ is a (finite) bound on the expected derivation height of $t$.

The following definition introduces the *disjoint union abstraction* formally. Here, $\mathrm{A}_{1}(t)$ is the set where all topmost subterms of $t$ with root from $\Sigma^{2}$ are replaced by fresh variables or by the abstractions of their subterms. The set $\mathrm{Cap}_{1}(t)$ results from $\mathrm{A}_{1}(t)$ by enforcing equality of any possible combination of the variables in the terms of $\mathrm{A}_{1}(t)$. In addition, we also want to label the function symbols in the abstraction, to indicate from which position the function symbol originated in the original term. For a term $t$,

let $\mathrm{pos}_{\mathcal{V}}(t)$ be the set of all its variable positions and let $\mathrm{pos}_{\Sigma}(t)$ be the set of all those positions of $t$ where $t$ has function symbols instead of variables.

Definition 5.4.5 – Disjoint Union Abstraction.

*Let $\mathcal{P}^1, \mathcal{P}^2$ be PTRSs with $\Sigma^1 \cap \Sigma^2 = \varnothing$. For any $\mathfrak{d} \in \{1, 2\}$, $t \in \mathcal{T}(\Sigma^1 \cup \Sigma^2, \mathcal{V})$, and position $\pi \in \mathrm{pos}(t)$, $\mathrm{A}^{\pi}_{\mathfrak{d}}(t|_\pi)$ (for the subterm $t|_\pi$ of $t$ at position $\pi$) and $\mathrm{Cap}_{\mathfrak{d}}(t)$ are sets of (labeled) terms from $\mathcal{T}(L(\Sigma^{\mathfrak{d}}), \mathcal{V})$, which are defined as follows. Here, we use the signature $L(\Sigma^{\mathfrak{d}}) = \Sigma^{\mathfrak{d}} \times \mathrm{pos}(t)$, where we write $f^\pi$ for $(f, \pi) \in \Sigma^{\mathfrak{d}} \times \mathrm{pos}(t)$.*

$$\begin{aligned}
\mathrm{A}^{\pi}_{\mathfrak{d}}(y) &= \{x\}, \text{ if } y \in \mathcal{V}, \text{ where } x \text{ is always a new fresh variable,}\\
\mathrm{A}^{\pi}_{\mathfrak{d}}(f(t_1, \ldots, t_k)) &= \{f^\pi(q_1, \ldots, q_k) \mid q_1 \in \mathrm{A}^{\pi.1}_{\mathfrak{d}}(t_1), \ldots, q_k \in \mathrm{A}^{\pi.k}_{\mathfrak{d}}(t_k)\}, \text{ if } f \in \Sigma^{\mathfrak{d}},\\
\mathrm{A}^{\pi}_{\mathfrak{d}}(f(t_1, \ldots, t_k)) &= \{x\} \cup \mathrm{A}^{\pi.1}_{\mathfrak{d}}(t_1) \cup \ldots \cup \mathrm{A}^{\pi.k}_{\mathfrak{d}}(t_k), \text{ otherwise}\\
&\quad \text{where } x \text{ is always a new fresh variable.}
\end{aligned}$$

*So $\mathrm{A}^{\pi}_{\mathfrak{d}}(t|_\pi)$ is always a set of linear terms.*
*For any function $\varphi : X \to X$ with $X \subseteq \mathcal{V}$, let $\sigma_\varphi$ be the substitution that replaces every variable $x \in X$ by $\varphi(x) \in X$ and leaves all other variables unchanged, i.e., $\sigma_\varphi(x) = \varphi(x)$ if $x \in X$ and $\sigma_\varphi(x) = x$ otherwise. Then we define*

$$\mathrm{Cap}_{\mathfrak{d}}(t) = \{\sigma_\varphi(q) \mid q \in \mathrm{A}^{\varepsilon}_{\mathfrak{d}}(t), \varphi : \mathcal{V}(q) \to \mathcal{V}(q)\}$$

*and $\mathrm{Abs}(t) = \mathrm{Cap}_1(t) \cup \mathrm{Cap}_2(t)$, called the disjoint union abstraction of $t$.*

Example 5.4.6. Consider the PTRS $\mathcal{P}^1_9$ with the rules $\mathsf{a} \to \{^1\mathsf{f}(\mathsf{b})\}$, $\mathsf{b} \to \{^1\mathsf{0}\}$, and $\mathsf{h}(y, y) \to \{^1y\}$, the PTRS $\mathcal{P}^2_9$ with the rule $\mathsf{g}(y) \to \{^1\mathsf{c}\}$, and the term $\mathsf{h}(\mathsf{g}(\mathsf{a}), \mathsf{g}(x))$. We obtain[2]

$$\begin{aligned}
\mathrm{A}^{\varepsilon}_1(\mathsf{h}(\mathsf{g}(\mathsf{a}), \mathsf{g}(x))) &= \{\mathsf{h}^\varepsilon(\mathsf{a}^{1.1}, z), \mathsf{h}^\varepsilon(y, z)\}\\
\mathrm{Cap}_1(\mathsf{h}(\mathsf{g}(\mathsf{a}), \mathsf{g}(x))) &= \{\mathsf{h}^\varepsilon(\mathsf{a}^{1.1}, z), \mathsf{h}^\varepsilon(z, y), \mathsf{h}^\varepsilon(y, z), \mathsf{h}^\varepsilon(z, z), \mathsf{h}^\varepsilon(y, y)\}\\
\mathrm{A}^{\varepsilon}_2(\mathsf{h}(\mathsf{g}(\mathsf{a}), \mathsf{g}(x))) &= \{x', \mathsf{g}^1(y), \mathsf{g}^2(z)\}\\
\mathrm{Cap}_2(\mathsf{h}(\mathsf{g}(\mathsf{a}), \mathsf{g}(x))) &= \{x', \mathsf{g}^1(y), \mathsf{g}^2(z)\}.
\end{aligned}$$

The following lemma states the two most important properties regarding the disjoint union abstraction.

Lemma 5.4.7 – Properties of $\mathrm{Cap}_{\mathfrak{d}}$.

*Let $\mathcal{P}^1, \mathcal{P}^2$ be finite PTRSs with $\Sigma^1 \cap \Sigma^2 = \varnothing$, $t \in \mathcal{T}(\Sigma^1 \cup \Sigma^2, \mathcal{V})$, and $\mathfrak{d} \in \{1, 2\}$. Then*

1. *$\mathrm{Cap}_{\mathfrak{d}}(t)$ is finite, and*
2. *after erasing the labels, we have $\mathrm{Cap}_{\mathfrak{d}}(t) \subseteq \mathcal{T}(\Sigma^{\mathfrak{d}}, \mathcal{V})$.*

*Proof.* Simple proof by induction on the term structure. *Proof QED*

With the disjoint union abstraction, we can prove the modularity of $\mathtt{SAST}(\xrightarrow{\mathrm{i}}_{\mathcal{P}})$ for disjoint unions.

Theorem 5.4.8 – Modularity of $\mathtt{SAST}(\xrightarrow{\mathrm{i}}_{\mathcal{P}})$ for Disjoint Unions.

Let $\mathcal{P}^{1+2} = \mathcal{P}^1 \uplus \mathcal{P}^2$ be a disjoint union of finite PTRSs. Then we have

$$\mathtt{SAST}(\xrightarrow{\mathrm{i}}_{\mathcal{P}^{1+2}}) \iff \mathtt{SAST}(\xrightarrow{\mathrm{i}}_{\mathcal{P}^1}) \wedge \mathtt{SAST}(\xrightarrow{\mathrm{i}}_{\mathcal{P}^2}).$$

[2] To be precise, $\mathrm{A}^{\varepsilon}_1(\mathsf{h}(\mathsf{g}(\mathsf{a}), \mathsf{g}(x)))$ contains two additional terms $\mathsf{h}^\varepsilon(\mathsf{a}^{1.1}, z')$ and $\mathsf{h}^\varepsilon(y, z')$ since $\mathrm{A}^2_1(\mathsf{g}(x)) = \{z, z'\}$. However, to ease readability, we disregarded them here.

Before we can prove modularity for disjoint unions w.r.t. $\mathtt{SAST}(\xrightarrow{\mathrm{i}}_{\mathcal{P}})$, we need some more definitions regarding the abstraction. We first define which terms $q \in \mathrm{Abs}(t)$ "cover" which of the nodes from $N^{\mathfrak{T}}$ for an arbitrary $\xrightarrow{\mathrm{i}}_{\mathcal{P}_1+\mathcal{P}_2}$-RT $\mathfrak{T}$ starting with $t$. The idea is that every step in an $\xrightarrow{\mathrm{i}}_{\mathcal{P}_1+\mathcal{P}_2}$-RT starting with $t$ (corresponding to an inner node of $\mathfrak{T}$) can also be performed when starting with a suitable $q \in \mathrm{Abs}(t)$. For this, we first define the notion of an *origin graph*. For every rewrite sequence starting with $t$, the graph indicates which subterm "originates" from which subterm of $t$. Moreover, every function symbol in the origin graph is labeled by its position relative to the symbol of $t$ it originates from, which is obtained by concatenating the positions of the corresponding symbols on the right-hand sides of the rules that created it.

Definition 5.4.9 – Origin Graph.

*Let $\mathcal{P}$ be a PTRS and let $\mathfrak{T}$ be an $\xrightarrow{\mathrm{i}}_{\mathcal{P}}$-RT. The origin graph for $\mathfrak{T}$ is a labeled graph with the nodes $(x,\tau)$ for all $x \in N^{\mathfrak{T}}$ and all $\tau \in \mathrm{pos}_\Sigma(t_x)$ where the edges and labels are defined as follows. For the root $\mathfrak{r}$ of $\mathfrak{T}$ we label the node $(\mathfrak{r},\tau)$ by $\varepsilon$. For $x \in N^{\mathfrak{T}}$, let the rewrite step $t_x \xrightarrow{\mathrm{i}}_{\mathcal{P}} \{^{p_1}t_{y_1},\dots,^{p_k}t_{y_k}\}$ be performed using the rule $\ell \to \{^{p_1}r_1,\dots,^{p_k}r_k\}$, the position $\pi$, and the substitution $\sigma$, i.e., $t_x|_\pi = \ell\sigma$ and $t_{y_j} = t_x[r_j\sigma]_\pi$ for all $1 \le j \le k$. Additionally, let $\tau \in \mathrm{pos}_\Sigma(t_x)$.*

(a) *If $\tau < \pi$ or $\tau \perp \pi$ (i.e., $\tau$ is above or parallel to $\pi$), then there is an edge from $(x,\tau)$ to $(y_j,\tau)$. If $(x,\tau)$ was labeled by $\chi$, then $(y_j,\tau)$ is labeled by $\chi$ as well.*

(b) *For $\tau = \pi$, there is an edge from $(x,\pi)$ to $(y_j,\pi.\alpha)$ for all $\alpha \in \mathrm{pos}_\Sigma(r_j)$. If $(x,\pi)$ was labeled by $\chi$, then $(y_j,\pi.\alpha)$ is labeled by $\chi.\alpha$.*

(c) *For every variable position $\alpha_\ell \in \mathrm{pos}_\mathcal{V}(\ell)$, for all positions $\alpha_{r_j} \in \mathrm{pos}_\mathcal{V}(r_j)$ with $r_j|_{\alpha_{r_j}} = \ell|_{\alpha_\ell}$, and for all $\beta \in \mathbb{N}^*$ with $\alpha_\ell.\beta \in \mathrm{pos}_\Sigma(\ell\sigma)$, there is an edge from $(x,\pi.\alpha_\ell.\beta)$ to $(y_j,\pi.\alpha_{r_j}.\beta)$. If $(x,\pi.\alpha_\ell.\beta)$ was labeled by $\chi$, then $(y_j,\pi.\alpha_{r_j}.\beta)$ is labeled by $\chi$ as well.*

(d) *For all other positions $\tau \in \mathrm{pos}_\Sigma(t_x)$, i.e., the positions that are inside the redex $\ell\sigma$ but neither at the root of $\ell$ nor inside the substitution $\sigma$, there is no outgoing edge from the node $(x,\tau)$.*

Example 5.4.10. Reconsider the PTRSs $\mathcal{P}_9^1$ with the rules $\mathsf{a} \to \{^1\mathsf{f}(\mathsf{b})\}$, $\mathsf{b} \to \{^1\mathsf{0}\}$, and $\mathsf{h}(y,y) \to \{^1y\}$, the PTRS $\mathcal{P}_9^2$ with the rule $\mathsf{g}(y) \to \{^1\mathsf{c}\}$, and the term $\mathsf{h}(\mathsf{g}(\mathsf{a}),\mathsf{g}(x))$ from Example 5.4.6. Furthermore, consider the following RT

$$\mathsf{h}(\mathsf{g}(\underline{\mathsf{a}}),\mathsf{g}(x)) \xrightarrow{(1)} \mathsf{h}(\mathsf{g}(\mathsf{f}(\underline{\mathsf{b}})),\mathsf{g}(x)) \xrightarrow{(2)} \mathsf{h}(\underline{\mathsf{g}}(\mathsf{f}(\mathsf{0})),\mathsf{g}(x)) \xrightarrow{(3)} \mathsf{h}(\mathsf{c},\underline{\mathsf{g}}(x)) \xrightarrow{(4)} \underline{\mathsf{h}}(\mathsf{c},\mathsf{c}) \xrightarrow{(5)} \mathsf{c},$$

where we omitted the (trivial) probabilities and numbered each rewrite step.

This RT yields the origin graph depicted in Figure 5.9.

Note that the root of every subterm that is not in normal form is reachable from exactly one node $(\mathfrak{r},\tau)$. Because of innermost rewriting, only normal forms can be duplicated by duplicating rules (leading to multiple outgoing edges) or merged by non-left-linear rules (leading to multiple incoming edges), and thus, only nodes $(x,\tau)$ where $t_x|_\tau$ is in normal form may have multiple incoming edges. Labels are only needed for nodes $(x,\tau)$ where $t_x|_\tau$ is not in normal form, which have at most one predecessor, so that their labels are well defined.

Our goal is to split up the $\xrightarrow{\mathrm{i}}_{\mathcal{P}_1+\mathcal{P}_2}$-RT that started with $t$ into $\xrightarrow{\mathrm{i}}_{\mathcal{P}_1}$-RTs and $\xrightarrow{\mathrm{i}}_{\mathcal{P}_2}$-RTs

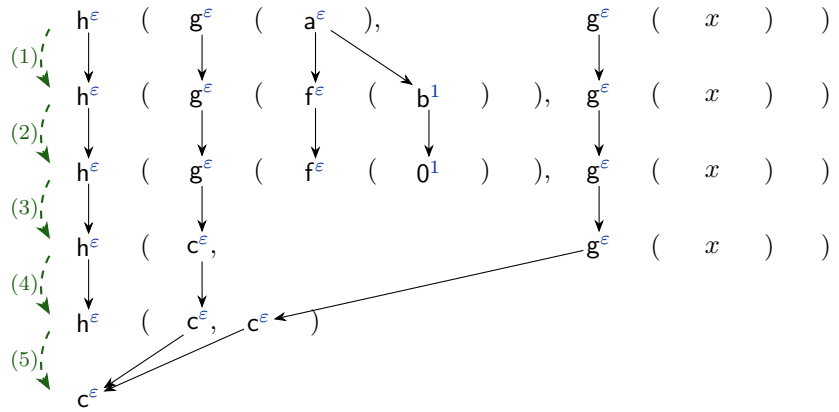



Figure 5.9: Origin graph.

that start with $q$ for $q \in \mathrm{Abs}(t)$. The reason is that these terms $q$ only contain symbols from either $\Sigma^{\mathsf{1}}$ or $\Sigma^{\mathsf{2}}$ and hence, there is a bound on the expected derivation length of all these RTs.

The labels in the origin graph and the labels in the terms $q \in \mathrm{Abs}(t)$ can now be used to construct the RTs that start with $q \in \mathrm{Abs}(t)$ from the original RT that starts with $t$. Then every rewrite step in the original RT corresponds to at least one step in one of these new RTs. Let us illustrate this with our running example.

Example 5.4.11. From the $\xrightarrow{\mathsf{i}}_{\mathcal{P}_9}$-RT in Example 5.4.10 we obtain the following $\xrightarrow{\mathsf{i}}_{\mathcal{P}_9^{\mathsf{1}}}$-RTs and $\xrightarrow{\mathsf{i}}_{\mathcal{P}_9^{\mathsf{2}}}$-RTs[3]

$$\begin{array}{lcll} \mathsf{h}^{\varepsilon}(\underline{\mathsf{a}^{1.1}}, z) & \xrightarrow{(1')} & \mathsf{h}(\mathsf{f}(\underline{\mathsf{b}}), z) & \xrightarrow{(2')} \quad \mathsf{h}(\mathsf{f}(0), z) \\ \underline{\mathsf{h}^{\varepsilon}}(z, z) & \xrightarrow{(5')} & z & \\ \underline{\mathsf{g}^{1}}(y) & \xrightarrow{(3')} & \mathsf{c} & \\ \underline{\mathsf{g}^{2}}(z) & \xrightarrow{(4')} & \mathsf{c} & \end{array}$$

where the labels are ignored for the rewrite steps, but they are used to determine which start term from $\mathrm{Abs}(t)$ to use for which step. The rewrite step $(i)$ in the original RT corresponds to the rewrite step $(i')$ in our new RTs. In the original RT, one first performs rewrite steps for the symbols $\mathsf{a}$ and $\mathsf{b}$ from $\Sigma^{\mathcal{P}_9^{\mathsf{1}}}$, then one rewrites the symbol $\mathsf{g}$ from $\Sigma^{\mathcal{P}_9^{\mathsf{2}}}$, and finally one rewrites the top symbol $\mathsf{h}$ from $\Sigma^{\mathcal{P}_9^{\mathsf{1}}}$. In contrast, these rewrite steps are now separated such that in each of the above RTs, one either only rewrites symbols from $\Sigma^{\mathcal{P}_9^{\mathsf{1}}}$ or only symbols from $\Sigma^{\mathcal{P}_9^{\mathsf{2}}}$.

Let us explain how to detect that the rewrite step (2) that is performed at position $\pi = 1.1.1$ in $\mathsf{h}(\mathsf{g}(\mathsf{f}(\mathsf{b})), \mathsf{g}(x))$ in the original RT should be applied at position 1.1 in $\mathsf{h}(\mathsf{f}(\mathsf{b}), z)$ for rewrite step $(2')$. We consider the (unique) predecessor $(\mathfrak{r}, 1.1)$ (i.e., $\mathsf{a}^{\varepsilon}$) of $(x_2, 1.1.1)$ (i.e., $\mathsf{b}^{1}$), where $\mathfrak{r}$ is the root of the original RT and $x_2$ denotes the second node of the original RT containing the term $\mathsf{h}(\mathsf{g}(\mathsf{f}(\mathsf{b})), \mathsf{g}(x))$. This indicates that the rewrite step (2) in the original RT corresponds to a rewrite step in a new RT that

[3] In addition to the RT starting with $\mathsf{h}^{\varepsilon}(z, z)$ we also obtain the corresponding RT starting with $\mathsf{h}^{\varepsilon}(y, y)$, but omitted it here for readability.

starts with $q$ containing $\mathrm{root}(t_\mathfrak{r}|_{1.1})^{1.1} = \mathrm{root}(\mathsf{h}(\mathsf{g}(\mathsf{a}), \mathsf{g}(x))|_{1.1})^{1.1} = \mathsf{a}^{1.1}$. Hence, we have to consider the new RT starting with $\mathsf{h}^\varepsilon(\underline{\mathsf{a}^{1.1}}, z)$.

To find the actual rewrite step that corresponds to (2) in this new RT, we determine the position of the symbol $\mathsf{a}^{1.1}$ within $\mathsf{h}^\varepsilon(\mathsf{a}^{1.1}, z)$, which is $\gamma = 1$, and the label of $(x_2, 1.1.1)$ (i.e., $\mathsf{b}^1$), which is $\chi = 1$. This indicates that in the new RT we need to rewrite at position $\gamma.\chi = 1.1$.

Now, we define this covering formally in order to determine which term $q \in \mathrm{Abs}(t)$ to use for the rewrite step. More precisely, for any $q \in \mathrm{Abs}(t)$ we define the *abstraction cover* $AC_q \subseteq N^\mathfrak{T}$ which contains all inner nodes of the original RT whose rewrite step can be simulated by a rewrite step in the new RT for $q$. Moreover, for every node $x$, we define the term $\psi_x(q)$ which we would obtain instead of $t_x$ if we had started the RT with $q$ instead of $t$. Thus, if there is an edge from the node $x$ to the node $y$ in the original RT $\mathfrak{T}$, then $\psi_x(q)$ rewrites to $\psi_y(q)$ or they are equal. Therefore, the term $\psi_\mathfrak{r}(q)$ rewrites (in zero or more steps) to $\psi_x(q)$ for every node of $\mathfrak{T}$.

Definition 5.4.12 – Abstraction Cover.
*Let $\mathcal{P}^{1+2} = \mathcal{P}^1 \uplus \mathcal{P}^2$ be a PTRS with $\Sigma^1 \cap \Sigma^2 = \varnothing$ and let $\mathfrak{T}$ be an $\xrightarrow{\mathrm{i}}_{\mathcal{P}^{1+2}}$-RT that starts with $t$. We recursively define the abstraction cover $AC_q \subseteq N^\mathfrak{T} \setminus \mathrm{Leaf}^\mathfrak{T}$ for each $q \in \mathrm{Abs}(t)$, where we additionally define a term $\psi_x(q)$ for each node $x \in N^\mathfrak{T}$ that corresponds to $t_x$ in the RT starting with $q$.*
*For the root node $\mathfrak{r}$ of $\mathfrak{T}$, we initially set $\psi_\mathfrak{r}(q) = q$.*
*Now consider the rewrite step at an arbitrary node $x \in N^\mathfrak{T}$, which uses the rule $\ell \to \{\ldots, {}^{p_j} r_j, \ldots\}$ at position $\pi$ in $t_x$ with substitution $\delta$. Hence, we have $t_{y_j} = t_x[r_j\delta]_\pi$. Let the origin graph of $\mathfrak{T}$ contain a path from $(\mathfrak{r}, \tau)$ to $(x, \pi)$ for some position $\tau$ and let $Q \subseteq \mathrm{Abs}(t)$ be the set of all terms from $\mathrm{Abs}(t)$ that contain a function symbol labeled with $\tau$. Moreover, let $(x, \pi)$ be labeled with position $\chi$ in the origin graph of $\mathfrak{T}$, and for every $q \in Q$, let $\gamma$ be the position of the function symbol labeled with $\tau$ in $q$. Then we add $x$ to $AC_q$ for all those $q \in Q$ where there exists a substitution $\delta'$ with $\psi_x(q)|_{\gamma.\chi} = \ell\delta'$ such that all proper subterms of $\ell\delta'$ are normal forms. For these $q$, we set $\psi_{y_j}(q) = \psi_x(q)[r_j\delta']_{\gamma.\chi}$. Note that now indeed, $\psi_x(q)$ rewrites to $\psi_{y_j}(q)$ with an innermost rewrite step. For all other $q \in \mathrm{Abs}(t)$ we set $\psi_{y_j}(q) = \psi_x(q)$.*

Example 5.4.13. For instance, in Example 5.4.11 for $q = \mathsf{h}(\mathsf{a}, z)$ we have $\psi_{x_2}(q) = \mathsf{h}(\mathsf{f}(\mathsf{b}), z)$ and $\psi_{x_i}(q) = \mathsf{h}(\mathsf{f}(\mathsf{0}), z)$ for all $i \in \{3, 4, 5, 6\}$ (we remove the labels in $\psi_x(q)$).

Lemma 5.4.14 – $AC_q$ Covers All Inner Nodes.
*Let $\mathcal{P}^{1+2} = \mathcal{P}^1 \uplus \mathcal{P}^2$ be a finite PTRS with $\Sigma^1 \cap \Sigma^2 = \varnothing$ and let $\mathfrak{T}$ be an $\xrightarrow{\mathrm{i}}_{\mathcal{P}^{1+2}}$-RT that starts with $t$. Then $N^\mathfrak{T} \setminus \mathrm{Leaf}^\mathfrak{T} = \bigcup_{q \in \mathrm{Abs}(t)} AC_q$.*

*Proof.* In order to show that $\bigcup_{q \in \mathrm{Abs}(t)} AC_q$ is indeed a cover of $N^\mathfrak{T} \setminus \mathrm{Leaf}^\mathfrak{T}$, we have to show that for every node $x \in N^\mathfrak{T} \setminus \mathrm{Leaf}^\mathfrak{T}$ there exists a $q \in \mathrm{Abs}(t)$ with $x \in AC_q$. In other words, we have to show that there exists at least one term $q \in Q$ such that there is a substitution $\delta'$ with $\psi_x(q)|_{\gamma.\chi} = \ell\delta'$ and all proper subterms of $\ell\delta'$ are normal forms, where $\gamma$ is again the position of the function symbol labeled with $\tau$ in $q$. Without loss of generality, let $\mathrm{root}(\ell) \in \Sigma^1$. Let $C_1$ be the maximal context containing no symbols from $\Sigma^2$ such that there exists another context $C$ and terms $s_1, \ldots, s_m$ with $\mathrm{root}(s_i) \in \Sigma^2$ for all $1 \leq i \leq m$ such that $t_x = C[C_1[s_1, \ldots, s_m]]$ and

Proof

the position $\pi$ is within the context $C_1$.
Let $\Phi$ be the set of positions in $\mathrm{pos}_\Sigma(t_x)$ that are within the context $C_1$. There exists a term $q \in \mathrm{Abs}(t)$ that contains all function symbols that are labeled with positions $\varphi'$ where there is a path in the origin graph from $(\mathfrak{r}, \varphi')$ to $(x, \varphi)$ for some $\varphi \in \Phi$. This also holds if there is a normal form at position $\varphi$ which may be reached from several positions $\varphi'$. Moreover, let $\{\rho_1, \ldots, \rho_m\}$ be the set of root positions of the $s_1, \ldots, s_m$ within $\mathrm{pos}(t_x)$, i.e., the positions of the holes in $C_1$ within $t_x$. We choose $q$ in such a way that the fresh variables for symbols from $\Sigma^{(2)}$ at positions $\kappa'$ and $\kappa''$ of $t$ are the same if and only if the corresponding subterms of $t$ are rewritten to the same normal form on the path from the root to $x$. This is possible, since in an $\xrightarrow{\mathrm{i}}_{\mathcal{P}^{(1+2)}}$-RT every subterm with a root symbol from $\Sigma^{(2)}$ that is duplicated, erased, or compared with another subterm by a rewrite step with $\mathcal{P}^{(1)}$ is already a normal form (as it is a proper subterm of an innermost redex), and normal forms are never rewritten again. Thus, for this specific $q$ we have $\psi_x(q)|_{\gamma.\chi} = \ell\delta'$ where $\delta'$ is like $\delta$, but we replace every occurrence of the terms $s_i$ with the corresponding fresh variables. Moreover, all proper subterms of $\ell\delta'$ are normal forms. *QED*

Example 5.4.15. As a final example, consider the rewrite step (5), i.e., $\mathsf{h}(\mathsf{c}, \mathsf{c}) \to \mathsf{c}$. Following the notation from the proof above, we have $C_1 = \mathsf{h}(\Box, \Box)$, $s_1 = \mathsf{c}$, and $s_2 = \mathsf{c}$. Furthermore, we have $\rho_1 = \chi = 1$, $\rho_2 = \chi' = 2$, and the position $\varphi$ of the $\mathsf{h}$ in the context $C_1$ is $\varphi = \varepsilon$ with $\varphi^{-1} = \varepsilon$. Hence, we choose $\mathsf{h}^{\varepsilon}(z, z) \in \mathrm{Abs}(t)$, as this term contains the function symbol $\mathsf{h}$ labeled with $\varphi^{-1} = \varepsilon$, and uses the same variable $z$ for both subterms, as they are equal ($s_1 = s_2 = \mathsf{c}$).

*Proof of Theorem 5.4.8.* The direction "$\Longrightarrow$" is trivial and thus, we only prove "$\Longleftarrow$". *Proof*
Let $\mathcal{P}^{(1+2)} = \mathcal{P}^{(1)} \uplus \mathcal{P}^{(2)}$ be a disjoint union such that both $\mathtt{SAST}(\xrightarrow{\mathrm{i}}_{\mathcal{P}^{(1)}})$ and $\mathtt{SAST}(\xrightarrow{\mathrm{i}}_{\mathcal{P}^{(2)}})$ hold. Let $\mathfrak{T} = (N, E)$ be an arbitrary $\xrightarrow{\mathrm{i}}_{\mathcal{P}^{(1+2)}}$-RT that starts with $t$. We prove that $\mathrm{edl}(\mathfrak{T})$ is bounded by some constant which does not depend on $\mathfrak{T}$ but just on $t$. This then proves $\mathtt{SAST}(\xrightarrow{\mathrm{i}}_{\mathcal{P}})$.
In Definition 5.4.12 we defined sets $AC_q \subseteq N^{\mathfrak{T}}$ for each $q \in \mathrm{Abs}(t)$ such that $x \in AC_q$ if the rewrite step at node $x \in N^{\mathfrak{T}}$ is performed at some position $\pi$ in $t_x$, $q$ contains a function symbol labeled with $\tau$ where there is a path from $(\mathfrak{r}, \tau)$ to $(x, \pi)$ in the origin graph of $\mathfrak{T}$, $(x, \pi)$ is labeled with some position $\chi$ in the origin graph, and we can perform the same rewrite step on $\psi_x(q)$ at position $\gamma.\chi$, where $\gamma$ is the position of the function symbol labeled with $\tau$ in $q$. In Lemma 5.4.14 we showed that $\bigcup_{q \in \mathrm{Abs}(t)} AC_q = N^{\mathfrak{T}} \setminus \mathrm{Leaf}^{\mathfrak{T}}$.
With the definition of $AC_q$ we can now prove the upper bound on the expected derivation height of $t$. Let $H \in \mathbb{N}$ and let $\mathfrak{T}_H$ be the tree consisting of all nodes of $\mathfrak{T}$ with depth at most $H$. For every $q \in \mathrm{Abs}(t)$, let $AC_q^H = AC_q \cap (N^{\mathfrak{T}_H} \setminus \mathrm{Leaf}^{\mathfrak{T}_H})$ be the set of inner nodes of $\mathfrak{T}_H$ that are in $AC_q$. By Lemma 5.4.14, we have $\bigcup_{q \in \mathrm{Abs}(t)} AC_q^H = N^{\mathfrak{T}_H} \setminus \mathrm{Leaf}^{\mathfrak{T}_H}$. As in the proof of the Parallel Execution Lemma for $\mathtt{SAST}$ (Lemma 5.2.4), for each $H \in \mathbb{N}$ and each $q \in \mathrm{Abs}(t)$ we split the tree $\mathfrak{T}_H$ into (finitely many) sets $\mathbb{T}_H^q$ containing (finitely many) pairs of $\xrightarrow{\mathrm{i}}_{\mathcal{P}^{(1)}}$-RTs and $\xrightarrow{\mathrm{i}}_{\mathcal{P}^{(2)}}$-RTs $\mathfrak{t}$ with certain probabilities $p_{\mathfrak{t}}$. Each of these RTs $\mathfrak{t}$ starts with $q$, has height at most $H$, and

(Cov-1) $\sum_{(p_{\mathfrak{t}}, \mathfrak{t}) \in \mathbb{T}_H^q} p_{\mathfrak{t}} = 1$.

(Cov-2) For all $x \in AC_q^H$ we have $p_x^{\mathfrak{T}_H} = \sum_{(p_{\mathfrak{t}}, \mathfrak{t}) \in \mathbb{T}_H^q} p_{\mathfrak{t}} \cdot p_x^{\mathfrak{t}}$.

(Cov-3) For all $(p_{\mathfrak{t}}, \mathfrak{t}) \in \mathbb{T}_H^q$, the RT $\mathfrak{t}$ starts with $q$ and is a valid $\xrightarrow{\mathrm{i}}_{\mathcal{P}^{(0)}}$-RT for some

$\mathfrak{d} \in \{1, 2\}$.

The construction is analogous to the ones in Lemmas 5.2.2 and 5.2.4. Now we keep all nodes that are in $AC_q^H$ and split a tree into multiple ones at every other inner node of $\mathfrak{T}_H$, i.e., if the rewrite step required a different starting term in the beginning. Since $|\mathrm{Abs}(t)| = K$ is finite and the expected derivation lengths of all $\xrightarrow{\mathfrak{i}}_{\mathcal{P}^\mathfrak{d}}$-RTs with $\mathfrak{d} \in \{1, 2\}$ that start with $q$ for a term $q \in \mathrm{Abs}(t)$ are bounded by some constant $C_q < \infty$, there is a $C_{\max} < \infty$ such that for all RTs $\mathfrak{t}$ with $(p_\mathfrak{t}, \mathfrak{t}) \in \mathbb{T}_H^q$ we have $\mathrm{edl}(\mathfrak{t}) \leq C_{\max}$ by (Cov-3). Hence, we obtain for each $H \in \mathbb{N}$:

$$
\begin{aligned}
\mathrm{edl}(\mathfrak{T}_H) &= \sum_{x \in N^{\mathfrak{T}_H} \setminus \mathrm{Leaf}^{\mathfrak{T}_H}} p_x^{\mathfrak{T}_H} \\
&\quad \downarrow \ (\text{since } \bigcup_{q \in \mathrm{Abs}(t)} AC_q^H = N^{\mathfrak{T}_H} \setminus \mathrm{Leaf}^{\mathfrak{T}_H}) \\
&\leq \sum_{q \in \mathrm{Abs}(t)} \sum_{x \in AC_q^H} p_x^{\mathfrak{T}_H} \\
&\quad \downarrow \ (\text{by (Cov-2)}) \\
&= \sum_{q \in \mathrm{Abs}(t)} \sum_{x \in AC_q^H} \sum_{(p_\mathfrak{t}, \mathfrak{t}) \in \mathbb{T}_H^q} p_\mathfrak{t} \cdot p_x^\mathfrak{t} \\
&= \sum_{q \in \mathrm{Abs}(t)} \sum_{(p_\mathfrak{t}, \mathfrak{t}) \in \mathbb{T}_H^q} p_\mathfrak{t} \cdot \sum_{x \in AC_q^H} p_x^\mathfrak{t} \\
&= \sum_{q \in \mathrm{Abs}(t)} \sum_{(p_\mathfrak{t}, \mathfrak{t}) \in \mathbb{T}_H^q} p_\mathfrak{t} \cdot \sum_{x \in N^\mathfrak{t} \setminus \mathrm{Leaf}^\mathfrak{t}} p_x^\mathfrak{t} \\
&\quad \downarrow \ (\text{by assumption}) \\
&\leq \sum_{q \in \mathrm{Abs}(t)} \sum_{(p_\mathfrak{t}, \mathfrak{t}) \in \mathbb{T}_H^q} p_\mathfrak{t} \cdot C_{\max} \\
&= \sum_{q \in \mathrm{Abs}(t)} C_{\max} \cdot \sum_{(p_\mathfrak{t}, \mathfrak{t}) \in \mathbb{T}_H^q} p_\mathfrak{t} \\
&\quad \downarrow \ (\text{by (Cov-1)}) \\
&= \sum_{q \in \mathrm{Abs}(t)} C_{\max} \cdot 1 = \sum_{q \in \mathrm{Abs}(t)} C_{\max} = K \cdot C_{\max}.
\end{aligned}
$$

Thus, we have $\mathrm{edl}(\mathfrak{T}) = \lim_{H \to \infty} \mathrm{edl}(\mathfrak{T}_H) \leq K \cdot C_{\max}$. *QED*

Example 5.4.16. Let us illustrate the notation in the previous proof by applying it to the PTRS from Example 5.4.4. If we reconsider the start term $\mathsf{f}(\mathsf{g}(\mathsf{a}), \mathsf{g}(\mathsf{a}))$, then $\mathrm{Cap}_1(\mathsf{f}(\mathsf{g}(\mathsf{a}), \mathsf{g}(\mathsf{a}))) = \{\mathsf{f}(\mathsf{a}, \mathsf{a}), \mathsf{f}(x, \mathsf{a}), \mathsf{f}(\mathsf{a}, y), \mathsf{f}(x, y), \mathsf{f}(y, x), \mathsf{f}(x, x), \mathsf{f}(y, y)\}$ and $\mathrm{Cap}_2(\mathsf{f}(\mathsf{g}(\mathsf{a}), \mathsf{g}(\mathsf{a}))) = \{x', \mathsf{g}(x), \mathsf{g}(y)\}$ (where we omit the labels), analogously to Example 5.4.6. Moreover, we obtain $C_{\max} = C_{\mathsf{f}(\mathsf{a},\mathsf{a})} = 2 + 2 + 2 \cdot \sum_{n=1}^{\infty} (1/2)^{n+1} \cdot n = 2 + 2 + 2 \cdot 1 = 6$, where $C_{\mathsf{f}(\mathsf{a},\mathsf{a})}$ is the bound on the expected derivation height of the term $\mathsf{f}(\mathsf{a}, \mathsf{a})$. (The reason is that reaching a normal form from the subterm $\mathsf{a}$ needs 2 steps in expectation, and then for each generated $\mathsf{s}$ we have one additional step, where each $\mathsf{a}$ generates $n$ $\mathsf{s}$-symbols with probability $(1/2)^{n+1}$.) Hence, the overall bound on the expected derivation height for $\mathsf{f}(\mathsf{g}(\mathsf{a}), \mathsf{g}(\mathsf{a}))$ is $K \cdot C_{\max} = 10 \cdot 6 = 60$. In fact, the actual expected derivation height for $\mathsf{f}(\mathsf{g}(\mathsf{a}), \mathsf{g}(\mathsf{a}))$ is $6 + 2 = 8$ (2 steps for the two

g-symbols, and in expectation 6 steps for the term $\mathsf{f}(\mathsf{a},\mathsf{a})$), but for the proof any finite bound suffices.

For full rewriting, it is well known that termination is not modular for TRSs.

Counterexample 5.4.17. Consider the TRSs $\mathcal{R}^1 = \{\mathsf{f}(\mathsf{a},\mathsf{b},x) \to \mathsf{f}(x,x,x)\}$ and $\mathcal{R}^2 = \{\mathsf{g}(x,y) \to x,\ \mathsf{g}(x,y) \to y\}$ from [Toy87a] (the TRS from Counterexample 4.1.1 is a variant of their union with shared constructors). Both $\mathcal{R}^1$ and $\mathcal{R}^2$ are terminating, but their disjoint union is not, due to the infinite rewrite sequence $\mathsf{f}(\mathsf{a},\mathsf{b},\mathsf{g}(\mathsf{a},\mathsf{b})) \to \mathsf{f}(\mathsf{g}(\mathsf{a},\mathsf{b}),\mathsf{g}(\mathsf{a},\mathsf{b}),\mathsf{g}(\mathsf{a},\mathsf{b})) \to \mathsf{f}(\mathsf{a},\mathsf{g}(\mathsf{a},\mathsf{b}),\mathsf{g}(\mathsf{a},\mathsf{b})) \to \mathsf{f}(\mathsf{a},\mathsf{b},\mathsf{g}(\mathsf{a},\mathsf{b})) \to \cdots$. Read as PTRSs with trivial probabilities, $\mathcal{R}^1$ and $\mathcal{R}^2$ are $\mathtt{SAST}$, but their union is not $\mathtt{AST}$.

However, one can reuse our results from Chapter 4 to obtain the following corollary.

Corollary 5.4.18 – Modularity of $\mathtt{AST}(\xrightarrow{\mathsf{f}}_{\mathcal{P}})$ and $\mathtt{SAST}(\xrightarrow{\mathsf{f}}_{\mathcal{P}})$ for Disjoint Unions.
*Let $\mathcal{P}^{1+2} = \mathcal{P}^1 \uplus \mathcal{P}^2$ be a disjoint union of finite PTRSs, and let $\mathcal{P}^{1+2}$ be $\mathtt{NO}$ and linear. Then we have*

$$\mathtt{AST}(\to_{\mathcal{P}^{1+2}}) \iff \mathtt{AST}(\to_{\mathcal{P}^1}) \wedge \mathtt{AST}(\to_{\mathcal{P}^2}),\ \text{and}$$
$$\mathtt{SAST}(\to_{\mathcal{P}^{1+2}}) \iff \mathtt{SAST}(\to_{\mathcal{P}^1}) \wedge \mathtt{SAST}(\to_{\mathcal{P}^2}).$$

*Proof.* Follows from Theorems 4.2.2, 5.4.2 and 5.4.8. *Proof QED*

### 5.4.2 Shared Constructor Unions

Now we consider unions of PTRSs that may share constructor symbols, i.e., we consider two PTRSs $\mathcal{P}^1$ and $\mathcal{P}^2$ such that $\mathcal{D}^1 \cap \Sigma^2 = \varnothing$ and $\mathcal{D}^2 \cap \Sigma^1 = \varnothing$, called *shared constructor unions*, see Figure 5.8b.

Definition 5.4.19 – Shared Constructor Union.
*We call $\mathcal{P}^{1+2} = \mathcal{P}^1 \uplus \mathcal{P}^2$ a shared constructor union if both $\mathcal{D}^1 \cap \Sigma^2 = \varnothing$ and $\mathcal{D}^2 \cap \Sigma^1 = \varnothing$.*

Again, we study innermost rewriting first. In the non-probabilistic setting, innermost termination is also modular for shared constructor unions [Gra95]. $\mathtt{PAST}(\xrightarrow{\mathsf{i}}_{\mathcal{P}})$ was already not modular w.r.t. disjoint unions, so this also holds for shared constructor unions. Additionally, $\mathtt{SAST}(\xrightarrow{\mathsf{i}}_{\mathcal{P}})$ also turns out to be not modular anymore for shared constructor unions.

Counterexample 5.4.20. Consider the PTRS $\mathcal{P}_{10} = \mathcal{P}_{10}^1 \cup \mathcal{P}_{10}^2$ with the rules

$$\mathcal{P}_{10}^1 : \mathsf{f}(\mathsf{c}(x,y)) \to \{^1\mathsf{c}(\mathsf{f}(x),\mathsf{f}(y))\} \qquad \mathcal{P}_{10}^2 : \mathsf{g}(x) \to \{^{1/2}\mathsf{g}(\mathsf{d}(x)),\ ^{1/2}x\}$$
$$\mathsf{f}(\mathsf{0}) \to \{^1\mathsf{0}\} \qquad \mathsf{d}(x) \to \{^1\mathsf{c}(x,x)\}$$

While no defined symbol of $\mathcal{P}_{10}^1$ occurs in $\mathcal{P}_{10}^2$ and vice versa, they share the constructor symbol $\mathsf{c}$. We do not have $\mathtt{PAST}(\xrightarrow{\mathsf{i}}_{\mathcal{P}_{10}})$ (and thus, not $\mathtt{SAST}(\xrightarrow{\mathsf{i}}_{\mathcal{P}_{10}})$), as the $\xrightarrow{\mathsf{i}}_{\mathcal{P}_{10}}$-RT depicted in Figure 5.10 has an infinite expected derivation length.

For any $n \in \mathbb{N}$, let $\mathsf{c}^n(\mathsf{0},\mathsf{0})$ denote the full binary tree of height $n$ with $\mathsf{c}$ in inner nodes and $\mathsf{0}$ in the leaves, i.e., $\mathsf{c}^0(\mathsf{0},\mathsf{0}) = \mathsf{0}$ and $\mathsf{c}^{n+1}(\mathsf{0},\mathsf{0}) = \mathsf{c}(\mathsf{c}^n(\mathsf{0},\mathsf{0}),\mathsf{c}^n(\mathsf{0},\mathsf{0}))$. Each term $\mathsf{f}(\mathsf{c}^n(\mathsf{0},\mathsf{0}))$ in the right branches can start a reduction of at least length $2^n$. Hence, the term $\mathsf{f}(\mathsf{g}(\mathsf{0}))$ has an expected derivation height of at least $\sum_{n=0}^{\infty} \frac{1}{2^{n+1}} \cdot 2^n = \sum_{n=0}^{\infty} \frac{1}{2}$,

which diverges to infinity.

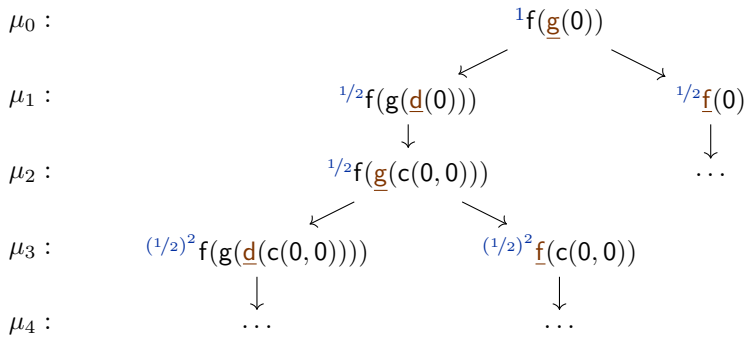



Figure 5.10: $\xrightarrow{\mathtt{i}}_{\mathcal{P}_{10}}$-RT.

On the other hand, we have $\mathtt{SAST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}_{10}^1})$, as $\mathcal{P}_{10}^1$ is a PTRS with only trivial probabilities that corresponds to a terminating TRS. Moreover, $\mathtt{SAST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}_{10}^2})$ holds as well, as the $\mathsf{d}$-rule can increase the number of $\mathsf{c}$-symbols in a term exponentially, but those $\mathsf{c}$-symbols will never be used. Thus, $\mathtt{SAST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}})$ is not modular for shared constructor unions.

In contrast to the proof of Theorem 5.4.8, we cannot use the disjoint union abstraction anymore to obtain a bound on the expected derivation height, since symbols from $\Sigma^2$ can now "generate" constructor symbols of $\Sigma^1$. However, for $\mathtt{AST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}})$ we can reuse the idea of the previous proof for Theorem 5.4.2 to obtain a similar result for shared constructor unions.

Theorem 5.4.21 – Modularity of $\mathtt{AST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}})$ for Shared Constructor Unions.
Let $\mathcal{P}^{1+2} = \mathcal{P}^1 \uplus \mathcal{P}^2$ be a shared constructor union of finite PTRSs. Then we have

$$\mathtt{AST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}^{1+2}}) \iff \mathtt{AST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}^1}) \wedge \mathtt{AST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}^2}).$$

*Proof.* The direction "$\Longrightarrow$" is trivial and thus, we only prove "$\Longleftarrow$". Let $\mathcal{P}^{1+2} = \mathcal{P}^1 \uplus \mathcal{P}^2$ be a shared constructor union such that both $\mathtt{AST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}^1})$ and $\mathtt{AST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}^2})$ hold. Due to the witness theorem it suffices to regard $\xrightarrow{\mathtt{i}}_{\mathcal{P}^{1+2}}$-RTs that start with a term $t = \ell\sigma \in \mathtt{ANF}_{\mathcal{P}^{1+2}}$ for some rule $\ell \to \mu \in \mathcal{P}^{1+2}$ and some substitution $\sigma$. Without loss of generality, we use a rule from $\mathcal{P}^1$ at the root of such an RT $\mathfrak{T}$. Let $\mathfrak{T}^1$ result from $\mathfrak{T}$ by starting with $t^1$, where we do not replace all maximal subterms with roots from $\Sigma^2$ by fresh variables, but instead just maximal subterms with root symbols that are not from $\Sigma^1$ (e.g., from $\mathcal{D}^2$), as the shared constructor symbols may be used by rules of $\mathcal{P}^1$. So we have $t^1 \in \mathcal{T}(\Sigma^1, \mathcal{V})$. Since this construction only replaces normal forms by variables, and the replaced subterms have a root symbol that is not from $\Sigma^1$, making them unusable for rules from $\mathcal{P}^1$, we can still apply the same rules as in $\mathfrak{T}$, such that $\mathfrak{T}^1$ is an $\xrightarrow{\mathtt{i}}_{\mathcal{P}^1}$-RT with $|\mathfrak{T}^1| = |\mathfrak{T}|$. However, since $|\mathfrak{T}| < 1$, we would have $|\mathfrak{T}^1| < 1$, which is a contradiction to $\mathcal{P}^1$ being $\mathtt{AST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}})$. *Proof* *QED*

Again, we can reuse our results from Chapter 4 on the relation between full and innermost rewriting to obtain the following corollary for full rewriting.

Corollary 5.4.22 – Modularity of $\mathtt{AST}(\xrightarrow{\mathtt{f}}_{\mathcal{P}})$ for Shared Constructor Unions.

*Let $\mathcal{P}^{1+2} = \mathcal{P}^1 \uplus \mathcal{P}^2$ be a shared constructor union of finite PTRSs, and let $\mathcal{P}^{1+2}$ be* NO *and linear. Then we have*

$$\mathtt{AST}(\to_{\mathcal{P}^{1+2}}) \iff \mathtt{AST}(\to_{\mathcal{P}^1}) \wedge \mathtt{AST}(\to_{\mathcal{P}^2}).$$

*Proof.* Follows from Theorems 4.2.2 and 5.4.21. *Proof QED*

### 5.4.3 Hierarchical Unions

Finally, we investigate unions of PTRSs that may share constructor symbols and where the defined symbols of the second one can be used as constructor symbols by the first one. To be precise, we consider two PTRSs $\mathcal{P}^1$ and $\mathcal{P}^2$ such that $\mathcal{D}^1 \cap \Sigma^2 = \varnothing$, but we may have $\mathcal{C}^1 \cap \mathcal{D}^2 \neq \varnothing$. Such a union is called *hierarchical*, see Figure 5.8c.

Definition 5.4.23 – Hierarchical Union.

*We call $\mathcal{P}^{1+2} = \mathcal{P}^1 \uplus \mathcal{P}^2$ a hierarchical union if $\mathcal{D}^1 \cap \Sigma^2 = \varnothing$.*

Since $\mathtt{PAST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}})$ is already not modular w.r.t. disjoint unions, and $\mathtt{SAST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}})$ is not modular w.r.t. shared constructor unions, both of them are not modular w.r.t. hierarchical unions either. Additionally, even $\mathtt{AST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}})$ is not closed under hierarchical unions anymore.

Counterexample 5.4.24. Consider the hierarchical union $\mathcal{P}^{1+2} = \mathcal{P}^1 \uplus \mathcal{P}^2$ where $\mathcal{P}^1$ contains the rule $\mathsf{f}(\mathsf{b}) \to \{^1\mathsf{f}(\mathsf{a})\}$ and $\mathcal{P}^2$ contains the rule $\mathsf{a} \to \{^1\mathsf{b}\}$. It is a hierarchical union because the only defined symbol $\mathsf{a}$ of $\mathcal{P}^2$ occurs as a constructor in $\mathcal{P}^1$, but the only defined symbol $\mathsf{f}$ of $\mathcal{P}^1$ does not occur at all in $\mathcal{P}^2$. However, while $\mathtt{AST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}^1})$ and $\mathtt{AST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}^2})$ hold, we have $\neg\mathtt{AST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}^{1+2}})$ for the union, as one can see by the infinite $\xrightarrow{\mathtt{i}}_{\mathcal{P}^{1+2}}$-rewrite sequence $\{^1\underline{\mathsf{f}}(\mathsf{b})\} \xrightarrow{\mathtt{i}}_{\mathcal{P}^{1+2}} \{^1\mathsf{f}(\underline{\mathsf{a}})\} \xrightarrow{\mathtt{i}}_{\mathcal{P}^{1+2}} \{^1\underline{\mathsf{f}}(\mathsf{b})\} \xrightarrow{\mathtt{i}}_{\mathcal{P}^{1+2}} \cdots$.

However, there exist conditions on the PTRSs $\mathcal{P}^1$ and $\mathcal{P}^2$ such that $\mathtt{AST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}})$ is preserved again. The problem of Counterexample 5.4.24 is that the symbol $\mathsf{a} \in \mathcal{D}^2$ occurs below the symbol $\mathsf{f} \in \mathcal{D}^1$. If we prevent that defined symbols of $\mathcal{P}^2$ can occur below defined symbols of $\mathcal{P}^1$, then $\mathtt{AST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}})$ is preserved even under hierarchical unions.

In order to prevent the creation of defined symbols of $\mathcal{P}^2$ below defined symbols of $\mathcal{P}^1$ it does not suffice to require that such behavior is prohibited in all the terms in the right-hand side of all rewrite rules.

Counterexample 5.4.25. Let us adjust the hierarchical union $\mathcal{P}^{1+2} = \mathcal{P}^1 \uplus \mathcal{P}^2$ from Counterexample 5.4.24 slightly. Let $\mathcal{P}^1$ now contain the two rules $\mathsf{f}(\mathsf{b}) \to \{^1\mathsf{f}(\mathsf{c})\}$ and $\mathsf{c} \to \{^1\mathsf{a}\}$, and $\mathcal{P}^2$ still contains the rule $\mathsf{a} \to \{^1\mathsf{b}\}$. Now in every term in the right-hand side of all rewrite rules, namely $\mathsf{f}(\mathsf{c})$ and $\mathsf{a}$, we have no defined symbol of $\mathcal{P}^2$ below a defined symbol of $\mathcal{P}^1$. However, we have the same problem as in Counterexample 5.4.24 and both $\mathcal{P}^1$ and $\mathcal{P}^2$ are $\mathtt{AST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}})$, but $\mathcal{P}^{1+2}$ is not $\mathtt{AST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}})$.

The problem in Counterexample 5.4.25 is that there are two nested defined symbols of $\mathcal{P}^1$ in the right-hand side of the rule $\mathsf{f}(\mathsf{b}) \to \{^1\mathsf{f}(\mathsf{c})\}$. This has to be additionally prohibited in order to guarantee our desired condition. All in all, this yields the following result, which lifts the corresponding result of the non-probabilistic setting,

where a similar condition was already considered [Gra95, Rao94, Rao95].[4]

**Theorem 5.4.26** – Preservation of $\mathtt{AST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}})$ under Hierarchical Unions.

Let $\mathcal{P}^{1+2} = \mathcal{P}^1 \uplus \mathcal{P}^2$ be a hierarchical union of finite PTRSs. Additionally, let there be no $\pi \in \mathrm{pos}_{\mathcal{D}^1}(r)$ and $\tau \in \mathrm{pos}_{\mathcal{D}^1 \cup \mathcal{D}^2}(r)$ for any $r \in \mathrm{Supp}(\mu)$ with $\ell \to \mu \in \mathcal{P}^{1+2}$ such that $\pi < \tau$. Then we have

$$\mathtt{AST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}^{1+2}}) \impliedby \mathtt{AST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}^1}) \wedge \mathtt{AST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}^2}).$$

*Proof.* We assume that we have both $\mathtt{AST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}^1})$ and $\mathtt{AST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}^2})$ (by Theorem 5.5.2, this also holds over the signature of $\mathcal{P}^{1+2}$). By Theorem 5.3.3 it suffices to show that every $\xrightarrow{\mathtt{i}}_{\mathcal{P}^{1+2}}$-RT converges with probability 1 if it starts with a term $t = \ell\sigma \in \mathtt{ANF}_{\mathcal{P}^{1+2}}$ for some rule $\ell \to \mu \in \mathcal{P}^{1+2}$ and some substitution $\sigma$.

We call a term $s$ *separated* if all proper subterms of all subterms of $s$ with a root symbol from $\mathcal{D}^1$ are normal forms w.r.t. $\mathcal{P}^{1+2}$. Every $t \in \mathtt{ANF}_{\mathcal{P}^{1+2}}$ is separated. Now let $s$ be separated, and consider an innermost rewrite step at a position $\pi$ of $s$ with a rule $\ell' \to \{p_1 r_1, \dots, p_k r_k\}$ and a substitution $\delta$. Then $\pi$ is not below a symbol from $\mathcal{D}^1$, as $s|_\pi$ is not a normal form. By the condition of the theorem, below every symbol from $\mathcal{D}^1$ in $r_j$ there are only constructors and variables (rules of $\mathcal{P}^2$ do not contain symbols from $\mathcal{D}^1$ at all), and $\delta(x)$ is a normal form for every $x \in \mathcal{V}(\ell')$, since $\ell'\delta$ is an innermost redex. Hence, every $s[r_j\delta]_\pi$ is separated as well, i.e., all terms in an $\xrightarrow{\mathtt{i}}_{\mathcal{P}^{1+2}}$-RT that starts with $t$ are separated. We call a subterm with a root symbol from $\mathcal{D}^1$ that is not a normal form *active*. So every rewrite step with $\mathcal{P}^1$ rewrites an active subterm at its root. Every rewrite step with $\mathcal{P}^2$ takes place at a position that is parallel to all active subterms (no active subterm is below it, as all proper subterms of an innermost redex are normal forms, and it is not below an active subterm, as it is not below a symbol from $\mathcal{D}^1$), and it does not create symbols from $\mathcal{D}^1$. Thus, rewrite steps with $\mathcal{P}^2$ neither modify nor create active subterms.

Now assume for a contradiction that there is an $\xrightarrow{\mathtt{i}}_{\mathcal{P}^{1+2}}$-RT $\mathfrak{T}$ that starts with $t$ such that $|\mathfrak{T}| < 1$. We partition the inner nodes of $\mathfrak{T}$ into the set $N_1$ of nodes where a rule from $\mathcal{P}^1$ is used and the set $N_2$ of nodes where a rule from $\mathcal{P}^2$ is used. Every subtree of $\mathfrak{T}$ that only contains inner nodes from $N_2$ is an $\xrightarrow{\mathtt{i}}_{\mathcal{P}^2}$-RT (as normal forms w.r.t. $\mathcal{P}^{1+2}$ are also normal forms w.r.t. $\mathcal{P}^2$), and thus, it converges with probability 1. By the Sequential Execution Lemma (Lemma 5.1.3) (applied with the roles of $N_1$ and $N_2$ exchanged), there exists a subtree $\mathfrak{T}' = (N', E')$ of $\mathfrak{T}$ with $|\mathfrak{T}'| < 1$ such that every infinite path of $\mathfrak{T}'$ contains infinitely many nodes from $N_1$.

It remains to remove the nodes from $N_2$. For every node $x$ of $\mathfrak{T}'$, let $\mathfrak{T}'_x = \mathfrak{T}'[xE'^*]$ be the subtree starting at $x$, where all probabilities are divided by $p_x$. If $x \in N_2$, then $|\mathfrak{T}'_x|$ is a convex combination of the values $|\mathfrak{T}'_y|$ for the successors $y$ of $x$, so we can choose a successor $y$ with $|\mathfrak{T}'_y| \leq |\mathfrak{T}'_x|$. Starting in a node from $N_2$ and always moving to the chosen successor, we reach a node from $N_1$ or a leaf after finitely many steps, as $\mathfrak{T}'$ has no infinite path that only contains nodes from $N_2$ from some point on. We construct a tree $\mathfrak{T}^1$ from $\mathfrak{T}'$ by skipping all nodes from $N_2$ in this way, i.e., the root of $\mathfrak{T}^1$ and every successor of a node from $N_1$ is replaced by the first node that is reached via chosen successors, and we keep the probabilities of the rewrite steps at the nodes from $N_1$. The root of $\mathfrak{T}^1$ is labeled with $t$, and we perform the rewrite steps of the nodes from $N_1$ with the same rules, substitutions, and positions. Since rewrite steps

*Proof*

[4] In [Rao95], a related condition that is formulated w.r.t. a dependency relation on the function symbols is used to define *proper extensions*.

with $\mathcal{P}^2$ neither modify nor create active subterms, every active subterm of a term in $\mathfrak{T}'$ also occurs at the same position in the label of the corresponding node of $\mathfrak{T}^1$. Hence, $\mathfrak{T}^1$ is an $\xrightarrow{\mathtt{i}}_{\mathcal{P}^1}$-RT (as normal forms w.r.t. $\mathcal{P}^{1+2}$ are also normal forms w.r.t. $\mathcal{P}^1$). By induction on $n$, for every node $x$ of $\mathfrak{T}^1$, the probability of reaching a leaf within $n$ steps from $x$ in $\mathfrak{T}^1$ is at most $|\mathfrak{T}'_x|$, since $|\mathfrak{T}'_x|$ does not increase along chosen successors, and $|\mathfrak{T}'_x|$ is the weighted sum of the values $|\mathfrak{T}'_y|$ of the successors $y$ of $x$ for $x \in N_1$. Thus, we obtain $|\mathfrak{T}^1| \leq |\mathfrak{T}'_\mathfrak{r}| = |\mathfrak{T}'| < 1$, which contradicts $\mathtt{AST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}^1})$. *QED*

Note that Theorem 5.4.26 only states one direction, so it is not a full modularity result. However, this is the direction that is important for proving $\mathtt{AST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}})$, since it allows us to analyze $\mathcal{P}^1$ and $\mathcal{P}^2$ separately. The converse direction does not hold in general.

Counterexample 5.4.27. Consider the hierarchical union $\mathcal{P}^{1+2} = \mathcal{P}^1 \uplus \mathcal{P}^2$ with $\mathcal{P}^1 = \{\mathsf{f}(\mathsf{a}, x) \to \{^1\mathsf{f}(x, x)\}\}$ and $\mathcal{P}^2 = \{\mathsf{a} \to \{^1\mathsf{b}\}\}$, which satisfies the condition of Theorem 5.4.26. We have $\mathtt{AST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}^{1+2}})$, since every instance of $\mathsf{f}(\mathsf{a}, x)$ contains the redex $\mathsf{a}$, so the $\mathsf{f}$-rule is never used in an innermost rewrite step. However, we do not have $\mathtt{AST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}^1})$, since $\mathsf{a}$ is a normal form w.r.t. $\mathcal{P}^1$, which yields the infinite rewrite sequence $\{^1\underline{\mathsf{f}}(\mathsf{a}, \mathsf{a})\} \xrightarrow{\mathtt{i}}_{\mathcal{P}^1} \{^1\underline{\mathsf{f}}(\mathsf{a}, \mathsf{a})\} \xrightarrow{\mathtt{i}}_{\mathcal{P}^1} \cdots$.

As for disjoint and shared constructor unions, we can use the results of Chapter 4 for full rewriting.

Corollary 5.4.28 – Preservation of $\mathtt{AST}(\xrightarrow{\mathtt{f}}_{\mathcal{P}})$ under Hierarchical Unions.

*Let $\mathcal{P}^{1+2} = \mathcal{P}^1 \uplus \mathcal{P}^2$ be a hierarchical union of finite PTRSs that satisfies the condition of Theorem 5.4.26, and let $\mathcal{P}^{1+2}$ be* NO *and linear. If $\mathtt{AST}(\to_{\mathcal{P}^1})$ and $\mathtt{AST}(\to_{\mathcal{P}^2})$, then $\mathtt{AST}(\to_{\mathcal{P}^{1+2}})$.*

*Proof.* Follows from Theorems 4.2.2 and 5.4.26, since $\mathtt{AST}(\xrightarrow{\mathtt{f}}_{\mathcal{P}})$ implies $\mathtt{AST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}})$. *Proof QED*

Instead of taking the union of disjoint sets of rules, one might also be interested in unions that share rewrite rules. For example, if we have rules that compute the addition of two natural numbers, and then additionally, two separate algorithms that use the rules for addition. In this case, we would like to keep the rules in each part of the union. Nevertheless, we stop our investigation at this point. Instead, in Chapter 7, we develop a divide-and-conquer framework generalizing all the results from this chapter. This framework uses two separate types of rules: *dependency pairs* and *rewrite rules*. The idea is that we only have to prove termination of the *dependency pairs*, where we can utilize the rewrite rules for addition as well, but we do not have to prove termination of the rewrite rules. For example, we may have already proven termination of the addition rewrite rules, but we need to consider them when analyzing a different part of the program. The *dependency graph processor* splits the system into simpler sub-problems similar to the modularity results in this chapter. In fact, all the results from this section are subsumed if we combine several processors of the DP framework. Nevertheless, the results of this section yield important theoretical insights showing the limitations of a corresponding DP framework.

In conclusion, we have shown the following results in this section. Here, (Yes) means that the property does not hold in general, but we have presented criteria that ensure that the property holds.

| Closed Under … | iAST | AST | iPAST | PAST | iSAST | SAST |
|---|---|---|---|---|---|---|
| Disjoint Unions | Yes | (Yes) | No | No | Yes | (Yes) |
| Shared Constructor Unions | Yes | (Yes) | No | No | No | No |
| Hierarchical Unions | (Yes) | (Yes) | No | No | No | No |

Table 5.4: Closure of probabilistic termination under unions.

SECTION 5.5

## Signature Extensions

Signature extensions often occur implicitly, and nearly all important properties of TRSs are closed under them. Moreover, many termination proofs do not require knowledge of the *exact* signature, but only the signature of the rules (all function symbols occurring in rules) is needed. For example, consider a termination proof via polynomial interpretations as in Corollary 3.2.64 or Corollary 3.4.39. Such a proof even works for an extended signature. Here, we can simply map all function symbols not occurring in rewrite rules to an arbitrary monotonic multilinear polynomial.

In this section, we study signature extensions of PTRSs and whether AST, PAST, and SAST are closed under those extensions. For brevity, we only consider full rewriting and innermost evaluation. We have already seen in Theorem 3.4.20 that $\mathtt{PAST}(\xrightarrow{\mathtt{f}}_{\mathcal{P}})$ and $\mathtt{PAST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}})$ are not closed under signature extensions. Let us restate the theorem here for the sake of completeness.

Theorem 5.5.1 – Signature Extensions for $\mathtt{PAST}(\xrightarrow{\mathtt{f}}_{\mathcal{P}})$ and $\mathtt{PAST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}})$.

There exist a finite PTRS $\mathcal{P}$ and signatures $\Sigma, \Sigma'$ with $\Sigma \subset \Sigma'$ such that $\mathtt{PAST}(\xrightarrow{\mathtt{f}}_{\mathcal{P}})$ holds over the signature $\Sigma$, but not over $\Sigma'$. Analogously, there exist a finite PTRS $\mathcal{P}$ and signatures $\Sigma, \Sigma'$ with $\Sigma \subset \Sigma'$ such that $\mathtt{PAST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}})$ holds over the signature $\Sigma$, but not over $\Sigma'$.

*Proof.* This is Theorem 3.4.20. *Proof QED*

Next, we consider AST and SAST. Signature extensions can be seen as special cases of disjoint unions, where the second PTRS $\mathcal{P}^2$ contains rules over the new signature that can never be applied (see the proof of Theorem 5.5.2). Hence, Theorems 5.4.2 and 5.4.8 already imply that $\mathtt{AST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}})$ and $\mathtt{SAST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}})$ are closed under signature extensions. Consequently, Corollary 5.4.18 implies that $\mathtt{AST}(\xrightarrow{\mathtt{f}}_{\mathcal{P}})$ and $\mathtt{SAST}(\xrightarrow{\mathtt{f}}_{\mathcal{P}})$ are closed under signature extensions for non-overlapping and linear PTRSs. We now show that this also holds for arbitrary PTRSs. So let $\mathcal{P}$ be an arbitrary finite PTRS over the signature $\Sigma^{\mathcal{P}}$ for the rest of this section. We consider two cases. To ease readability, we highlight the newly added function symbols from a signature extension $\Sigma'$.

First, let $\Sigma^{\mathcal{P}}$ contain only constants and unary symbols, e.g., $\Sigma^{\mathcal{P}} = \Sigma^{\mathcal{P}}_0 \uplus \Sigma^{\mathcal{P}}_1 = \{\mathsf{a}, \mathsf{b}\} \uplus \{\mathsf{f}\}$ where $\mathsf{f}$ is unary and $\mathsf{a}, \mathsf{b}$ are constants. If we extend $\Sigma^{\mathcal{P}}$ by a signature $\Sigma'$ that may also contain symbols of other arities, e.g., a symbol $\mathsf{c}$ of arity 2, and consider terms from $\mathcal{T}\big(\Sigma^{\mathcal{P}} \cup \Sigma', \mathcal{V}\big)$ like $\mathsf{f}(\mathsf{c}(\mathsf{f}(\mathsf{a}),\mathsf{f}(\mathsf{b})))$, then the fresh symbol $\mathsf{c}$ "completely separates" the function symbols from $\Sigma^{\mathcal{P}}$ occurring below and above $\mathsf{c}$. More precisely, instead of $\mathsf{f}(\mathsf{c}(\mathsf{f}(\mathsf{a}),\mathsf{f}(\mathsf{b})))$, it suffices to analyze the start terms $\mathsf{f}(\mathsf{a})$, $\mathsf{f}(\mathsf{b})$, and $\mathsf{f}(x_{\mathsf{c}})$ (where the $\mathsf{c}$-subterm is replaced by a fresh variable $x_{\mathsf{c}}$), which are all from $\mathcal{T}\big(\Sigma^{\mathcal{P}}, \mathcal{V}\big)$. The reason is that rewriting above $\mathsf{c}$ does not interfere with the terms below $\mathsf{c}$ and vice versa. Hence, if every rewrite sequence that starts with $\mathsf{f}(\mathsf{a})$, $\mathsf{f}(\mathsf{b})$, or $\mathsf{f}(x_{\mathsf{c}})$ converges

with probability 1, then so does the term $\mathsf{f}(\mathsf{c}(\mathsf{f}(\mathsf{a}),\mathsf{f}(\mathsf{b})))$. In other words, if $\mathtt{AST}(\xrightarrow{\mathsf{f}}_{\mathcal{P}})$ holds over the signature $\Sigma^{\mathcal{P}}$, then $\mathtt{AST}(\xrightarrow{\mathsf{f}}_{\mathcal{P}})$ also holds over the signature $\Sigma^{\mathcal{P}} \cup \Sigma'$. Furthermore, the expected derivation height of $\mathsf{f}(\mathsf{c}(\mathsf{f}(\mathsf{a}),\mathsf{f}(\mathsf{b})))$ is bounded by $3 \cdot C_{\max}$, where $C_{\max}$ is the maximum expected derivation height of the terms $\mathsf{f}(\mathsf{a})$, $\mathsf{f}(\mathsf{b})$, and $\mathsf{f}(x_{\mathsf{c}})$, by the Parallel Execution Lemma for $\mathtt{SAST}$ (Lemma 5.2.4). Thus, if we have $\mathtt{SAST}(\xrightarrow{\mathsf{f}}_{\mathcal{P}})$ over the signature $\Sigma^{\mathcal{P}}$, then we also have $\mathtt{SAST}(\xrightarrow{\mathsf{f}}_{\mathcal{P}})$ over the signature $\Sigma^{\mathcal{P}} \cup \Sigma'$.

Second, we consider the case where $\Sigma^{\mathcal{P}}$ itself already contains a function symbol $\mathsf{g}$ of arity at least 2. Again, the fresh symbols $\mathsf{c}$ of $\Sigma'$ separate the function symbols of $\Sigma^{\mathcal{P}}$ occurring above and below them. However, now the expected derivation height of a term like $\mathsf{f}(\mathsf{c}(\mathsf{f}(\mathsf{a}),\mathsf{f}(\mathsf{b})))$ from $\mathcal{T}\big(\Sigma^{\mathcal{P}} \cup \Sigma', \mathcal{V}\big)$ is not bounded by simply adding the expected derivation heights of the corresponding terms $\mathsf{f}(\mathsf{a})$, $\mathsf{f}(\mathsf{b})$, $\mathsf{f}(x_{\mathsf{c}})$ from $\mathcal{T}\big(\Sigma^{\mathcal{P}}, \mathcal{V}\big)$ anymore. The reason is that $\mathcal{P}$ might now duplicate subterms (e.g., there could be a rule like $\mathsf{f}(x) \to \{{}^{1}\mathsf{g}(x,x)\}$). However, for any term $t \in \mathcal{T}\big(\Sigma^{\mathcal{P}} \cup \Sigma', \mathcal{V}\big)$, we can now construct a term $t' \in \mathcal{T}\big(\Sigma^{\mathcal{P}}, \mathcal{V}\big)$ over the original signature $\Sigma^{\mathcal{P}}$ that has *at least* the same expected derivation height and *at most* the same convergence probability.

The construction works as follows: Let $\mathsf{g} \in \Sigma^{\mathcal{P}}$ be a symbol of arity 2 (if its arity is greater than two, then we use the term $\mathsf{g}(_, _, x, \dots, x)$ instead, where $x \in \mathcal{V}$). For example, if we extend the signature $\Sigma^{\mathcal{P}}$ by a symbol $\mathsf{c} \in \Sigma'$ of arity 2, then instead of a term like $\mathsf{f}(\mathsf{c}(\mathsf{f}(\mathsf{a}),\mathsf{f}(\mathsf{b})))$ from $\mathcal{T}\big(\Sigma^{\mathcal{P}} \cup \Sigma', \mathcal{V}\big)$, we can consider the term $\mathsf{f}(\mathsf{g}(\mathsf{f}(\mathsf{a}), \mathsf{g}(\mathsf{f}(\mathsf{b}), x_{\mathsf{c}})))$ from $\mathcal{T}\big(\Sigma^{\mathcal{P}}, \mathcal{V}\big)$ without the symbol $\mathsf{c}$, where we do not rewrite the newly added $\mathsf{g}$ symbols. So here, we replaced $\mathsf{c}(_, _)$ by $\mathsf{g}(_, \mathsf{g}(_, x_{\mathsf{c}}))$. Note that this construction works for full but not for innermost rewriting, since it may create new redexes with the symbol $\mathsf{g}$ that we may have to rewrite when using the innermost strategy. However, for innermost rewriting we have already proven closedness under signature extensions by Theorems 5.4.2 and 5.4.8, as explained above.

To summarize, this leads to the following theorem.

Theorem 5.5.2 – Signature Extensions for AST and SAST.
Let $\mathcal{P}$ be a finite PTRS, and let $\Sigma'$ be some fresh signature. Then

$$\begin{aligned}
\mathtt{AST}(\xrightarrow{\mathsf{i}}_{\mathcal{P}}) \text{ over } \Sigma^{\mathcal{P}} &\iff \mathtt{AST}(\xrightarrow{\mathsf{i}}_{\mathcal{P}}) \text{ over } \Sigma^{\mathcal{P}} \cup \Sigma' \\
\mathtt{AST}(\xrightarrow{\mathsf{f}}_{\mathcal{P}}) \text{ over } \Sigma^{\mathcal{P}} &\iff \mathtt{AST}(\xrightarrow{\mathsf{f}}_{\mathcal{P}}) \text{ over } \Sigma^{\mathcal{P}} \cup \Sigma' \\
\mathtt{SAST}(\xrightarrow{\mathsf{i}}_{\mathcal{P}}) \text{ over } \Sigma^{\mathcal{P}} &\iff \mathtt{SAST}(\xrightarrow{\mathsf{i}}_{\mathcal{P}}) \text{ over } \Sigma^{\mathcal{P}} \cup \Sigma' \\
\mathtt{SAST}(\xrightarrow{\mathsf{f}}_{\mathcal{P}}) \text{ over } \Sigma^{\mathcal{P}} &\iff \mathtt{SAST}(\xrightarrow{\mathsf{f}}_{\mathcal{P}}) \text{ over } \Sigma^{\mathcal{P}} \cup \Sigma'.
\end{aligned}$$

*Proof.* The results on innermost evaluation follow from Theorems 5.4.2 and 5.4.8. Let us consider the cases for full rewriting. It suffices to prove the non-trivial direction "$\Longrightarrow$". We consider two cases depending on the original signature $\Sigma^{\mathcal{P}}$.

*Proof*

**Step 1** $\Sigma^{\mathcal{P}}$ only contains constants and unary symbols

We first consider the case where $\Sigma^{\mathcal{P}}$ only contains constants and unary symbols. Let $\Sigma'$ be another signature containing fresh symbols, i.e., we have $\Sigma^{\mathcal{P}} \cap \Sigma' = \varnothing$. For any term from $\mathcal{T}\big(\Sigma^{\mathcal{P}} \cup \Sigma', \mathcal{V}\big)$ we now compute a multiset of terms from $\mathcal{T}\big(\Sigma^{\mathcal{P}}, \mathcal{V}\big)$ which can be regarded instead. Thus, we define a corresponding mapping A from $\mathcal{T}\big(\Sigma^{\mathcal{P}} \cup \Sigma', \mathcal{V}\big)$ to multisets of terms from $\mathcal{T}\big(\Sigma^{\mathcal{P}}, \mathcal{V}\big)$. For this definition, we need

two auxiliary mappings. The mapping $\mathrm{C} : \mathcal{T}\big(\Sigma^{\mathcal{P}} \cup \Sigma', \mathcal{V}\big) \to \mathcal{T}\big(\Sigma^{\mathcal{P}}, \mathcal{V}\big)$ replaces all topmost subterms with a root $f$ from $\Sigma'$ by the fresh variable $x_f$ and the mapping B maps any term $t \in \mathcal{T}\big(\Sigma^{\mathcal{P}} \cup \Sigma', \mathcal{V}\big)$ to the multiset that unites $\mathrm{A}(r)$ for the topmost subterms $r$ with $\mathrm{root}(r) \in \Sigma^{\mathcal{P}}$ occurring below a symbol from $\Sigma'$ in $t$.

$$
\begin{array}{ll}
\mathrm{C}(x) & = x, \text{ if } x \in \mathcal{V} \\
\mathrm{C}(f(t_1, \ldots, t_k)) & = f(\mathrm{C}(t_1), \ldots, \mathrm{C}(t_k)), \text{ if } f \in \Sigma^{\mathcal{P}} \\
\mathrm{C}(f(t_1, \ldots, t_k)) & = x_f, \text{ if } f \in \Sigma' \\
\\
\mathrm{A}(x) & = \{x\}, \text{ if } x \in \mathcal{V} \\
\mathrm{A}(f(t_1, \ldots, t_k)) & = \{f(\mathrm{C}(t_1), \ldots, \mathrm{C}(t_k))\} \cup \mathrm{B}(t_1) \cup \ldots \cup \mathrm{B}(t_k), \text{ if } f \in \Sigma^{\mathcal{P}} \\
\mathrm{A}(f(t_1, \ldots, t_k)) & = \mathrm{A}(t_1) \cup \ldots \cup \mathrm{A}(t_k), \text{ if } f \in \Sigma' \\
\\
\mathrm{B}(x) & = \varnothing, \text{ if } x \in \mathcal{V} \\
\mathrm{B}(f(t_1, \ldots, t_k)) & = \mathrm{B}(t_1) \cup \ldots \cup \mathrm{B}(t_k), \text{ if } f \in \Sigma^{\mathcal{P}} \\
\mathrm{B}(f(t_1, \ldots, t_k)) & = \mathrm{A}(t_1) \cup \ldots \cup \mathrm{A}(t_k), \text{ if } f \in \Sigma'
\end{array}
$$

Let $t \in \mathcal{T}\big(\Sigma^{\mathcal{P}} \cup \Sigma', \mathcal{V}\big)$ and let $\mathrm{A}(t) = \{q_1, \ldots, q_n\}$. Let $\mathsf{c}$ be another fresh $n$-ary constructor symbol (which is not from $\Sigma'$). Then instead of $t$, we consider the term $\mathsf{c}(q_1, \ldots, q_n)$. Every $\to_{\mathcal{P}}$-RT $\mathfrak{T}$ that starts with $t$ gives rise to an $\to_{\mathcal{P}}$-RT $\mathfrak{T}'$ that starts with $\mathsf{c}(q_1, \ldots, q_n)$ such that $|\mathfrak{T}| = |\mathfrak{T}'|$ and $\mathrm{edl}(\mathfrak{T}) = \mathrm{edl}(\mathfrak{T}')$. To see this, suppose that $t \to_{\mathcal{P}} \{{}^{p_1}s_1, \ldots, {}^{p_k}s_k\}$, $\mathrm{A}(t) = \{q_1, \ldots, q_n\}$, and $\mathrm{A}(s_i) = \{q_1^i, \ldots, q_{m_i}^i\}$ for all $1 \leq i \leq k$. Then there exists a $1 \leq j \leq n$ with $q_j \to_{\mathcal{P}} \{{}^{p_1}u_1, \ldots, {}^{p_k}u_k\}$ such that $\mathrm{A}(s_i)$ is a sub-multiset of $\{u_i\} \uplus (\mathrm{A}(t) \setminus \{q_j\})$ for all $1 \leq i \leq k$. Note that $m_i \leq n$, and we might even have $m_i < n$ if we use an erasing rule. Hence, every step in $\mathfrak{T}$ in a term $t$ can be mirrored by a step in $\mathfrak{T}'$ in a term $\mathsf{c}(q_1, \ldots, q_n)$.
Since the $q_i$ are terms from $\mathcal{T}\big(\Sigma^{\mathcal{P}}, \mathcal{V}\big)$, every $\to_{\mathcal{P}}$-RT that starts with $q_i$ converges with probability 1 or has bounded expected derivation length, respectively. Hence, by the Parallel Execution Lemmas (Lemmas 5.2.2 and 5.2.4) this also holds for every $\to_{\mathcal{P}}$-RT $\mathfrak{T}'$ that starts with $\mathsf{c}(q_1, \ldots, q_n)$. But then this also holds for every $\to_{\mathcal{P}}$-RT $\mathfrak{T}$ that starts with $t$.

**Step 2** $\Sigma^{\mathcal{P}}$ contains a symbol of arity at least 2

Now we consider the case where $\Sigma^{\mathcal{P}}$ contains at least one symbol $\mathsf{g}$ of arity at least 2. If $\mathsf{g}$ has an arity greater than two, then we use the term $\mathsf{g}(_, _, x, \ldots, x)$ instead, where $x \in \mathcal{V}$. Again, let $\Sigma'$ be another signature containing fresh symbols, i.e., w.l.o.g. we have $\Sigma^{\mathcal{P}} \cap \Sigma' = \varnothing$.
We now define a function $\phi : \mathcal{T}\big(\Sigma^{\mathcal{P}} \cup \Sigma', \mathcal{V}\big) \to \mathcal{T}\big(\Sigma^{\mathcal{P}}, \mathcal{V}\big)$ such that $\mathrm{edh}_{\to_{\mathcal{P}}}(t) \leq \mathrm{edh}_{\to_{\mathcal{P}}}(\phi(t))$ by

$$
\begin{array}{lll}
\phi(x) & = & x, \text{ if } x \in \mathcal{V}, \\
\phi(f(t_1, \ldots, t_k)) & = & f(\phi(t_1), \ldots, \phi(t_k)), \text{ if } f \in \Sigma^{\mathcal{P}}, \\
\phi(f) & = & x_f, \text{ if } f \in \Sigma' \text{ has arity 0}, \\
\phi(f(t)) & = & \mathsf{g}(\phi(t), x_f), \text{ if } f \in \Sigma' \text{ has arity 1, and} \\
\phi(f(t_1, \ldots, t_k)) & = & \mathsf{g}(\phi(t_1), \mathsf{g}(\phi(t_2), \ldots \mathsf{g}(\phi(t_k), x_f) \ldots)), \\
 & & \text{if } f \in \Sigma' \text{ has arity } k > 1.
\end{array}
$$

Every $\to_{\mathcal{P}}$-RT that starts with $t$ gives rise to an $\to_{\mathcal{P}}$-RT that starts with $\phi(t)$ using exactly the same rules, leading to the same convergence probability and the same expected derivation length. To see this, note that whenever $t \to_{\mathcal{P}} \{{}^{p_1}s_1, \ldots, {}^{p_k}s_k\}$, then also $\phi(t) \to_{\mathcal{P}} \{{}^{p_1}\phi(s_1), \ldots, {}^{p_k}\phi(s_k)\}$. *QED*

All the techniques to prove $\mathtt{AST}(\xrightarrow{\mathtt{f}}_{\mathcal{P}})$, $\mathtt{PAST}(\xrightarrow{\mathtt{f}}_{\mathcal{P}})$, and $\mathtt{SAST}(\xrightarrow{\mathtt{f}}_{\mathcal{P}})$ we present in this thesis in fact prove $\mathtt{AST}(\xrightarrow{\mathtt{f}}_{\mathcal{P}})$ or $\mathtt{SAST}(\xrightarrow{\mathtt{f}}_{\mathcal{P}})$, which are closed under signature extensions by Theorem 5.5.2. So none of these techniques can distinguish between $\mathtt{PAST}(\xrightarrow{\mathtt{f}}_{\mathcal{P}})$ and $\mathtt{SAST}(\xrightarrow{\mathtt{f}}_{\mathcal{P}})$, i.e., whenever these techniques can prove $\mathtt{PAST}(\xrightarrow{\mathtt{f}}_{\mathcal{P}})$, then $\mathtt{SAST}(\xrightarrow{\mathtt{f}}_{\mathcal{P}})$ holds as well. This also holds for every technique that is closed under signature extensions, i.e., that can prove $\mathtt{PAST}(\xrightarrow{\mathtt{f}}_{\mathcal{P}})$ for a PTRS $\mathcal{P}$ over $\Sigma^{\mathcal{P}} \cup \Sigma'$ whenever it can prove $\mathtt{PAST}(\xrightarrow{\mathtt{f}}_{\mathcal{P}})$ for $\mathcal{P}$ over $\Sigma^{\mathcal{P}}$. The reason for this is that if such a technique is able to prove $\mathtt{PAST}(\xrightarrow{\mathtt{f}}_{\mathcal{P}})$ for a (finite) PTRS $\mathcal{P}$, then it is able to prove $\mathtt{PAST}(\xrightarrow{\mathtt{f}}_{\mathcal{P}})$ for $\mathcal{P}$ over an extended signature that contains at least one binary function symbol. But then by Theorem 3.4.24 it follows that $\mathcal{P}$ is $\mathtt{SAST}(\xrightarrow{\mathtt{f}}_{\mathcal{P}})$ as well. Whether there exists some (non-trivial) technique that can prove $\mathtt{PAST}(\xrightarrow{\mathtt{f}}_{\mathcal{P}})$ of (finite) PTRSs that are not $\mathtt{SAST}(\xrightarrow{\mathtt{f}}_{\mathcal{P}})$ remains future work. Due to all of these results we presented in this chapter, the stronger notion $\mathtt{SAST}(\xrightarrow{\mathtt{f}}_{\mathcal{P}})$ is more interesting and robust, and should be investigated in future research instead of $\mathtt{PAST}(\xrightarrow{\mathtt{f}}_{\mathcal{P}})$.

| Closed Under … | iAST | AST | iPAST | PAST | iSAST | SAST |
|---|---|---|---|---|---|---|
| Signature Extensions | Yes | Yes | No | No | Yes | Yes |

Table 5.5: Closure of probabilistic termination under signature extensions.

SECTION 5.6

## Related Work & Open Problems

For ordinary rewriting, modularity has been studied extensively, including foundational results for disjoint systems [Toy87a, Rus87, Mid90, TKB95], for systems with constructor sharing [Gra94, Ohl94], for hierarchical systems [Rao94, Rao95, Ohl02b], and for the relation between innermost and full rewriting [Gra95]. For an overview of this line of work and further refinements, see [Ohl02a, Gra12]. Our modularity and composability results for PTRSs can be viewed as a continuation of this classical line of work into the probabilistic setting.

This chapter also connects to a broader compositionality perspective in probabilistic verification: in imperative probabilistic settings, compositional proof rules are central for scalability, but expectations are often not preserved by naive program composition [MM05, MMKK18]. In particular, compositionality of proofs for positive almost-sure termination and modular verification of almost-sure termination have been studied for imperative probabilistic programs in [FFH15] and [HFCG19], respectively. Our positive and negative closure results for PTRSs mirror this phenomenon in a rewrite-based model. Similarly, modular complexity analysis of TRSs [ZK14] can be seen as a non-probabilistic counterpart of our composability results for the expected runtime.

In addition to termination and runtime analysis, modularity is a highly relevant topic in confluence analysis of TRSs as well. There are several works investigating under which conditions on the TRSs confluence is a modular property, see for example [Toy87b, KO92, WKMTd94, Ohl95].

As future work, one can extend the work presented in this chapter by a more fine-grained analysis of unions of PTRSs. For example, it is interesting to investigate unions of systems that may share rewrite rules. Such results would better match realistic program libraries where strict disjointness is rare.

Open Problem 5.6.1 – Finer Modularity Results for PTRSs.

Develop modularity criteria for richer combinations than hierarchical unions where both parts of the PTRSs can share rewrite rules.

Another interesting direction is to develop modularity results for confluence in the probabilistic setting.

Open Problem 5.6.2 – Modularity Results for Confluence of PTRSs.

Under which properties is confluence a modular property of PTRSs?

Finally, as shown in Section 5.5, `PAST` is not closed under signature extensions. Thus, it is an interesting question what a technique to prove `PAST` for systems that are not `SAST` would look like and whether it has to exploit the precise signature of the system.

Open Problem 5.6.3 – Techniques Specifically for `PAST`.

Develop automated techniques that can prove `PAST` for PTRSs that are not `SAST`, potentially by exploiting the precise signature of the system.

# 6 Decidable Subclasses

In Section 3.4.5 we have already seen a first glimpse of how to prove AST, how to prove SAST, and how to prove upper bounds on the expected derivational complexity of a given PTRS via supermartingales. All of this can be fully automated with SMT solvers. Unfortunately, supermartingales based on interpretations are quite limited in power. In general, we cannot hope to find an algorithm that completely solves the problem, since term rewriting is a TURING-complete programming language, and for such TURING-complete programming languages termination is an undecidable property [HL78], even for TRSs that consist of only a single rewrite rule [Dau92]. Before we introduce the more complex techniques on how to automatically prove AST (see Chapter 7), on how to automatically prove SAST and upper complexity bounds (see Chapter 8), and on how to automatically disprove AST and SAST (see Chapter 9), we shall investigate how difficult the problem of termination analysis for PTRSs is in general.

There are multiple approaches to analyze the "difficulty" of a decision problem in computer science. One approach is to find conditions under which the problem becomes decidable, e.g., one only considers a certain type of PTRSs where the analysis of AST becomes decidable. Alternatively, one can relate the problem to other undecidable problems. Here, one investigates what other undecidable problems would have to be decidable in order to decide AST of arbitrary PTRSs. The latter gives rise to an order of undecidable problems stating that problem $A$ is "harder" than problem $B$ if $A$ remains undecidable under the assumption that an oracle can decide problem $B$. For example, universal termination remains undecidable even if we can solve the halting problem (termination on a given input), and thus, universal termination is "harder" to solve.

There are different hierarchies that order decision problems by "hardness". One of the most prominent hierarchies is the *arithmetical hierarchy*. For ordinary term rewriting, the precise levels of several properties, including termination, within the arithmetical hierarchy were determined in [EGSZ11], and corresponding results for derivational and runtime complexity were presented in [SS11]. In [KK15, KKM19], the problem of deciding AST and SAST[1] has been investigated for imperative programs, locating the decision problems within the arithmetical hierarchy. Astonishingly, the decision problem for AST is as hard as the decision problem for certain termination, where both

[1] In [KKM19], only imperative programs without non-determinism have been investigated for which PAST = SAST holds. While [KKM19] only talks about PAST, the definition used in [KKM19] for PAST relates to our definition of SAST, i.e., having a bound on the expected derivation length of a computation.

are meant universally (AST and certain termination for all possible start terms) as is always the case in this thesis. If one only considers the halting problem for AST and certain termination, where one has an additional input on which the algorithm has to terminate, then AST is indeed "harder" than certain termination. The decision problem for SAST is one "hardness"-level higher and thus more difficult than AST and ordinary termination (again seen as the universal decision problem). In [MS24] the problem of deciding PAST has been located in a different hierarchy, namely the *analytical hierarchy*.

Since the corresponding decision problems for imperative probabilistic programs have already been located in these hierarchies, we consider the first approach and try to restrict the structure of PTRSs such that AST becomes decidable. In the non-probabilistic setting, there exists a well-known subclass of TRSs where termination is decidable: *right-ground TRSs*. A TRS is *right-ground* if all of its right-hand sides are ground terms, i.e., do not contain any variables. For example, the rule $\mathsf{eq}(x, x) \to \mathsf{true}$ is right-ground, but $\mathsf{leq}(\mathsf{s}(x), \mathsf{s}(x)) \to \mathsf{leq}(x, x)$ is not right-ground. Since there are no variables in the right-hand sides of rules, we cannot store arbitrary information inside recursive function calls.

We show in this chapter why the decision procedure for termination of right-ground TRSs does not generalize to the probabilistic setting. Consequently, we try to restrict the subclass even further such that we can relate it to a stochastic system with a known decision procedure for AST. We have already seen multiple stochastic systems where AST is decidable, e.g., random walks and finite Markov decision processes. Let us briefly explain why these two stochastic systems do not yield powerful results if we try to restrict PTRSs to behave like them.

**Finite MDPs?** We have already seen many examples on how one can view a PTRS as an MDP. Since termination analysis for finite MDPs is decidable, an obvious idea is to find properties of $\mathcal{P}$ such that the corresponding MDP $\mathcal{M}_{\mathcal{P}}$ of a PTRS $\mathcal{P}$ is finite, i.e., only finitely many terms are reachable from each start term.

Example 6.0.1. Consider the PTRS $\mathcal{P}_{\mathsf{mdp}}$ with the three rules $\mathsf{a} \to \{^{1/2}\mathsf{a}, ^{1/2}\mathsf{b}\}$, $\mathsf{a} \to \{^{1}\mathsf{c}\}$, and $\mathsf{b} \to \{^{1/2}\mathsf{a}, ^{1/2}\mathsf{c}\}$. We can view this PTRS as an MDP, see Figure 6.1. Since the corresponding MDP is finite, we can use the technique from Theorem 2.4.33 to compute the termination probabilities for each start term.

However, the underlying MDP of a PTRS is very rarely finite. In fact, if we have a single function symbol of arity $> 0$, e.g., a function symbol $\mathsf{g}$ of arity 1, then we can already build infinitely many terms $\mathsf{g}(x)$, $\mathsf{g}(\mathsf{g}(x))$, $\mathsf{g}(\mathsf{g}(\mathsf{g}(x)))$, etc., which means that already a single non-constant function symbol yields infinitely many states in the corresponding MDP, and already a single rule like $\mathsf{f}(x) \to \{^{1}\mathsf{f}(\mathsf{g}(x))\}$ reaches infinitely many of them from a single start term. Therefore, such an approach would only be applicable to very restricted PTRSs, e.g., PTRSs that only contain constant function symbols.

**Random Walks?** Random walks allow for infinitely many states, but the transition relation between the different states is strongly restricted. In fact, a random walk only allows for a single possible reduction step (with multiple probabilistic outcomes) in each state, and it needs to be the same for each state.

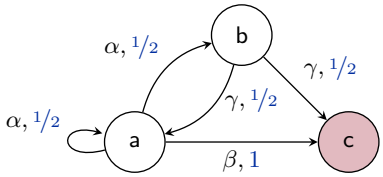



Figure 6.1: $\mathcal{P}_{\mathsf{mdp}}$ illustrated as a finite MDP.

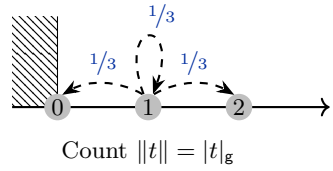



Figure 6.2: $\mathcal{P}_{\mathsf{lazy}}$ illustrated as an RWP.

Example 6.0.2. Consider the PTRS $\mathcal{P}_{\mathsf{lazy}}$ with the only rule $\mathsf{g}(x) \to \{^{1/3}\mathsf{g}(\mathsf{g}(x)), ^{1/3}\mathsf{g}(x), ^{1/3}x\}$. We can view this PTRS as a random walk, see Figure 6.2. Since the corresponding random walk is symmetric, we know that the PTRS $\mathcal{P}_{\mathsf{lazy}}$ is `AST`.

However, the restriction to a single rule with only unary function symbols is unsatisfactory as well, and does not capture practical programs modeled by PTRSs.

**Stochastic Context-Free Grammars!** To decide `AST` for a broad class of PTRSs we present yet another stochastic system where analyzing `AST` is decidable: *stochastic context-free grammars*. Stochastic context-free grammars (or SCFGs) are a generalization of context-free grammars with a probabilistic choice that replaces the non-deterministic one. So this parallels the step from graphs to Markov chains rather than the step to MDPs, since we have no non-determinism in the system. Still, the system allows for infinitely many states and allows for several different transition relations depending on the state we are in. Note that SCFGs are not the only stochastic systems beyond Markov chains, MDPs, and random walk programs for which `AST` is decidable. For example, `AST` is also decidable for recursive Markov chains [EY09, EY15], probabilistic pushdown automata [BEKK13, PBC25], and restricted classes of probabilistic higher-order recursion schemes [KDLG20, DLFP26], see Section 6.5 for an overview. We focus on SCFGs in this chapter, as they are the conceptually simplest of these systems, and our goal is to obtain a first decidability result for a non-trivial subclass of PTRSs in the first place.

In the remainder of this chapter, we first introduce SCFGs and show how one can decide `AST` for them. Then, we show how one can transform every PTRS $\mathcal{P}$ from a certain subclass into an SCFG $\mathcal{G}$ such that $\mathcal{P}$ is `AST` if and only if $\mathcal{G}$ is `AST`. For this transformation we require that the PTRS is (`G`) ground (rules only contain ground terms), (`NO`) non-overlapping (to remove additional non-determinism in the rule selection), and (`TR`) tail-recursive (there are no nested defined symbols as in a context-free grammar). This yields a decision procedure for PTRSs that are `G`, `NO`, and `TR`. Conversely, every SCFG can be encoded as such a PTRS (see Example 6.2.10), so this subclass is essentially a syntactic counterpart of SCFGs within term rewriting. Afterwards, we show how to broaden the class from ground to right-ground (`RG`) similar to the decision procedure in the non-probabilistic setting, i.e., we present another decision procedure for PTRSs that are `RG`, `NO`, and `TR`.

### Contributions

We present the following contributions in this chapter, which are all novel and unpublished ( !⋆! ).

!⋆! Definition of the subclass of ground, non-overlapping, tail-recursive PTRSs (GNT), and a transformation from PTRSs that are GNT to SCFGs that preserves `AST`. This yields the first decision procedure for `AST` of a non-trivial subclass of PTRSs (Lemma 6.2.20 and Theorem 6.2.22).

!⋆! Extension of the decidability results to the broader subclass of right-ground, non-overlapping, tail-recursive PTRSs (RNT) via a transformation to GNT PTRSs that preserves `AST` (Lemma 6.3.4 and Theorem 6.3.5).

## Structure

We first recapitulate the notion of SCFGs in Section 6.1 and show how one can decide `AST` for a given SCFG. Then, we investigate the properties of a PTRS needed to transform it into a termination-equivalent SCFG in Section 6.2. Here, we only show that every PTRS that is *non-overlapping*, *tail-recursive*, and *ground* can be transformed into an SCFG. In Section 6.3 we show how to extend the result to *right-ground* PTRSs that are *non-overlapping* and *tail-recursive*. We end this chapter by performing an experimental evaluation in Section 6.4, and discussing related work and giving important open problems for future work in Section 6.5.

SECTION 6.1

## Stochastic Grammars

Grammars are a well-known computational model to describe formal languages. They can be seen as potentially restricted forms of *string rewrite systems*, i.e., TRSs where all the left-hand sides and all the right-hand sides are *strings*. Strings are terms where every occurring function symbol has arity 1. Instead of $\mathsf{a}(\mathsf{b}(\mathsf{b}(\mathsf{a}(x))))$ one typically omits the brackets and the variable and simply writes $\mathsf{abba}$ for strings. So this chapter takes a closer look at TRSs containing rules like $\mathsf{a} \to \mathsf{bab}$, whose left-hand side consists of a single symbol, and analyzes their termination.

According to the well-known Chomsky hierarchy, there are four different types of grammars that all have different expressive power.

Type-0: Grammars able to describe all *recursively enumerable* languages.

Type-1: Grammars able to describe *context-sensitive* languages.

Type-2: Grammars able to describe *context-free* languages.

Type-3: Grammars able to describe *regular* languages.

The lower the type of the grammar, the more expressive it is, but also the harder it is to analyze in terms of termination. For the most general grammars of type 0, termination is undecidable. In fact, those grammars are string rewrite systems without any restriction, and thus, they are a Turing-complete programming language similar to term rewrite systems. In this section, we focus on grammars of type 2, namely

*context-free grammars*, where (universal) termination is still decidable. Note that in the probabilistic setting, termination-like properties for probabilistic grammars have been studied extensively for stochastic context-free grammars, while research on stochastic context-sensitive grammars is scarce. Since we want to utilize the probabilistic version of the grammars, we choose to focus on type 2 grammars.

### 6.1.1 Context-Free Grammars

A context-free grammar (CFG) is a string rewrite system where the left-hand side only contains a single function symbol. Similar to defined symbols and constructor symbols, one distinguishes between *non-terminal symbols*, i.e., those function symbols that occur on the left-hand sides, and *terminal symbols*, i.e., all other function symbols that cannot be evaluated further.

Definition 6.1.1 – Context-Free Grammar.
*A context-free grammar is a tuple* $\mathcal{G} = (N, T, P)$ *where*

- $N$ *is a finite set of non-terminals,*
- $T$ *is a finite set of terminals with* $N \cap T = \varnothing$*,*
- $P \subseteq N \times (N \cup T)^*$ *is a finite set of production rules. Instead of* $(A, \alpha) \in P$ *we simply write* $A \to \alpha$*.*

Throughout this chapter, we assume that every non-terminal has at least one production rule (for SCFGs, this holds by definition).

Example 6.1.2 (Context-Free Grammar). As running example we consider the Knudsen-Hein grammar [KH99, GR14]. It is defined as the CFG $\mathcal{G}_{\mathsf{KH}} = (N, T, P)$ with $N = \{S, L, F\}$, $T = \{s, d\}$, and the following six production rules in $P$.

$$\begin{array}{lcl} S & \to & L \\ S & \to & LS \end{array} \qquad \begin{array}{lcl} L & \to & s \\ L & \to & dFd \end{array} \qquad \begin{array}{lcl} F & \to & dFd \\ F & \to & LS \end{array}$$

Grammars can also be seen as an abstract reduction system on words or strings.

Definition 6.1.3 – ARS of a CFG.
*Let* $\mathcal{G} = (N, T, P)$ *be a CFG. The CFG* $\mathcal{G}$ *induces an ARS* $((N \cup T)^*, \to_{\mathcal{G}})$ *via* $\alpha \to_{\mathcal{G}} \beta$ *if* $\alpha = \pi A \tau$ *with* $\pi \in T^*$, $\tau \in (N \cup T)^*$, $A \to \zeta \in P$, *and* $\beta = \pi \zeta \tau$.

Notice the condition $\pi \in T^*$, namely that the prefix of $\alpha$ before the non-terminal $A$ contains only terminal symbols. Due to this condition we essentially fix some kind of "leftmost" evaluation strategy. We elaborate on this in the following sections.

Example 6.1.4. For the CFG $\mathcal{G}_{\mathsf{KH}}$ we have $S \to_{\mathcal{G}}^* dssd$, because

$$\underline{S} \to_{\mathcal{G}} \underline{L} \to_{\mathcal{G}} d\underline{F}d \to_{\mathcal{G}} d\underline{L}Sd \to_{\mathcal{G}} ds\underline{S}d \to_{\mathcal{G}} ds\underline{L}d \to_{\mathcal{G}} dssd.$$

Normally, one is interested in the language described by a given CFG where we fix some initial non-terminal. The language is then defined as all words over terminal symbols that are reachable from the initial non-terminal. In fact, the normal forms of $\to_{\mathcal{G}}$ are exactly the finite words over $T$, i.e., $\mathtt{NF}_{\to_{\mathcal{G}}} = T^*$. For instance, Example 6.1.4 shows that the word $dssd$ is in the language generated by the grammar $\mathcal{G}_{\mathsf{KH}}$ and the

initial non-terminal $S$. Instead of the described language, we are only interested in the termination behavior. Termination of CFGs is decidable since this problem just boils down to finding a cycle in the control flow graph of the non-terminals, where we have an edge from $A \in N$ to $B \in N$ if there exists a rule $A \to \alpha \in P$ with $B$ occurring in $\alpha$.

### 6.1.2 Stochastic Context-Free Grammars

Stochastic context-free grammars extend non-probabilistic context-free grammars by replacing non-determinism with probabilistic choices, similar to Markov chains that are based on ordinary graphs. In this way, non-determinism is completely removed.

Definition 6.1.5 – Stochastic Context-Free Grammar.
*A stochastic context-free grammar (SCFG) is a tuple* $\mathcal{G} = (N, T, P)$ *where*

- $N$ *is a finite set of non-terminals,*
- $T$ *is a finite set of terminals with* $N \cap T = \varnothing$,
- $P : N \times (N \cup T)^* \to [0, 1]$ *assigns probabilities to production rules such that for all* $A \in N$ *we have* $\sum_{\zeta \in (N \cup T)^*} P(A, \zeta) = 1$ *and the support* $\mathrm{Supp}(P(A, \cdot)) = \{\zeta \in (N \cup T)^* \mid P(A, \zeta) > 0\}$ *is finite. Instead of* $P(A, \zeta) = p > 0$ *we simply write* $A \xrightarrow{p} \zeta$.

Example 6.1.6. A probabilistic version of the Knudsen-Hein grammar is defined as the SCFG $\mathcal{G}^p_{\mathsf{KH}} = (N, T, P)$ with $N = \{S, L, F\}$, $T = \{s, d\}$, and $P$ is defined as

$$\begin{array}{llll} S \xrightarrow{p_1} L & L \xrightarrow{p_3} s & F \xrightarrow{p_5} dFd \\ S \xrightarrow{p_2} LS & L \xrightarrow{p_4} dFd & F \xrightarrow{p_6} LS \end{array}$$

where $p_1, \ldots, p_6 > 0$ are probabilities with $p_i + p_{i+1} = 1$ for all $i \in \{1, 3, 5\}$.

Next, we formalize how an SCFG induces a PARS to get a formal definition of the termination probability of an SCFG.

Definition 6.1.7 – PARS of an SCFG.
*Let* $\mathcal{G} = (N, T, P)$ *be an SCFG. The SCFG* $\mathcal{G}$ *induces a PARS* $((N \cup T)^*, \to_{\mathcal{G}})$ *via* $\alpha \to_{\mathcal{G}} \{{}^{p_1}\beta_1, \ldots, {}^{p_k}\beta_k\}$ *if* $\alpha = \pi A \tau$ *with* $\pi \in T^*$ *and* $\tau \in (N \cup T)^*$, $\mathrm{Supp}(P(A, \cdot)) = \{\zeta_1, \ldots, \zeta_k\}$, *and for all* $1 \leq i \leq k$ *we have* $\beta_i = \pi \zeta_i \tau$ *and* $P(A, \zeta_i) = p_i$.

Example 6.1.8. Reconsider the SCFG $\mathcal{G}^p_{\mathsf{KH}} = (N, T, P)$ from Example 6.1.6. The corresponding PARS has the universe $(\{S, L, F\} \cup \{s, d\})^*$ and the reduction relation $\to_{\mathcal{G}^p_{\mathsf{KH}}}$ contains, e.g., $\underline{S} \to_{\mathcal{G}^p_{\mathsf{KH}}} \{{}^{p_1}L, {}^{p_2}LS\}$, $\underline{L} \to_{\mathcal{G}^p_{\mathsf{KH}}} \{{}^{p_3}s, {}^{p_4}dFd\}$, and $\underline{F} \to_{\mathcal{G}^p_{\mathsf{KH}}} \{{}^{p_5}dFd, {}^{p_6}LS\}$. But also $\underline{S}S \to_{\mathcal{G}^p_{\mathsf{KH}}} \{{}^{p_1}LS, {}^{p_2}LSS\}$ holds, as we can reduce the first $S$ using the production rule for $S$. We do not have $S\underline{L} \to_{\mathcal{G}^p_{\mathsf{KH}}} \{{}^{p_3}Ss, {}^{p_4}SdFd\}$ due to the leftmost evaluation.

Notions that have been defined for PARSs, like `AST`, can also be used for SCFGs by requiring the notion for the induced PARS $\to_{\mathcal{G}}$ of an SCFG $\mathcal{G}$.

Once again, we only consider leftmost evaluations within the induced PARS. This definition removes any non-determinism from the system. Hence, for every $\alpha \in (N \cup T)^*$

there is a unique infinite $\rightrightarrows_{\mathcal{G}}$-reduction $\vec{\mu}$ with $\mu_0 = \{^1\alpha\}$, and we call $q_\alpha = |\vec{\mu}|$ the *termination probability* of $\alpha$. For $\alpha = \pi_0 A_1 \pi_1 \cdots A_k \pi_k$ with $\pi_0, \ldots, \pi_k \in T^*$ and $A_1, \ldots, A_k \in N$, we have $q_\alpha = \prod_{i=1}^{k} q_{A_i}$. In particular, $\mathtt{AST}(\to_{\mathcal{G}})$ holds if and only if $q_A = 1$ for all $A \in N$. Our definition of SCFGs was taken from [EY09] (including the leftmost evaluation strategy), from which also the following decision procedure for AST originates.

Termination analysis of SCFGs consists of two main phases and is very similar to the termination analysis of Markov chains. First, we detect *unproductive* non-terminals, i.e., those non-terminals that are not able to reach any normal form at all. Remember that $\mathtt{NF}_{\to_{\mathcal{G}}} = T^*$, i.e., the normal forms of an SCFG are exactly those words that do not contain any non-terminal anymore. An SCFG where every non-terminal symbol can reach (without considering any probabilities) a normal form is called *productive*.

For a formal definition of a productive grammar, we first define how an SCFG induces a CFG (without probabilities) by replacing probabilistic choices with non-determinism.

Definition 6.1.9 – Probability Removal.

*Let $\mathcal{G} = (N, T, P)$ be an SCFG. By $\mathrm{np}(\mathcal{G}) = (N, T, \mathrm{np}(P))$ we denote the induced non-probabilistic CFG of $\mathcal{G}$ where $\mathrm{np}(P) = \{A \to \zeta \mid P(A, \zeta) > 0\}$.*

Example 6.1.10. The induced non-probabilistic CFG for the probabilistic version $\mathcal{G}^p_{\mathsf{KH}}$ from Example 6.1.6 is the grammar $\mathcal{G}_{\mathsf{KH}}$ from Example 6.1.2, i.e., $\mathrm{np}(\mathcal{G}^p_{\mathsf{KH}}) = \mathcal{G}_{\mathsf{KH}}$.

Definition 6.1.11 – Productive SCFG.

*An SCFG $\mathcal{G} = (N, T, P)$ is called productive if for every $A \in N$ there exists a derivation $A \to^*_{\mathrm{np}(\mathcal{G})} \alpha$ with $\alpha \in T^*$.*

Checking whether an SCFG is productive can be done in linear time via simple graph algorithms [EY09].

Example 6.1.12. The SCFG $\mathcal{G}^p_{\mathsf{KH}}$ from Example 6.1.6 is productive, because we have

$$\begin{aligned} &\underline{S} \to_{\mathcal{G}_{\mathsf{KH}}} \underline{L} \to_{\mathcal{G}_{\mathsf{KH}}} s \in T^* \\ &\underline{L} \to_{\mathcal{G}_{\mathsf{KH}}} s \in T^* \\ &\underline{F} \to_{\mathcal{G}_{\mathsf{KH}}} \underline{L}S \to_{\mathcal{G}_{\mathsf{KH}}} s\underline{S} \to_{\mathcal{G}_{\mathsf{KH}}} s\underline{L} \to_{\mathcal{G}_{\mathsf{KH}}} ss \in T^*. \end{aligned}$$

If an SCFG is not productive, then there exists a non-terminal $A \in N$ with termination probability $q_A = 0$, and thus, the SCFG is not AST. If the SCFG is productive, then it is not necessarily AST, but there is still the possibility that the probability of termination is strictly less than 1.

Counterexample 6.1.13 (Positively Drifting Random Walk as an SCFG). Consider the SCFG $\mathcal{G}_{\mathsf{rw}} = (N, T, P)$ with $N = \{S\}$, $T = \varnothing$, and $P$ is defined in Figure 6.3.

$$S \xrightarrow{2/3} SS$$
$$S \xrightarrow{1/3} \varepsilon$$

Figure 6.3: Definition of $P$.

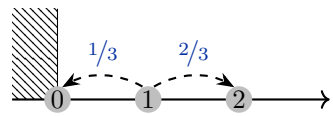



Figure 6.4: $\mathcal{G}_{\mathsf{rw}}$ illustrated as an RWP.

This grammar models the positively drifting random walk depicted in Figure 6.4. Even though the symbol $S$ can reach the normal form $\varepsilon$, its termination probability is only

$1/2$.

Similar to the termination analysis of Markov chains, the goal is to create a system of equations such that the least solution of the system of equations yields the probability of termination. For each of the non-terminals, we not only have to consider *what* other non-terminals are created, but in addition, we have to count *how many* of them are created. The *(first) moment matrix* $B_{\mathcal{G}}$ of an SCFG $\mathcal{G}$ describes the expected number of occurrences of each non-terminal that are created by a single derivation step.

Definition 6.1.14 – Moment Matrix of an SCFG.
*Let $\mathcal{G} = (N, T, P)$ be an SCFG and let $N = \{S_1, \ldots, S_n\}$. Then the (first) moment matrix $B_{\mathcal{G}} \in \mathbb{R}_{\geq 0}^{n \times n}$ corresponding to $\mathcal{G}$ is defined by*

$$(B_{\mathcal{G}})_{i,j} = \sum_{\alpha \in \mathrm{Supp}(P(S_i, \cdot))} P(S_i, \alpha) \cdot |\alpha|_{S_j}$$

*for each $i, j \in \{1, \ldots, n\}$. Here, for each $\alpha \in (N \cup T)^*$ we denote by $|\alpha|_A$ the number of occurrences of $A$ in the word $\alpha$, where $A \in N \cup T$.*

So the entry in the $i$-th row and the $j$-th column of $B_{\mathcal{G}}$ is the expected number of non-terminals $S_j$ that are generated from a single non-terminal $S_i$ in a single step. Compared to Markov chains, where we were either able to move towards a state or not, we have an additional counting requirement to check how often we generate a certain state, because each of those generated states now has to move to a normal form individually.

Example 6.1.15 (Moment Matrix). Consider $\mathcal{G}^p_{\mathsf{KH}}$ where we renamed the non-terminals $S$, $L$, and $F$ to $S_1$, $S_2$, and $S_3$, respectively, i.e.,

$$\begin{array}{llll} S_1 \xrightarrow{p_1} S_2 & \qquad S_2 \xrightarrow{p_3} s & \qquad S_3 \xrightarrow{p_5} dS_3d \\ S_1 \xrightarrow{p_2} S_2S_1 & \qquad S_2 \xrightarrow{p_4} dS_3d & \qquad S_3 \xrightarrow{p_6} S_2S_1. \end{array}$$

The moment matrix of $\mathcal{G}^p_{\mathsf{KH}}$ is

$$B_{\mathcal{G}^p_{\mathsf{KH}}} = \begin{pmatrix} p_2 & p_1 + p_2 & 0 \\ 0 & 0 & p_4 \\ p_6 & p_6 & p_5 \end{pmatrix} = \begin{pmatrix} p_2 & 1 & 0 \\ 0 & 0 & p_4 \\ p_6 & p_6 & p_5 \end{pmatrix}.$$

For example, we have $(B_{\mathcal{G}^p_{\mathsf{KH}}})_{1,2} = p_1 + p_2$, because in both production rules $S_1 \xrightarrow{p_1} S_2$ and $S_1 \xrightarrow{p_2} S_2S_1$ we create a single $S_2$, and we sum up the probabilities $p_1$ and $p_2$.

In [EY09] it has been shown that almost-sure termination of a productive SCFG can be characterized via the *spectral radius* of the moment matrix. The spectral radius is the largest absolute value of all *eigenvalues* of a matrix. It determines the long-term growth rate of a linear dynamic system, and in our case, it determines whether a productive SCFG terminates with probability 1.

Before we present the result of [EY09], we recapitulate some terminology of linear algebra.

Definition 6.1.16 – Eigenvalue, Spectrum, and Spectral Radius.

*Let $A \in \mathbb{R}^{n \times n}$ be a matrix.*

a) *A value $\lambda \in \mathbb{C}$ is an eigenvalue of $A$ if there exists a vector $x \in \mathbb{C}^n$ with $x \neq 0$ and $Ax = \lambda x$. Then $x$ is an eigenvector for the eigenvalue $\lambda$.*

b) *The spectrum spectrum$(A)$ of $A$ is the set of its eigenvalues $\{\lambda_1, \ldots, \lambda_m\} \subseteq \mathbb{C}$.*

c) *The spectral radius $\rho(A)$ of $A$ is the maximum absolute value of any eigenvalue.*

Example 6.1.17 (Spectrum). Let $A = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$, $\vec{x} = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$ and $\vec{y} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$. We have

- $A\vec{x} = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} 1 \\ -1 \end{pmatrix} = \begin{pmatrix} 2-1 \\ 1-2 \end{pmatrix} = 1 \cdot \begin{pmatrix} 1 \\ -1 \end{pmatrix} = 1\vec{x}$, and
- $A\vec{y} = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 2+1 \\ 1+2 \end{pmatrix} = 3 \cdot \begin{pmatrix} 1 \\ 1 \end{pmatrix} = 3\vec{y}$.

Therefore, $\vec{x}$ is an eigenvector for the eigenvalue 1, and $\vec{y}$ is an eigenvector for the eigenvalue 3. All in all, we have $spectrum(A) = \{1, 3\}$ (there are no further eigenvalues, since a $2 \times 2$ matrix has at most two eigenvalues) and $\rho(A) = 3$.

The main result from [EY09] that we want to use for our analysis of PTRSs is the following Theorem 6.1.18. It states that a productive SCFG is AST (terminates with probability 1 for each possible start word) if and only if the corresponding moment matrix $B_{\mathcal{G}}$ has spectral radius at most 1.

Theorem 6.1.18 – AST via the Spectral Radius of the Moment Matrix, [EY09].

Let $\mathcal{G}$ be a productive SCFG with moment matrix $B_{\mathcal{G}}$. Then $\mathcal{G}$ is AST if and only if $\rho(B_{\mathcal{G}}) \leq 1$.

*Proof.* In [EY09], AST is called *consistency*, and only a fixed initial non-terminal is considered. Recall that $\mathcal{G}$ is AST if and only if $q_A = 1$ for all $A \in N$. For $A \in N$, let $\mu$ be the unique infinite $\rightrightarrows_{\mathcal{G}}$-reduction $\vec{\mu}$ with $\mu_0 = \{^1A\}$. Note that $|\mu_n|_{\to_{\mathcal{G}}}$ is the probability that we reach a word from $T^*$ after $n$ steps.
Consider the graph on $N$ with an edge from $S_i$ to $S_j$ if and only if $(B_{\mathcal{G}})_{i,j} > 0$, and for $A \in N$ let $B_A$ be the restriction of $B_{\mathcal{G}}$ to the rows and columns of the non-terminals that are reachable from $A$ in this graph. Then $B_A$ is the moment matrix of the SCFG that results from $\mathcal{G}$ by removing all non-terminals that are not reachable from $A$. Since $\mathcal{G}$ is productive, the consistency check of [EY09] for the initial non-terminal $A$ does not reject in its second step, so its correctness [EY09] yields $q_A = 1$ if and only if $\rho(B_A) \leq 1$. If we order $N$ along a topological order of the strongly connected components (SCCs) of the graph, then $B_{\mathcal{G}}$ is a block triangular matrix with one diagonal block $B_C$ for every SCC $C$. Here, SCCs are maximal sets of non-terminals that reach each other via possibly empty paths. Hence, the eigenvalues of $B_{\mathcal{G}}$ are exactly the eigenvalues of the matrices $B_C$, and $\rho(B_{\mathcal{G}})$ is the maximum of all $\rho(B_C)$. Thus, $\mathcal{G}$ is AST if and only if $\rho(B_A) \leq 1$ for all $A \in N$, i.e., if and only if $\rho(B_{\mathcal{G}}) \leq 1$. *Proof* *QED*

By Theorem 6.1.18, for productive SCFGs it suffices to decide whether $\rho(B_{\mathcal{G}}) \leq 1$. For decision procedures, the probabilities have to be given in a finite representation. Therefore, we call an SCFG $\mathcal{G} = (N, T, P)$ *rational* if $P(A, \zeta) \in \mathbb{Q}$ for all $A \in N$ and

all $\zeta \in (N \cup T)^*$, and analogously, we call a PTRS *rational* if all probabilities in its rules are rational numbers. For a rational SCFG $\mathcal{G}$, all entries of $B_\mathcal{G}$ are rational as well. It remains to show that the condition $\rho(B_\mathcal{G}) \leq 1$ is decidable for every such moment matrix $B_\mathcal{G}$. Following [EGK13], we do not compute any eigenvalues for this, but only solve linear equation systems. As in the proof of Theorem 6.1.18, we order the non-terminals along a topological order of the SCCs of the graph with an edge from $S_i$ to $S_j$ if and only if $(B_\mathcal{G})_{i,j} > 0$. Then $B_\mathcal{G}$ is a block triangular matrix with one diagonal block $B_C$ for every SCC $C$, and $\rho(B_\mathcal{G})$ is the maximum of all $\rho(B_C)$. Each block $B_C$ is *irreducible*, where a matrix $A \in \mathbb{R}_{\geq 0}^{k \times k}$ is called irreducible if the graph on $\{1, \ldots, k\}$ with an edge from $i$ to $j$ if and only if $A_{i,j} > 0$ is strongly connected. For such blocks, one can compare the spectral radius with 1 by solving linear equation systems. In the following, $I_k$ denotes the $k \times k$ identity matrix and $\mathbf{1}_k = (1, \ldots, 1)^\top \in \mathbb{R}^k$.

Lemma 6.1.19 – Spectral Radius via Linear Equation Systems, [EGK13].

*Let $A \in \mathbb{R}_{\geq 0}^{k \times k}$ be irreducible.*

a) *If there is a vector $v \neq 0$ with $(I_k - A) \cdot v = 0$, then $\rho(A) = 1$ if $v_i > 0$ for all $1 \leq i \leq k$ or $v_i < 0$ for all $1 \leq i \leq k$, and $\rho(A) > 1$ otherwise.*

b) *Otherwise, $(I_k - A) \cdot x = \mathbf{1}_k$ has a unique solution $x$, and $\rho(A) < 1$ if $x_i \geq 1$ for all $1 \leq i \leq k$, and $\rho(A) > 1$ otherwise.*

*Proof.* See [EGK13]. *Proof QED*

These criteria rely on the Perron-Frobenius theory of non-negative matrices [LT85], which guarantees that the spectral radius of a non-negative matrix is itself a real eigenvalue, and that this eigenvalue has a strictly positive eigenvector if the matrix is irreducible. Hence, for every block $B_C$ of size $k \times k$, we first solve $(I_k - B_C) \cdot v = 0$. If there is a solution $v \neq 0$, then $\rho(B_C) = 1$ if the entries of $v$ are either all positive or all negative, and $\rho(B_C) > 1$ otherwise. If $v = 0$ is the only solution, then we solve $(I_k - B_C) \cdot x = \mathbf{1}_k$, and we have $\rho(B_C) < 1$ if $x_i \geq 1$ for all $i$, and $\rho(B_C) > 1$ otherwise. For a rational SCFG, all of this only requires Gaussian elimination over $\mathbb{Q}$, and the whole check of $\rho(B_\mathcal{G}) \leq 1$ can be performed in polynomial time [EGK13]. All in all, we can decide `AST` for a rational SCFG $\mathcal{G}$ by the following algorithm.

Algorithm 6.1.20 – Deciding `AST` for SCFGs.

***Input:*** *Rational SCFG $\mathcal{G}$*

1. *Check whether $\mathcal{G}$ is productive: No → Reject, Yes → go on.*
2. *Check whether $\rho(B_C) \leq 1$ for the block $B_C$ of every SCC $C$: Yes → Accept, No → Reject.*

Theorem 6.1.21 – `AST` is Decidable for SCFGs.
Almost-sure termination is decidable for rational SCFGs.

*Proof.* Consider Algorithm 6.1.20. It is correct, since an SCFG that is not productive is not `AST`, and a productive SCFG $\mathcal{G}$ is `AST` if and only if $\rho(B_\mathcal{G}) \leq 1$ by Theorem 6.1.18, i.e., if and only if $\rho(B_C) \leq 1$ for all SCCs $C$. Both steps are computable: productivity can be checked by simple graph algorithms, and for the rational SCFG $\mathcal{G}$, the condition $\rho(B_C) \leq 1$ can be decided by Gaussian elimination via Lemma 6.1.19 as described above. *Proof QED*

Example 6.1.22 (Decision Procedure for AST of SCFGs). Reconsider $\mathcal{G}^p_{\mathsf{KH}}$ from Example 6.1.6 and its moment matrix $B_{\mathcal{G}^p_{\mathsf{KH}}}$ from Example 6.1.15. Let us consider two different instantiations of the probabilities $p_1, \dots, p_6$ and analyze AST for the corresponding versions of $\mathcal{G}^p_{\mathsf{KH}}$. First, let $p_i = 1/2$ for all $1 \leq i \leq 6$. Then the grammar is productive, and we have

$$B_{\mathcal{G}^p_{\mathsf{KH}}} = \begin{pmatrix} 1/2 & 1 & 0 \\ 0 & 0 & 1/2 \\ 1/2 & 1/2 & 1/2 \end{pmatrix}$$

Its graph is strongly connected (with the cycle $S_1 \to S_2 \to S_3 \to S_1$), so there is a single SCC and $B_{\mathcal{G}^p_{\mathsf{KH}}}$ itself is irreducible. The matrix $I_3 - B_{\mathcal{G}^p_{\mathsf{KH}}}$ is non-singular, so $v = 0$ is the only solution of $(I_3 - B_{\mathcal{G}^p_{\mathsf{KH}}}) \cdot v = 0$, and the unique solution of $(I_3 - B_{\mathcal{G}^p_{\mathsf{KH}}}) \cdot x = \mathbf{1}_3$ is $x = (-10, -6, -14)^\top$. Hence, $\rho(B_{\mathcal{G}^p_{\mathsf{KH}}}) > 1$ by Lemma 6.1.19 (b) (indeed, $\rho(B_{\mathcal{G}^p_{\mathsf{KH}}}) \in [11/10, 12/10]$), and therefore, $\mathcal{G}^p_{\mathsf{KH}}$ is not AST in this case.

Second, let $p_2 = p_4 = p_5 = 0$ and $p_1 = p_3 = p_6 = 1$, where we drop the production rules with probability 0. Then the grammar is again productive, and we have

$$B_{\mathcal{G}^p_{\mathsf{KH}}} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 0 \end{pmatrix}$$

Now every SCC consists of a single non-terminal $S_i$ with the block $B_{\{S_i\}} = (0)$, where $(1-0) \cdot v = 0$ only has the solution $v = 0$ and $(1-0) \cdot x = 1$ has the solution $x = 1$. So $\rho(B_{\{S_i\}}) < 1$ for all SCCs by Lemma 6.1.19 (b), and therefore, $\mathcal{G}^p_{\mathsf{KH}}$ is AST in this case.

SECTION 6.2

## Transformation to Stochastic Grammars

Before we explain how to transform a given PTRS into an SCFG, we briefly recapitulate how termination can be decided given a non-probabilistic *right-ground* TRS. This illustrates limitations for the termination analysis already in the non-probabilistic setting. Afterwards, we explain why the same idea is not possible in the probabilistic setting.

A TRS $\mathcal{R}$ is right-ground if all of its right-hand sides are ground terms.

> Definition 6.2.1 – Right-Ground (TRS).
> *A TRS $\mathcal{R}$ is right-ground if we have $r \in \mathcal{T}^{\mathrm{gr}}$ for all $\ell \to r \in \mathcal{R}$.*

If right-hand sides contain no variables, then one cannot pass arbitrary information through recursive calls, but only data fixed within the rules themselves.

For example, every context-free grammar can be expressed by a right-ground TRS. Note that this is not the string rewrite system mentioned above, which contains variables in both the left-hand side and the right-hand side of each rule.

Example 6.2.2 (String Rewrite System for a CFG). Consider the non-probabilistic version of the Knudsen-Hein grammar $\mathcal{G}_{\mathsf{KH}}$ from Example 6.1.2. The corresponding string rewrite system $\mathcal{R}^{\mathsf{string}}_{\mathsf{KH}}$ has the form

$$\begin{array}{lcl@{\qquad}lcl@{\qquad}lcl} S(x) & \to & L(x) & L(x) & \to & s(x) & F(x) & \to & d(F(d(x))) \\ S(x) & \to & L(S(x)) & L(x) & \to & d(F(d(x))) & F(x) & \to & L(S(x)). \end{array}$$

The TRS $\mathcal{R}_{\mathsf{KH}}^{\mathsf{string}}$ from Example 6.2.2 is not right-ground, since all right-hand sides contain the variable $x$. Instead, we can combine the function symbols from the alphabet $N \cup T$ at parallel positions using a constructor $\mathrm{Com}_n$ for every possible arity $n \in \mathbb{N}$.

Example 6.2.3 (Right-Ground TRS for a CFG). Once again consider the non-probabilistic version of the Knudsen-Hein grammar $\mathcal{G}_{\mathsf{KH}}$ from Example 6.1.2. We can view the grammar as the right-ground (even ground) TRS $\mathcal{R}_{\mathsf{KH}}^{\mathrm{Com}}$ consisting of

$$\begin{array}{lcl@{\qquad}lcl@{\qquad}lcl} S & \to & \mathrm{Com}_1(L) & L & \to & \mathrm{Com}_1(s) & F & \to & \mathrm{Com}_3(d,F,d) \\ S & \to & \mathrm{Com}_2(L,S) & L & \to & \mathrm{Com}_3(d,F,d) & F & \to & \mathrm{Com}_2(L,S) \end{array}$$

where instead of nesting the symbols, we write them parallel to each other using the fresh constructors $\mathrm{Com}_1$, $\mathrm{Com}_2$, and $\mathrm{Com}_3$ of arity 1, 2, and 3, respectively.

To analyze termination of such right-ground TRSs it suffices to analyze sequences that start with a right-hand side of some rewrite rule. Moreover, for those terms we only have to check whether they can reach a term containing themselves, i.e., we have to check whether $r \to_{\mathcal{R}}^{+} C[r]$ holds for some $\ell \to r \in \mathcal{R}$ and some context $C$.

Lemma 6.2.4 – Termination Criterion for Right-Ground TRSs, [Der81].

*Let $\mathcal{R}$ be a finite right-ground TRS. $\mathcal{R}$ is terminating if and only if there exists no rule $\ell \to r \in \mathcal{R}$ such that $r \to_{\mathcal{R}}^{+} C[r]$ for some context $C$.*

*Proof.* See [Der81]. *Proof QED*

Lemma 6.2.4 tells us that we can simply evaluate all the right-hand sides until either all evaluations terminate (which then proves termination), or until we have $r \to_{\mathcal{R}}^{+} C[r]$ for some rule $\ell \to r \in \mathcal{R}$ (which then disproves termination). Thus, termination can be decided with the following algorithm.

Algorithm 6.2.5 – Deciding Termination for Right-Ground TRSs.

***Input:*** *Finite right-ground TRS $\mathcal{R}$*

1. *For $\mathcal{R} = \{\ell_1 \to r_1, \ldots, \ell_n \to r_n\}$ let $T_i = \{r_i\}$, $1 \leq i \leq n$.*
2. *For all $i$ set $T_i = \{t \mid s \in T_i, s \to_{\mathcal{R}} t\}$.*
3. *If $T_i = \varnothing$ for all $i$, then Accept.*
4. *If there is an $i$ and $t \in T_i$ such that $r_i$ is a subterm of $t$, then Reject.*
5. *Go back to Step 2.*

Example 6.2.6. Reconsider the ground TRS $\mathcal{R}_{\mathsf{KH}}^{\mathrm{Com}}$ from Example 6.2.3. For its set of right-hand sides $rhs(\mathcal{R}_{\mathsf{KH}}^{\mathrm{Com}}) = \{r \mid \ell \to r \in \mathcal{R}_{\mathsf{KH}}^{\mathrm{Com}}\}$, we have

$$rhs(\mathcal{R}_{\mathsf{KH}}^{\mathrm{Com}}) = \{\mathrm{Com}_1(L), \mathrm{Com}_1(s), \mathrm{Com}_3(d,F,d), \mathrm{Com}_2(L,S)\},$$

and $\mathrm{Com}_2(L, \underline{S}) \to_{\mathcal{R}_{\mathsf{KH}}^{\mathrm{Com}}} \mathrm{Com}_2(L, \mathrm{Com}_2(L,S))$, so $\mathcal{R}_{\mathsf{KH}}^{\mathrm{Com}}$ is not terminating.

Theorem 6.2.7 – Termination for Right-Ground TRSs, [Der81].
Termination is decidable for finite right-ground TRSs.

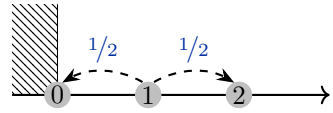
*Proof.* Consider Algorithm 6.2.5. If $\mathcal{R}$ terminates, then Step 4 never applies, since $r_i \to_{\mathcal{R}}^{+} C[r_i]$ would yield the infinite rewrite sequence $r_i \to_{\mathcal{R}}^{+} C[r_i] \to_{\mathcal{R}}^{+} C[C[r_i]] \to_{\mathcal{R}}^{+} \cdots$. Moreover, as $\mathcal{R}$ is finite, every term has only finitely many direct successors, so by KÖNIG's Lemma there is a bound on the lengths of all rewrite sequences starting with $r_1, \ldots, r_n$. Hence, the algorithm will terminate and yield *Accept*. If $\mathcal{R}$ does not terminate, then by Lemma 6.2.4 there must be a rule $\ell \to r \in \mathcal{R}$ such that $r \to_{\mathcal{R}}^{m} C[r]$ for some $m \in \mathbb{N}$. But then, the algorithm will terminate after at most $m$ iterations and return *Reject*. *Proof* *QED*

Note that this decision procedure is purely based on a *qualitative* property. We only have to show existence of some bad behavior, but we do not have to consider how likely it is to encounter such behavior if the rules had additional probabilities. In contrast, as we have seen for the grammars, it is important how many recursive calls are encountered in probabilistic evaluations.

Counterexample 6.2.8 (Symmetric Random Walk as a PTRS). Consider the PTRS $\mathcal{P}_{\mathsf{sym}}$ with the rule defined in Figure 6.5.

$$\mathsf{g} \to \{^{1/2}\mathsf{c}(\mathsf{g},\mathsf{g}), ^{1/2}\mathsf{a}\}$$

Figure 6.5: Definition of $\mathcal{P}_{\mathsf{sym}}$.




Count $\|t\| = |t|_{\mathsf{g}}$

Figure 6.6: $\mathcal{P}_{\mathsf{sym}}$ illustrated as an RWP.

The PTRS $\mathcal{P}_{\mathsf{sym}}$ models the symmetric random walk depicted in Figure 6.6. Even though the term g can reach the term $\mathsf{c}(\mathsf{g},\mathsf{g})$ containing g again, we have $\mathtt{AST}(\to_{\mathcal{P}_{\mathsf{sym}}})$.

Instead, we try to relate PTRSs to SCFGs, where AST is decidable and the corresponding decision algorithm takes both aspects into account, i.e., the existence and the number of recursive calls. As in the non-probabilistic setting, we restrict ourselves to right-ground PTRSs. For simplicity, we first consider the case where the PTRS is fully ground, and we lift the result to right-ground PTRSs in Section 6.3. A PTRS $\mathcal{P}$ is right-ground or left-ground if all the terms in all its right-hand sides or left-hand sides are ground terms, respectively. It is ground if all occurring terms are ground.

Definition 6.2.9 – Ground.
*Let $\mathcal{P}$ be a PTRS.*

a) *$\mathcal{P}$ is right-ground (RG) if we have $r \in \mathcal{T}^{\mathrm{gr}}$ for all $\ell \to \mu \in \mathcal{P}$ and all $r \in \mathrm{Supp}(\mu)$.*
b) *$\mathcal{P}$ is left-ground (LG) if we have $\ell \in \mathcal{T}^{\mathrm{gr}}$ for all $\ell \to \mu \in \mathcal{P}$.*
c) *$\mathcal{P}$ is ground (G) if it is both left-ground and right-ground.*

Example 6.2.10 (Ground PTRS for an SCFG). Consider the probabilistic version of the KNUDSEN-HEIN grammar from Example 6.1.6. We can view the grammar as the ground PTRS $\mathcal{P}_{\mathsf{KH}}$ consisting of the three rules only containing ground terms.

$$S \to \{^{p_1}\mathrm{Com}_1(L), ^{p_2}\mathrm{Com}_2(L,S)\}$$

$$L \to \{^{p_3}\mathrm{Com}_1(s), {}^{p_4}\mathrm{Com}_3(d, F, d)\}$$
$$F \to \{^{p_5}\mathrm{Com}_3(d, F, d), {}^{p_6}\mathrm{Com}_2(L, S)\}$$

We can see that PTRSs like $\mathcal{P}_{\mathsf{KH}}$ can also easily be transformed back into an SCFG. There are two properties of $\mathcal{P}_{\mathsf{KH}}$ (in addition to groundness) that guarantee that a transformation to SCFGs works: There are no nested defined symbols (tail-recursion), and the left-hand sides are non-overlapping such that there is no additional non-determinism except for the position in the term. The non-determinism due to the position of the rewrite step is resolved by only considering leftmost-innermost reductions, as required by the definition of SCFGs. Here, innermost rewriting is already guaranteed, since all occurring defined symbols have arity 0.

Definition 6.2.11 – Non-Overlapping.
*Let $\mathcal{P}$ be a PTRS. Two rules $\ell_1 \to \mu_1, \ell_2 \to \mu_2 \in \mathcal{P}$ with renamed variables such that $\mathcal{V}(\ell_1) \cap \mathcal{V}(\ell_2) = \varnothing$ are overlapping if there exists a non-variable position $\pi$ of $\ell_1$ such that $\ell_1|_\pi$ and $\ell_2$ are unifiable, i.e., there exists a substitution $\sigma$ such that $\ell_1|_\pi\sigma = \ell_2\sigma$. If $(\ell_1 \to \mu_1) = (\ell_2 \to \mu_2)$, then we require that $\pi \neq \varepsilon$. $\mathcal{P}$ is non-overlapping (*NO*) if it has no overlapping rules.*

Example 6.2.12 (Non-Overlapping). The PTRS $\mathcal{P}_{\mathsf{KH}}$ from Example 6.2.10 is non-overlapping, and the PTRS with $\mathsf{a} \to \{^1\mathsf{b}\}$ and $\mathsf{f}(\mathsf{a}) \to \{^1\mathsf{a}\}$ is overlapping.

Definition 6.2.13 – Tail-Recursive.
*A PTRS $\mathcal{P}$ is tail-recursive (*TR*) if there exists no rule $\ell \to \mu \in \mathcal{P}$, no term $r \in \mathrm{Supp}(\mu)$, and no positions $\pi, \tau \in \mathrm{pos}_{\mathcal{D}}(r)$ of defined function symbols in $r$ such that $\pi < \tau$. Here, $\mathrm{pos}_{\mathcal{D}}(r) = \{\pi \in \mathrm{pos}(r) \mid \mathrm{root}(r|_\pi) \in \mathcal{D}\}$.*

Example 6.2.14 (Tail-Recursive). The PTRS $\mathcal{P}_{\mathsf{KH}}$ from Example 6.2.10 is tail-recursive, and the PTRS containing the two rules $\mathsf{g}(x) \to \{^1x\}$ and $\mathsf{f}(x) \to \{^1\mathsf{g}(\mathsf{f}(\mathsf{a}))\}$ is not.

We want to decide AST for the subclass of PTRSs that are ground, non-overlapping, and tail-recursive, called GNT.

Definition 6.2.15 – GNT.
*The set of all finite PTRSs that are* G*,* NO*, and* TR *is denoted by* GNT.

Note that in general the left-hand sides do not have to be constants. Instead, they can be arbitrary ground terms. During the transformation to an SCFG we create an individual non-terminal for each whole left-hand side $\ell$. Then, for the right-hand sides we extract each occurrence of a left-hand side and create a string containing all occurrences from left to right.

Definition 6.2.16 – Transformation from GNT to SCFG.
*Let $\mathcal{P} \in$ GNT. The extracted grammar $\mathcal{T}_{gram}(\mathcal{P}) = \mathcal{G}$ is defined as the SCFG $\mathcal{G} = (N, T, P)$ with $N = lhs(\mathcal{P}) = \{\ell \mid \ell \to \mu \in \mathcal{P}\}$, $T = \varnothing$, and $P(\ell, w) = \sum_{p_{r \in \mu},\, \mathcal{T}_{word}(r) = w} p$ for every $\ell \in N$ and every word $w \in N^*$, where $\ell \to \mu \in \mathcal{P}$ is the unique rule with the left-hand side $\ell$ (it is unique, since $\mathcal{P}$ is non-overlapping) and the sum ranges over all entries of the multi-distribution $\mu$.*

*Here, $\mathcal{T}_{word}$ is recursively defined as*

$$\mathcal{T}_{word}(t) = \begin{cases} t, & \text{if } t \in N, \\ \mathcal{T}_{word}(t_1) \ \cdots \ \mathcal{T}_{word}(t_k), & \text{otherwise, if } t = f(t_1, \ldots, t_k), \\ \varepsilon, & \text{otherwise.} \end{cases}$$

Note that for a subterm $t = f(t_1, \ldots, t_k) \notin N$ of a right-hand side $r \in \mathrm{Supp}(\mu)$ with $f \in \mathcal{D}$ we have $\mathcal{T}_{\mathrm{word}}(t) = \varepsilon$ since $\mathcal{P}$ is tail-recursive, and thus, there cannot occur another defined symbol below the root of $t$.

Example 6.2.17 (SCFG Extraction). Consider the probabilistic version of the Knudsen-Hein grammar $\mathcal{G}^p_{\mathsf{KH}}$ from Example 6.1.6 and the ground PTRS $\mathcal{P}_{\mathsf{KH}}$ from Example 6.2.10. We have $\mathcal{P}_{\mathsf{KH}} \in \mathrm{GNT}$, i.e., the PTRS $\mathcal{P}_{\mathsf{KH}}$ is ground, non-overlapping, and tail-recursive, and moreover, $\mathcal{T}_{\mathrm{gram}}(\mathcal{P}_{\mathsf{KH}}) = (\{S, L, F\}, \varnothing, P)$ with $P$ defined as

$$\begin{array}{llll} S \xrightarrow{p_1} L & L \xrightarrow{p_3} \varepsilon & F \xrightarrow{p_5} F \\ S \xrightarrow{p_2} LS & L \xrightarrow{p_4} F & F \xrightarrow{p_6} LS. \end{array}$$

Note that in $\mathcal{T}_{\mathrm{gram}}(\mathcal{P}_{\mathsf{KH}})$ we have no terminal symbols in contrast to $\mathcal{G}^p_{\mathsf{KH}}$, as these symbols are irrelevant for termination analysis.

Example 6.2.18 (SCFG Extraction). Consider the PTRS $\mathcal{P}_{\mathsf{gnt}} \in \mathrm{GNT}$ with the three rules

$$\begin{aligned} \mathsf{f}(\mathsf{a}) &\to \{^{1/3}\mathsf{c}(\mathsf{g}, \mathsf{c}(\mathsf{f}(\mathsf{a}), \mathsf{f}(\mathsf{b}))), \ ^{2/3}\mathsf{c}(\mathsf{a}, \mathsf{b})\}, \\ \mathsf{f}(\mathsf{b}) &\to \{^{1}\mathsf{c}(\mathsf{g}, \mathsf{b})\}, \\ \mathsf{g} &\to \{^{1}\mathsf{f}(\mathsf{a})\}. \end{aligned}$$

Here, we have $\mathcal{T}_{\mathrm{gram}}(\mathcal{P}_{\mathsf{gnt}}) = (\{\mathsf{f}(\mathsf{a}), \mathsf{f}(\mathsf{b}), \mathsf{g}\}, \varnothing, P)$ where $P$ is defined as

$$\begin{array}{llll} \mathsf{f}(\mathsf{a}) \xrightarrow{1/3} \mathsf{g}\ \mathsf{f}(\mathsf{a})\ \mathsf{f}(\mathsf{b}), & & \mathsf{f}(\mathsf{b}) \xrightarrow{1} \mathsf{g}, \\ \mathsf{f}(\mathsf{a}) \xrightarrow{2/3} \varepsilon, & & \mathsf{g} \xrightarrow{1} \mathsf{f}(\mathsf{a}). \end{array}$$

The construction of the extracted SCFG preserves the almost-sure termination property as shown in the following lemma. Note that while reductions of grammars are always leftmost by definition, we have no fixed evaluation strategy for PTRSs. However, we can use our relations between the different evaluation strategies as shown in Chapter 4. Remember that $\mathtt{AST}(\xrightarrow{\mathtt{li}}_{\mathcal{P}})$ is equivalent to $\mathtt{AST}(\xrightarrow{\mathtt{f}}_{\mathcal{P}})$ for PTRSs that are linear and non-overlapping. In fact, PTRSs that are in GNT are ground and non-overlapping, and thus, linear and non-overlapping.

Lemma 6.2.19 – From Leftmost-Innermost to Full Rewriting for GNT.
*Let $\mathcal{P} \in \mathrm{GNT}$. Then $\mathtt{AST}(\xrightarrow{\mathtt{li}}_{\mathcal{P}}) \iff \mathtt{AST}(\xrightarrow{\mathtt{f}}_{\mathcal{P}})$.*

*Proof.* Every ground PTRS is linear, as there are no variables in its rules at all. Thus, we can apply Theorems 4.2.2 and 4.2.12. *Proof* | *QED*

Lemma 6.2.20 – SCFG Extraction.
*Let $\mathcal{P} \in \mathrm{GNT}$. Then $\mathtt{AST}(\mathcal{T}_{gram}(\mathcal{P})) \iff \mathtt{AST}(\mathcal{P})$.*

*Proof.* Let $\mathcal{G} = \mathcal{T}_{\text{gram}}(\mathcal{P}) = (N, \varnothing, P)$. We say that a term $s$ has *nested defined symbols* if there are positions $\pi, \tau \in \text{pos}_{\mathcal{D}}(s)$ with $\pi < \tau$. Let $s$ be a term without nested defined symbols. Since $\mathcal{P}$ is ground, the redexes of $s$ are exactly its subterms in $N = \mathit{lhs}(\mathcal{P})$, and since there are no defined symbols below their roots, they are at pairwise parallel positions and their proper subterms are normal forms. Hence, every rewrite step on $s$ is innermost, $\mathcal{T}_{\text{word}}(s)$ lists exactly the redexes of $s$ from left to right, the leftmost-innermost redex of $s$ is the first symbol of $\mathcal{T}_{\text{word}}(s)$, and $s \in \mathtt{NF}_{\mathcal{P}}$ if and only if $\mathcal{T}_{\text{word}}(s) = \varepsilon$. Moreover, let $s|_\pi = \ell$ be a redex of $s$ and let $r \in \text{Supp}(\mu)$ for the rule $\ell \to \mu \in \mathcal{P}$. Then $s[r]_\pi$ has no nested defined symbols either, since there is no defined symbol above $\pi$ in $s$, and $r$ has no nested defined symbols, as $\mathcal{P}$ is tail-recursive. For the same reason, no new redex arises above $\pi$. So if $s|_\pi$ is the leftmost redex of $s$, i.e., $\mathcal{T}_{\text{word}}(s) = \ell \cdot w$ for some $w \in N^*$, then we obtain

$$\mathcal{T}_{\text{word}}(s[r]_\pi) = \mathcal{T}_{\text{word}}(r) \cdot w. \tag{6.1}$$

▶ *Proof*($\Longrightarrow$): Let $\mathfrak{T}$ be a $\to_{\mathcal{P}}$-RT that converges with probability $|\mathfrak{T}| < 1$. By Theorem 5.3.2, we can assume that $\mathfrak{T}$ starts with a left-hand side $\ell \in N$ (as $\mathcal{P}$ is ground) and rewrites it at the root with the (by non-overlappingness unique) rule $\ell \to \{{}^{p_1}r_1, \ldots, {}^{p_k}r_k\} \in \mathcal{P}$. Let $\mathfrak{T}_j$ be the subtree below the child ${}^{p_j}r_j$ of the root, where all probabilities are divided by $p_j$. As $|\mathfrak{T}| = \sum_{j=1}^{k} p_j \cdot |\mathfrak{T}_j| < 1$, we have $|\mathfrak{T}_j| < 1$ for some $j$, so we can assume w.l.o.g. that $\mathfrak{T}$ starts with a right-hand side $t = r_j$. As $\mathcal{P}$ is tail-recursive, by induction no term in $\mathfrak{T}$ has nested defined symbols, so $\mathfrak{T}$ is an $\xrightarrow{\text{i}}_{\mathcal{P}}$-RT. By Lemmas 3.3.38 and 4.2.13, there is a fully evaluated $\xrightarrow{\text{li}}_{\mathcal{P}}$-RT starting with $t$ that converges with probability at most $|\mathfrak{T}| < 1$. As $\mathcal{P}$ is ground and non-overlapping, $\xrightarrow{\text{li}}_{\mathcal{P}}$ is deterministic, so this is the unique fully evaluated $\xrightarrow{\text{li}}_{\mathcal{P}}$-RT starting with $t$, and from now on, we denote it by $\mathfrak{T}$.
We obtain a $\to_{\mathcal{G}}$-RT $\mathfrak{T}'$ by replacing the label $s$ of every node of $\mathfrak{T}$ by $\mathcal{T}_{\text{word}}(s)$. If a node labeled with $s$ applies the rule $\ell \to \mu_\ell$ at the leftmost-innermost redex $s|_\pi = \ell$, then $\mathcal{T}_{\text{word}}(s) = \ell \cdot w$ for some $w \in N^*$, and by (6.1), its child for the outcome ${}^{p}r \in \mu_\ell$ is labeled with $\mathcal{T}_{\text{word}}(s[r]_\pi) = \mathcal{T}_{\text{word}}(r) \cdot w$, i.e., the corresponding node of $\mathfrak{T}'$ applies the production $\ell \xrightarrow{p} \mathcal{T}_{\text{word}}(r)$. If several outcomes yield the same word, then we merge the corresponding children and add up their probabilities, as in the definition of $P$ in Definition 6.2.16. This is possible, since $\xrightarrow{\text{li}}_{\mathcal{P}}$ is deterministic, so by (6.1) and induction on the depth, the $\mathcal{T}_{\text{word}}$-image of the subtree below a node labeled with $u$ only depends on $\mathcal{T}_{\text{word}}(u)$. Leaves with $s \in \mathtt{NF}_{\mathcal{P}}$ are labeled with $\mathcal{T}_{\text{word}}(s) = \varepsilon \in \mathtt{NF}_{\mathcal{G}}$. Hence, $\mathfrak{T}'$ is a $\to_{\mathcal{G}}$-RT starting with $\mathcal{T}_{\text{word}}(t)$ that converges with probability $|\mathfrak{T}'| = |\mathfrak{T}| < 1$.
▶ *Proof*($\Longleftarrow$): Assume that $\mathcal{G}$ is not $\mathtt{AST}$. Then there is a non-terminal $\ell \in N$ with $q_\ell < 1$ (see Section 6.1), so by Lemma 3.3.36 there is a fully evaluated $\to_{\mathcal{G}}$-RT $\mathfrak{T}'$ starting with $\ell$ that converges with probability $|\mathfrak{T}'| = q_\ell < 1$. We construct a $\to_{\mathcal{P}}$-RT $\mathfrak{T}$ starting with $\ell$ such that every node of $\mathfrak{T}'$ labeled with a word $w$ corresponds to a set of nodes of $\mathfrak{T}$ whose probabilities add up to the probability of the node in $\mathfrak{T}'$ and whose terms $s$ satisfy $\mathcal{T}_{\text{word}}(s) = w$. If a node of $\mathfrak{T}'$ labeled with $\ell' \cdot w'$ is rewritten, then each corresponding node of $\mathfrak{T}$ with term $s$ applies the unique rule $\ell' \to \mu_{\ell'}$ at the leftmost redex $s|_\pi = \ell'$, and each child $s[r]_\pi$ satisfies $\mathcal{T}_{\text{word}}(s[r]_\pi) = \mathcal{T}_{\text{word}}(r) \cdot w'$. For the root step, this holds as $s = \ell$, $\pi = \varepsilon$, and $w' = \varepsilon$, and all other terms of $\mathfrak{T}$ result from right-hand sides, so they have no nested defined symbols and we can use (6.1). If several outcomes ${}^{p_1}r_1, \ldots, {}^{p_m}r_m$ of $\mu_{\ell'}$ yield the same word $\zeta$, then the child of the $\mathfrak{T}'$-node labeled with $\zeta \cdot w'$ has the probability $P(\ell', \zeta) = p_1 + \ldots + p_m$, and we continue with a copy of the subtree of $\mathfrak{T}'$ below this child below each of the children $s[r_1]_\pi, \ldots, s[r_m]_\pi$ in $\mathfrak{T}$. Leaves labeled with $\varepsilon \in \mathtt{NF}_{\mathcal{G}}$ correspond to terms $s$

with $\mathcal{T}_{\text{word}}(s) = \varepsilon$, i.e., $s \in \mathtt{NF}_{\mathcal{P}}$. Hence, $\mathfrak{T}$ is a $\to_{\mathcal{P}}$-RT starting with $\ell$ that converges with probability $|\mathfrak{T}| = |\mathfrak{T}'| < 1$. *QED*

With Theorem 6.1.21 and Lemma 6.2.20, AST is decidable for rational PTRSs in GNT using the following algorithm.

Algorithm 6.2.21 – Deciding AST for GNT.
***Input:*** *Rational* $\mathcal{P} \in$ GNT

1. *Transform* $\mathcal{P}$ *into* $\mathcal{T}_{gram}(\mathcal{P}) = \mathcal{G}$.
2. *Check whether* $\mathcal{G}$ *is productive: No* $\to$ *Reject, Yes* $\to$ *go on.*
3. *Check whether* $\rho(B_{\mathcal{G}}) \leq 1$*: Yes* $\to$ *Accept, No* $\to$ *Reject.*

Theorem 6.2.22 – AST for GNT is Decidable.
It is decidable whether a given rational PTRS $\mathcal{P} \in$ GNT is AST.

*Proof.* The decision procedure is given by Algorithm 6.2.21. The SCFG $\mathcal{T}_{\text{gram}}(\mathcal{P})$ is computable and rational, as its probabilities are sums of probabilities of $\mathcal{P}$. Hence, the correctness and termination of the procedure follow from Theorem 6.1.21 and Lemma 6.2.20. *Proof* *QED*

Example 6.2.23 (Applying the Decision Procedure for GNT). Consider $\mathcal{P}_{\mathsf{gnt}}$ from Example 6.2.18 with the rules

$$\begin{aligned}\mathsf{f(a)} &\to \{^{1/3}\mathsf{c(g,c(f(a),f(b)))}, {}^{2/3}\mathsf{c(a,b)}\},\\ \mathsf{f(b)} &\to \{^{1}\mathsf{c(g,b)}\},\\ \mathsf{g} &\to \{^{1}\mathsf{f(a)}\}.\end{aligned}$$

Here, we have $\mathcal{G} = \mathcal{T}_{\text{gram}}(\mathcal{P}_{\mathsf{gnt}}) = (\{\mathsf{f(a)}, \mathsf{f(b)}, \mathsf{g}\}, \varnothing, P)$ where $P$ is defined as

$$\begin{aligned}\mathsf{f(a)} &\xrightarrow{1/3} \mathsf{g\ f(a)\ f(b)}, & \mathsf{f(b)} &\xrightarrow{1} \mathsf{g},\\ \mathsf{f(a)} &\xrightarrow{2/3} \varepsilon, & \mathsf{g} &\xrightarrow{1} \mathsf{f(a)}.\end{aligned}$$

The moment matrix of $\mathcal{G}$ is

$$B_{\mathcal{G}} = \begin{pmatrix} 1/3 & 1/3 & 1/3 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

Its graph is strongly connected (with the cycle $\mathsf{f(a)} \to \mathsf{f(b)} \to \mathsf{g} \to \mathsf{f(a)}$), so $B_{\mathcal{G}}$ itself is irreducible. All rows of $I_3 - B_{\mathcal{G}}$ sum up to 0, so $v = (1,1,1)^\top$ is a solution of $(I_3 - B_{\mathcal{G}}) \cdot v = 0$. As all entries of $v$ are positive, we obtain $\rho(B_{\mathcal{G}}) = 1$ by Lemma 6.1.19 (a). Since $\mathcal{G}$ is also productive, $\mathcal{G}$ is AST, and hence, $\mathcal{P}_{\mathsf{gnt}}$ is AST by Lemma 6.2.20.

SECTION 6.3

## Extending Decidability

In the previous section we have seen the first non-trivial subclass of PTRSs for which AST is decidable. In this section, we try to broaden the class such that AST remains

decidable. There are three separate properties defining the class GNT: groundness, non-overlappingness, and tail-recursion. Non-overlappingness is used to completely exclude any form of non-determinism that does not occur in SCFGs. Similarly, tail-recursion ensures that there is no dependency between two recursive calls. The best candidate to broaden the class is the property of being fully ground, as already in the non-probabilistic setting, right-groundness suffices for the decidability result. We show that we can broaden the class to only require right-groundness of the PTRSs such that the decidability result remains intact. We do this via a transformation from right-ground, non-overlapping, and tail-recursive PTRSs to ground, non-overlapping, and tail-recursive PTRSs. This transformation again preserves the almost-sure termination property, and thus, we can simply prepend this transformation to our decision algorithm.

The core idea of the construction is that not all (infinitely many) instantiations of the left-hand sides are of interest but only the (finitely many) instantiations of the left-hand sides that occur in the right-hand sides. Therefore, we can generate multiple copies of the rules, one for each occurring instantiation.

Let us first define the class of PTRSs we are interested in.

Definition 6.3.1 – RNT.
*The set of all finite PTRSs that are* RG, NO, *and* TR *is denoted by* RNT.

Note that membership in GNT and RNT is decidable, since groundness, right-groundness, and tail-recursion can be checked syntactically for the finitely many rules, and non-overlappingness only requires solving finitely many unification problems.

Example 6.3.2 (Resolving Right-Groundness to Groundness). Consider the following PTRS $\mathcal{P}_{\mathsf{rnt}} \in \text{RNT}$ with the rule

$$\mathsf{f}(x) \to \{{}^{1/2}\mathsf{g}, {}^{1/2}\mathsf{c}(\mathsf{f}(\mathsf{a}), \mathsf{f}(\mathsf{b}))\}.$$

The transformed PTRS $\mathcal{T}_{\text{GNT}}(\mathcal{P}_{\mathsf{rnt}}) \in \text{GNT}$ has one rule for each of the instantiations $\mathsf{f}(\mathsf{a})$ and $\mathsf{f}(\mathsf{b})$ of $\mathsf{f}(x)$ that occur in the right-hand side, i.e., it consists of the two rules

$$\mathsf{f}(\mathsf{a}) \to \{{}^{1/2}\mathsf{g}, {}^{1/2}\mathsf{c}(\mathsf{f}(\mathsf{a}), \mathsf{f}(\mathsf{b}))\},$$
$$\mathsf{f}(\mathsf{b}) \to \{{}^{1/2}\mathsf{g}, {}^{1/2}\mathsf{c}(\mathsf{f}(\mathsf{a}), \mathsf{f}(\mathsf{b}))\}.$$

Note that the two new left-hand sides $\mathsf{f}(\mathsf{a})$ and $\mathsf{f}(\mathsf{b})$ are not overlapping, and thus, the resulting PTRS is still non-overlapping.

Next, we formalize the transformation we illustrated in Example 6.3.2.

Definition 6.3.3 – Transformation from RNT to GNT.
*Let* $\mathcal{P} \in \text{RNT}$. *We define the set of instantiations of left-hand sides occurring in the right-hand sides of* $\mathcal{P}$ *by*

$$\text{iLhs}(\mathcal{P}) = \{\ell\sigma \mid \ell \to \mu, \ell' \to \mu' \in \mathcal{P}, t \in \text{Supp}(\mu'), \ell\sigma \trianglelefteq t \text{ for some substitution } \sigma\}.$$

*The transformation* $\mathcal{T}_{GNT}$ *maps* $\mathcal{P}$ *to the PTRS*

$$\mathcal{T}_{GNT}(\mathcal{P}) = \{\ell\sigma \to \mu \mid \ell \to \mu \in \mathcal{P}, \ell\sigma \in \text{iLhs}(\mathcal{P}) \text{ for some substitution } \sigma\}.$$

The resulting PTRS $\mathcal{T}_{\text{GNT}}(\mathcal{P})$ is still tail-recursive and right-ground, because the right-hand sides remain the same. Moreover, it remains non-overlapping, because all

instantiations of the left-hand sides that we generate are fully ground. To see this, note that two different instantiations of the same left-hand side cannot overlap at the root position, as they are different ground instantiations. Additionally, two instantiations of different left-hand sides cannot overlap at the root position either, because the original left-hand sides do not overlap. Finally, an instantiated left-hand side cannot overlap any instantiated left-hand side (including itself) at a non-root position, because this would imply that there exists a term in a right-hand side that contains nested defined function symbols, which contradicts the tail-recursion of $\mathcal{P}$.

Lemma 6.3.4 – From RNT to GNT.
*Let $\mathcal{P} \in$ RNT be a PTRS. Then, $\mathcal{T}_{GNT}(\mathcal{P}) \in$ GNT (i.e., $\mathcal{T}_{GNT}(\mathcal{P})$ is ground, non-overlapping, and tail-recursive), and we have $\mathtt{AST}(\mathcal{P}) \iff \mathtt{AST}(\mathcal{T}_{GNT}(\mathcal{P}))$.*

*Proof.* First, note that since $\mathcal{P}$ is finite, its right-hand sides contain only finitely many subterms, so iLhs($\mathcal{P}$) and hence $\mathcal{T}_{\mathrm{GNT}}(\mathcal{P})$ are finite as well. We prove all three statements separately. *Proof*

▶ *Proof*($\mathcal{T}_{\mathrm{GNT}}(\mathcal{P}) \in$ GNT): To see that $\mathcal{T}_{\mathrm{GNT}}(\mathcal{P}) \in$ GNT, note that $\mathcal{P} \in$ RNT and the right-hand sides of $\mathcal{T}_{\mathrm{GNT}}(\mathcal{P})$ are the same as for $\mathcal{P}$. Therefore, $\mathcal{T}_{\mathrm{GNT}}(\mathcal{P})$ remains tail-recursive and right-ground. Left-groundness of $\mathcal{T}_{\mathrm{GNT}}(\mathcal{P})$ follows by construction. The property that is non-trivial to show is non-overlappingness of $\mathcal{T}_{\mathrm{GNT}}(\mathcal{P})$. Let $\ell_1 \to \ldots, \ell_2 \to \ldots \in \mathcal{T}_{\mathrm{GNT}}(\mathcal{P})$ be overlapping. W.l.o.g., we have $\ell_1 = \ell_2$ or $\ell_1 \lhd \ell_2$, since all the resulting left-hand sides are ground terms.
First, consider the case $\ell_1 \lhd \ell_2$. Then $\ell_2 \in$ iLhs($\mathcal{P}$) is a subterm of a right-hand side of $\mathcal{P}$ that properly contains the instantiated left-hand side $\ell_1$, i.e., we have two nested defined function symbols in a right-hand side, which contradicts the tail-recursion of $\mathcal{P}$.
Second, consider the case $\ell_1 = \ell_2$. If both are instantiations of the same left-hand side $\ell' \in \mathit{lhs}(\mathcal{P})$, then they stem from the same rule of $\mathcal{T}_{\mathrm{GNT}}(\mathcal{P})$ (as $\mathcal{P}$ is non-overlapping, $\ell'$ is the left-hand side of only one rule of $\mathcal{P}$), and such trivial root overlaps are ignored in the definition of non-overlappingness. If they are instantiations of the left-hand sides of two different rules $\ell'_1, \ell'_2 \in \mathit{lhs}(\mathcal{P})$, i.e., $\ell'_1\sigma_1 = \ell_1 = \ell_2 = \ell'_2\sigma_2$ for some substitutions $\sigma_1$, $\sigma_2$, then $\ell'_1$ and $\ell'_2$ are unifiable after renaming their variables apart (as $\ell_1 = \ell_2$ is a common instance) and overlap at the root. This is a contradiction to non-overlappingness of $\mathcal{P}$.

▶ *Proof*($\neg\mathtt{AST}(\mathcal{P}) \implies \neg\mathtt{AST}(\mathcal{T}_{\mathrm{GNT}}(\mathcal{P}))$): Let $\mathfrak{T}$ be a $\to_{\mathcal{P}}$-RT starting with $t$ that converges with probability $|\mathfrak{T}| < 1$. W.l.o.g., we can assume that $t$ is a ground right-hand side, since by Theorem 5.3.2 the initial rewrite step can be assumed to be performed at the root, and such a root step yields only ground right-hand sides. As $|\mathfrak{T}| < 1$, the subtree below one of the successors of the root also converges with probability $< 1$ (after dividing its probabilities by the probability of this successor), and we can start with this subtree.
We construct a $\to_{\mathcal{T}_{\mathrm{GNT}}(\mathcal{P})}$-RT $\mathfrak{T}'$ with the same tree structure by replacing, at each node, the PTRS rule application with the instantiated version of the rule. More precisely, each node of $\mathfrak{T}$ labeled with a term $s$ becomes a node of $\mathfrak{T}'$ labeled with $s$ as well. If a node of $\mathfrak{T}$ labeled with $s$ applies rule $\ell \to \mu_\ell$ at position $\pi$ and substitution $\sigma$ with $s|_\pi = \ell\sigma$, then we have $\ell\sigma \in$ iLhs($\mathcal{P}$) because every redex arises as a subterm of a previously produced right-hand side (note that no term in $\mathfrak{T}$ contains nested defined function symbols, as right-hand sides are tail-recursive and rewriting cannot create new nestings of defined symbols, so no redex can span the boundary of an inserted

right-hand side). Thus, the corresponding node of $\mathfrak{T}'$ applies the rule $\ell\sigma \to \mu_\ell$.
Since all probabilities are preserved, $\mathfrak{T}'$ is a $\to_{\mathcal{T}_{\mathrm{GNT}}(\mathcal{P})}$-RT starting with $t$ that converges with probability $|\mathfrak{T}'| = |\mathfrak{T}| < 1$.

▶ *Proof*($\neg\mathtt{AST}(\mathcal{T}_{\mathrm{GNT}}(\mathcal{P})) \implies \neg\mathtt{AST}(\mathcal{P})$): Let $\mathfrak{T}'$ be a $\to_{\mathcal{T}_{\mathrm{GNT}}(\mathcal{P})}$-RT starting with $t$ that converges with probability $|\mathfrak{T}'| < 1$. W.l.o.g., we can again assume that $t$ is a ground term using Theorem 5.3.2.
We construct a $\to_{\mathcal{P}}$-RT $\mathfrak{T}$ with the same tree structure by replacing, at each node, the PTRS rule application with the original version of the rule. More precisely, each node of $\mathfrak{T}'$ labeled with a term $s$ becomes a node of $\mathfrak{T}$ labeled with $s$ as well. If a node of $\mathfrak{T}'$ labeled with $s$ applies rule $\ell\sigma \to \mu_\ell$ at position $\pi$ with $s|_\pi = \ell\sigma$, then the corresponding node of $\mathfrak{T}$ applies the rule $\ell \to \mu_\ell$ with substitution $\sigma$.
Since all probabilities are preserved, $\mathfrak{T}$ is a $\to_{\mathcal{P}}$-RT starting with $t$ that converges with probability $|\mathfrak{T}| = |\mathfrak{T}'| < 1$. *QED*

Theorem 6.3.5 – AST for RNT is Decidable.
It is decidable whether a given rational PTRS $\mathcal{P} \in \mathrm{RNT}$ is AST.

*Proof.* Use the computable transformation $\mathcal{T}_{\mathrm{GNT}}(\mathcal{P})$, which is rational since it has the same right-hand sides as $\mathcal{P}$, and then apply the decision procedure for AST on GNT, see Algorithm 6.2.21. This yields Algorithm 6.3.6 below. Correctness follows from Lemma 6.3.4 and Theorem 6.2.22. *Proof* *QED*

Algorithm 6.3.6 – Deciding AST for RNT.
***Input:*** *Rational* $\mathcal{P} \in \mathrm{RNT}$

1. *Transform* $\mathcal{P}$ *into* $\mathcal{T}_{GNT}(\mathcal{P}) = \mathcal{P}'$.
2. *Transform* $\mathcal{P}'$ *into* $\mathcal{T}_{gram}(\mathcal{P}') = \mathcal{G}$.
3. *Check whether* $\mathcal{G}$ *is productive: No* $\to$ *Reject, Yes* $\to$ *go on.*
4. *Check whether* $\rho(B_{\mathcal{G}}) \leq 1$*: Yes* $\to$ *Accept, No* $\to$ *Reject.*

We illustrate this decision procedure with a final, comprehensive example.

Example 6.3.7 (Applying the Decision Procedure for RNT). Consider the PTRS $\mathcal{P}_{\mathsf{rnt}} \in \mathrm{RNT}$ from Example 6.3.2 with the rule

$$\mathsf{f}(x) \to \{{}^{1/2}\mathsf{g}, {}^{1/2}\mathsf{c}(\mathsf{f}(\mathsf{a}), \mathsf{f}(\mathsf{b}))\}.$$

Here, we have $\mathrm{iLhs}(\mathcal{P}_{\mathsf{rnt}}) = \{\mathsf{f}(\mathsf{a}), \mathsf{f}(\mathsf{b})\}$, and thus, the transformed PTRS $\mathcal{T}_{\mathrm{GNT}}(\mathcal{P}_{\mathsf{rnt}}) \in \mathrm{GNT}$ has the two rules

$$\mathsf{f}(\mathsf{a}) \to \{{}^{1/2}\mathsf{g}, {}^{1/2}\mathsf{c}(\mathsf{f}(\mathsf{a}), \mathsf{f}(\mathsf{b}))\},$$
$$\mathsf{f}(\mathsf{b}) \to \{{}^{1/2}\mathsf{g}, {}^{1/2}\mathsf{c}(\mathsf{f}(\mathsf{a}), \mathsf{f}(\mathsf{b}))\}.$$

The corresponding SCFG is $\mathcal{T}_{\mathrm{gram}}(\mathcal{T}_{\mathrm{GNT}}(\mathcal{P}_{\mathsf{rnt}})) = \mathcal{G} = (\{\mathsf{f}(\mathsf{a}), \mathsf{f}(\mathsf{b})\}, \varnothing, P)$ where $P$ is defined as

$$\mathsf{f}(\mathsf{a}) \xrightarrow{1/2} \varepsilon, \qquad \mathsf{f}(\mathsf{b}) \xrightarrow{1/2} \varepsilon,$$
$$\mathsf{f}(\mathsf{a}) \xrightarrow{1/2} \mathsf{f}(\mathsf{a})\ \mathsf{f}(\mathsf{b}), \qquad \mathsf{f}(\mathsf{b}) \xrightarrow{1/2} \mathsf{f}(\mathsf{a})\ \mathsf{f}(\mathsf{b}).$$

The moment matrix of $\mathcal{G}$ is

$$B_{\mathcal{G}} = \begin{pmatrix} 1/2 & 1/2 \\ 1/2 & 1/2 \end{pmatrix}$$

which is irreducible, and $v = (1,1)^\top$ is a solution of $(I_2 - B_{\mathcal{G}}) \cdot v = 0$ with only positive entries. Hence, $\rho(B_{\mathcal{G}}) = 1$ by Lemma 6.1.19 (a). Since $\mathcal{G}$ is also productive, this implies that $\mathcal{P}_{\mathsf{rnt}}$ is `AST`.

SECTION 6.4

## Evaluation

As in Section 4.5, we implemented all techniques of this chapter in AProVE and applied them to all 199 PTRSs from the *Termination Problem Database* [TPDB]. All experiments were performed on a machine with 16 GB of RAM and a timeout of 30 seconds for each example. To check $\rho(B_{\mathcal{G}}) \leq 1$, our implementation uses the procedure from Section 6.1: it splits the moment matrix into the blocks of its SCCs and decides for every block $B_C$ whether $\rho(B_C) < 1$, $\rho(B_C) = 1$, or $\rho(B_C) > 1$ by exact Gaussian elimination over the rationals (Lemma 6.1.19). Here, AProVE represents all probabilities as fractions whose numerators and denominators are integers of arbitrary precision, so that no rounding errors can occur.

We first count how many of these PTRSs fall into our decidable subclasses. Recall that GNT requires the system to be ground, non-overlapping, and tail-recursive, while RNT relaxes groundness to right-groundness and hence subsumes GNT. We then apply the corresponding decision procedures, which for every in-class PTRS either prove or disprove `AST`, see Table 6.1.

| | GNT | RNT |
|---|---|---|
| PTRSs in the class | 23 (11.6%) | 30 (15.1%) |
| `AST` proven | 18 (9.0%) | 22 (11.1%) |
| `AST` disproven | 5 (2.5%) | 8 (4.0%) |

Table 6.1: Evaluation of the decision procedures for the classes GNT and RNT.

Percentages are relative to the 199 benchmarks. Since RNT subsumes GNT, the broader subclass covers 30 instead of 23 of the benchmarks and yields correspondingly more results (22 instead of 18 proofs and 8 instead of 5 disproofs). As both procedures are *decision* procedures, they decide `AST` for *every* PTRS in the respective class within the timeout.

We also show the contribution of the decision procedures within the full AProVE strategy, as we did in Chapter 4 for the different techniques: we run AProVE on full-rewriting `AST` with all techniques enabled, and then with the decision procedures disabled in turn, see Table 6.2. Percentages are again relative to the 199 benchmarks.

Disabling both decision procedures loses five `AST` proofs on this benchmark set ($90 \to 85$) and the two disproofs that only the decision procedures find within the timeout ($38 \to 36$). The five proofs belong to PTRSs that are not in RNT themselves, but where the ADP framework of Chapter 7 separates a sub-problem whose underlying PTRS is in RNT, so that the decision procedure is applied via the switching back processor (Section 7.4.6). With only the GNT decision procedure, one of the disproofs

| AProVE | proved | disproved |
|---|---|---|
| Full Rewriting `AST` | 90 (45.2%) | 38 (19.1%) |
| without GNT & RNT decision procedure | 85 (42.7%) | 36 (18.1%) |
| without RNT decision procedure (only GNT) | 85 (42.7%) | 37 (18.6%) |

Table 6.2: Evaluation of the decision procedures within the full AProVE strategy.

is recovered (37): the remaining example is right-ground but not ground, so it indeed requires the RNT extension, and the five proofs are not recovered either. Thus, within the full tool the decision procedures contribute both additional proofs and additional disproofs.

SECTION 6.5

## Related Work & Open Problems

The *hardness* of termination analysis of probabilistic programs has been investigated in [KKM19, MS24], showing that the analysis of `AST` is even more difficult than ordinary termination when considering the halting problem for given inputs. In this thesis, `AST` and `SAST` refer to universal termination, as we consider rewrite sequences starting with arbitrary terms. While for non-probabilistic programs, universal termination is "harder" than the halting problem for given inputs, these problems are equally hard for `AST` of probabilistic programs [KKM19]. However, for `SAST` we again have the same increase in "hardness" as in the non-probabilistic setting [KKM19].

In the setting of non-probabilistic term rewriting, subclasses of TRSs that admit a computable termination checking algorithm are most prominently right-ground TRSs and related syntactic restrictions that constrain how information can flow through recursive calls [HL78, Der81]. In contrast, if one restricts the considered class of ranking functions instead of the class of TRSs, then even the question whether termination can be proven via a linear polynomial interpretation remains undecidable [MMT24].

Another prominent line of work considers decidability of termination for imperative *linear single-loop programs*. Here, loops of the form $\mathsf{while}\ \Phi\ \mathsf{do}\ \vec{x} \leftarrow A\vec{x} + \vec{b}\ \mathsf{end}$ are considered. Termination of such linear loops has been shown to be decidable over the reals [Tiw04], over the rationals [Bra06], and over the integers (first for diagonalizable update matrices [OPW15] and then in general [HOW19]). Additionally, decidability of *triangular weakly non-linear loops*, whose updates may be polynomial, has been shown in [FG19, FHG20]. There also exist decidability results for runtime complexity, e.g., [HFG20]. Very recently, decidability of constant runtime for linear loops over the reals with real eigenvalues has been shown in [FGGL26].

In the probabilistic setting, there exist decidability results for simple randomized programs [MG25], for constant probability programs [GGH19], and for `PAST` of polynomial random walks [WK26]. Moreover, `AST` is decidable for several other classes of stochastic systems. For recursive Markov chains and probabilistic pushdown automata, `AST` and even the exact termination probabilities can be analyzed via the existential theory of the reals [EKM06, EY09, BEKK13, EY15]. These two models are equivalent w.r.t. termination probabilities [EY09]. Moreover, certificates for bounds on termination probabilities and expected runtimes of probabilistic pushdown automata

were studied in [WK23]. In addition, with POPACheck there exists a model checker specifically for probabilistic pushdown automata [PBC25]. For probabilistic higher-order recursion schemes, AST is undecidable for schemes of order 2 or higher, but it is decidable for schemes of order 1 [KDLG20], which are essentially as expressive as probabilistic pushdown automata and recursive Markov chains w.r.t. termination probabilities. Very recently, this decidability result was extended to affine probabilistic higher-order recursion schemes with bounded exponentials via the weighted relational model of linear logic [DLFP26]. Furthermore, for probabilistic vector addition systems with states, it is decidable whether the expected termination time is linear in the size of the initial configuration [BCK+19]. Similarly, for probabilistic counter programs, AST is decidable in dimension one, but undecidable for higher dimensions [NYC+25]. Note that one-dimensional probabilistic counter programs induce countably infinite Markov chains, i.e., a subclass of infinite MDPs. Hence, in contrast to the restriction to finite MDPs discussed at the beginning of this chapter, the decision procedure of [NYC+25] (which is based on a finitization of the underlying Markov chain) shows that AST can remain decidable on infinite MDPs if their infinite state space is sufficiently structured. All of these results are highly related to the results presented in this chapter, since we showed a correspondence between our subclass of PTRSs and stochastic context-free grammars, which correspond exactly to 1-exit recursive Markov chains and are thus subsumed by probabilistic pushdown automata and by affine probabilistic higher-order recursion schemes w.r.t. termination probabilities [EY09, DLFP26].

While we restricted our attention to SCFGs in this chapter, it is an interesting research question whether another stochastic system can be used to develop a decision procedure for another subclass of PTRSs.

> Open Problem 6.5.1 – Can We Use Different Stochastic Systems to Decide AST?.
>
> Find a subclass of PTRSs such that every PTRS $\mathcal{P}$ in this subclass can be transformed into a stochastic system $\mathcal{S}$ (different from SCFGs) that is equivalent w.r.t. termination probability and where AST is decidable for $\mathcal{S}$.

Another idea is to develop transformations in the PTRS setting, like the transformation from RNT to GNT to get rid of further restrictions.

> Open Problem 6.5.2 – From RNT to Broader Right-Ground Classes.
>
> Broaden decidability of AST beyond the current RNT restriction. In particular, can we remove the non-overlappingness or the tail-recursion requirement?

Finally, using results like [BCK+19] on the runtime analysis of probabilistic vector addition systems with states, or the results on expected runtimes of probabilistic programs with unbounded recursion [BKKV15, WK23], to get decision procedures for complexity analysis or even SAST is another interesting future work direction.

> Open Problem 6.5.3 – Decidability for SAST and Expected Complexity.
>
> Develop decision procedures for SAST and complexity analysis of PTRSs.

# 7 Proving Almost-Sure Termination

Using ranking functions based on interpretations alone to prove termination, as in Section 3.4.5, is often too weak to analyze real-world programs, for the following three main reasons:

1. First, one has to consider the whole system at once and has to find a single ranking function that orients every rewrite rule. Instead, one should take advantage of the modularity results presented in Section 5.4 to split a PTRS $\mathcal{P} = \mathcal{P}^{(1)} \uplus \mathcal{P}^{(2)}$ into smaller subsystems. Then one can use different ranking functions for different parts. However, these modularity results are currently not applicable to unions of systems that share rules, which is common in real-world programs. For example, the addition of natural numbers may be needed in many different parts of a program.

2. Second, ranking functions consider the whole right-hand side of a rewrite rule and ignore the fact that some parts of the right-hand side cannot be evaluated on their own. For example, constructor symbols cannot be evaluated on their own without another defined symbol above them. Instead, we would like to compare left-hand sides to more fine-grained versions of the right-hand side only containing the "important" part for termination.

3. Finally, to lift ranking functions from orienting all rewrite rules to orienting arbitrary rewrite steps, we have to require monotonicity, which is a hard restriction for proving termination.

Counterexample 7.0.1 (Polynomial Orders are too Weak). Consider the signature $\Sigma = \{0, \mathsf{s}, \mathsf{minus}, \mathsf{div}\}$, a variable set with $\{x, y\} \subseteq \mathcal{V}$, and the following PTRS $\mathcal{P}_{\mathsf{div}}$:

$$\mathsf{minus}(x, 0) \to \{{}^{1/2}\mathsf{minus}(x, 0), {}^{1/2}x\} \tag{7.1}$$

$$\mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)) \to \{{}^{1/2}\mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)), {}^{1/2}\mathsf{minus}(x, y)\} \tag{7.2}$$

$$\mathsf{div}(0, \mathsf{s}(y)) \to \{{}^{1/2}\mathsf{div}(0, \mathsf{s}(y)), {}^{1/2}0\} \tag{7.3}$$

$$\mathsf{div}(\mathsf{s}(x), \mathsf{s}(y)) \to \{{}^{1/2}\mathsf{div}(\mathsf{s}(x), \mathsf{s}(y)), {}^{1/2}\mathsf{s}(\mathsf{div}(\mathsf{minus}(x, y), \mathsf{s}(y)))\} \tag{7.4}$$

The PTRS $\mathcal{P}_{\mathsf{div}}$ models an algorithm for the integer division of two natural numbers, where we have a chance of $1/2$ to do the actual rewrite step and a chance of $1/2$ to do nothing (i.e., the term stays the same). We have $\mathtt{AST}(\to_{\mathcal{P}_{\mathsf{div}}})$, since the underlying TRS is terminating, and with every step, we have a constant probability ($1/2$) to make a step "towards termination" and in the other case, we do nothing. A direct

application of polynomial interpretations to prove AST is not possible. In order to satisfy the constraints of Corollary 3.2.64, we would need to find a natural, monotonic, and multilinear polynomial interpretation Pol, such that for every rule $\ell \to \mu \in \mathcal{P}_{\mathsf{div}}$ there is at least one term in the support of the right-hand side that has a strictly smaller value than the left-hand side. But this is not possible for (7.4):

$$\mathsf{div}(\mathsf{s}(x), \mathsf{s}(y)) \to \{{}^{1/2}\mathsf{div}(\mathsf{s}(x), \mathsf{s}(y)), {}^{1/2}\mathsf{s}(\mathsf{div}(\mathsf{minus}(x, y), \mathsf{s}(y)))\}$$

The first term of the right-hand side ($\mathsf{div}(\mathsf{s}(x), \mathsf{s}(y))$) is equal to the left-hand side, so it has the same polynomial value. The second term of the right-hand side is $\mathsf{s}(\mathsf{div}(\mathsf{minus}(x, y), \mathsf{s}(y)))$. Let us assume for a contradiction that we have

$$\mathrm{Pol}\big(\mathsf{div}(\mathsf{s}(x), \mathsf{s}(y))\big) > \mathrm{Pol}\big(\mathsf{s}\,(\mathsf{div}(\mathsf{minus}(x, y), \mathsf{s}(y)))\big).$$

As Pol is monotonic, we have $\mathrm{Pol}(\mathsf{s}(x)) \geq \mathrm{Pol}(x)$ for all instantiations of $x$, and thus

$$\mathrm{Pol}\big(\mathsf{div}(\mathsf{s}(x), \mathsf{s}(y))\big) > \mathrm{Pol}\big(\mathsf{s}\,(\mathsf{div}(\mathsf{minus}(x, y), \mathsf{s}(y)))\big) \geq \mathrm{Pol}\big(\mathsf{div}(\mathsf{minus}(x, y), \mathsf{s}(y))\big).$$

Now, this inequality needs to hold for all instantiations of the variables, so also if we instantiate $y$ with $\mathsf{s}(x)$. In this case, we get

$$\mathrm{Pol}\big(\mathsf{div}(\mathsf{s}(x), \mathsf{s}(\mathsf{s}(x)))\big) > \mathrm{Pol}\big(\mathsf{div}(\mathsf{minus}(x, \mathsf{s}(x)), \mathsf{s}(\mathsf{s}(x)))\big).$$

We have $\mathrm{Pol}(\mathsf{minus}(x, \mathsf{s}(x))) \geq \mathrm{Pol}(\mathsf{s}(x))$, and thus by the monotonicity of Pol we get

$$\mathrm{Pol}\big(\mathsf{div}(\mathsf{minus}(x, \mathsf{s}(x)), \mathsf{s}(\mathsf{s}(x)))\big) \geq \mathrm{Pol}\big(\mathsf{div}(\mathsf{s}(x), \mathsf{s}(\mathsf{s}(x)))\big).$$

All in all, we have

$$\mathrm{Pol}\big(\mathsf{div}(\mathsf{s}(x), \mathsf{s}(\mathsf{s}(x)))\big) > \mathrm{Pol}\big(\mathsf{div}(\mathsf{s}(x), \mathsf{s}(\mathsf{s}(x)))\big),$$

which is a contradiction.

We show in Example 7.5.6 that $\mathtt{AST}(\xrightarrow{\mathsf{i}}_{\mathcal{P}_{\mathsf{div}}})$ holds using our new ADP framework that we present in this chapter.

To overcome all of the restrictions mentioned above in the non-probabilistic setting, the *dependency pair framework* (DP framework) was developed, which is a modular *divide-and-conquer* framework that integrates processors to split large termination problems into simpler sub-problems. Moreover, this framework allows us to *combine* different termination techniques that can be used on sub-problems without having to consider the whole system at once.

The core principle behind dependency pairs is that a recursive function terminates if and only if its arguments strictly decrease with each recursive call. Rather than comparing entire right-hand sides of rewrite rules, dependency pairs focus on comparing the arguments of the left-hand side with the arguments of the recursive function calls in the right-hand side. For this comparison, we construct *dependency pairs* that capture the arguments of these recursive function calls, and in addition, the function that is being called. For example, to show termination of the Ackermann function $\mathsf{ack}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{ack}(x, \mathsf{ack}(\mathsf{s}(x), y))$, we have to compare the arguments of the initial function call $\langle \mathsf{s}(x), \mathsf{s}(y) \rangle$ with the recursive calls on the right: $\langle x, \mathsf{ack}(\mathsf{s}(x), y) \rangle$ and $\langle \mathsf{s}(x), y \rangle$. The dependency pairs that one has to consider are $\mathsf{ack}^\sharp(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{ack}^\sharp(x, \mathsf{ack}(\mathsf{s}(x), y))$ and $\mathsf{ack}^\sharp(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{ack}^\sharp(\mathsf{s}(x), y)$. Here,

the symbol $\mathsf{ack}^\sharp$ represents a call to the function represented by the defined symbol $\mathsf{ack}$, where we store the arguments additionally. We have to prove that the arguments of the $\mathsf{ack}$-function decrease in each recursive function application, which can be done via a lexicographic comparison of the arguments. Since the arguments themselves may be rewritten due to nested function calls, we analyze pairs $(\mathcal{D}, \mathcal{R})$, where $\mathcal{D}$ consists of dependency pairs requiring a termination proof, and where $\mathcal{R}$ consists of rewrite rules used for evaluating the arguments of the dependency pairs $\mathcal{D}$.

Within the dependency pair framework we use processors Proc that transform a complicated problem $(\mathcal{D}, \mathcal{R})$ into simpler sub-problems $(\mathcal{D}_1, \mathcal{R}_1), \ldots, (\mathcal{D}_k, \mathcal{R}_k)$ until all of them are proven to terminate. Two prominent examples of processors in the dependency pair framework are the *dependency graph processor* and the *reduction pair processor*. The *dependency graph processor* plays a similar role as the modularity results presented in Section 5.4 in the setting of dependency pairs. Due to the additional set of rewrite rules $\mathcal{R}$, sharing rewrite rules in different sub-parts of the program is now possible. The *reduction pair processor* is a generalization of ranking functions as presented in Theorem 3.2.62, where instead of monotonic algebras it suffices to use weakly monotonic ones. This holds due to the fact that we only have to prove a strict decrease for the dependency pairs and not for the ordinary rewrite rules, i.e., we only need a strict decrease for the function calls without any context around them.

**Algorithm 7:** pGCL-Alg.

```
x ← 0
while x = 0 do
    {
     x ← 0 ⊕_{1/2} x ← 1;
     y ← 2 · y;
    }□{
     x ← 0 ⊕_{1/3} x ← 1;
     y ← 3 · y;
    }
while y > 0 do
    y ← y − 1;
```

In this chapter we present an adaptation of the dependency pair framework for termination of TRSs in order to analyze almost-sure termination for a given PTRS $\mathcal{P}$ w.r.t. an arbitrary evaluation strategy or w.r.t. an innermost evaluation strategy. Instead of dependency pairs as explained above, we have to consider a more involved notion called *annotated dependency pairs*. While dependency pairs consider each of the function calls in the right-hand side of a rule individually, we have to consider them simultaneously in the probabilistic setting. The framework for innermost evaluation turns out to be drastically more powerful than the one for arbitrary evaluation strategies. Within this framework we consider processors based on the syntactic structure of the rules (e.g., the dependency graph processor), processors simply based on ranking functions (e.g., the reduction pair processor), and processors that transform dependency pairs into new ones that are easier to analyze. While many of the most powerful processors can be lifted in a non-trivial way into the probabilistic setting, there are hard limitations that make the analysis harder. We discuss the reason for all of these restrictions whenever we encounter them.

The following are two running examples throughout the first sections of this chapter.

Example 7.0.2. The following PTRS $\mathcal{P}_{\mathsf{alg}}$ corresponds to Algorithm 7, which repeats Algorithm 1 from Chapter 1. As usual, natural numbers are represented via the constructors $\mathsf{0}$ and $\mathsf{s}$, and the non-determinism is modeled by the non-deterministic choice between the overlapping rules (7.5) and (7.6). In this chapter, we prove $\mathtt{AST}(\xrightarrow{\mathrm{i}}_{\mathcal{P}_{\mathsf{alg}}})$ and $\mathtt{AST}(\to_{\mathcal{P}_{\mathsf{alg}}})$.

$$\mathsf{loop1}(y) \to \{^{1/2}\mathsf{loop1}(\mathsf{double}(y)),\ ^{1/2}\mathsf{loop2}(\mathsf{double}(y))\} \tag{7.5}$$

$$\mathsf{loop1}(y) \to \{{}^{1/3}\mathsf{loop1}(\mathsf{triple}(y)),\ {}^{2/3}\mathsf{loop2}(\mathsf{triple}(y))\} \tag{7.6}$$

$$\mathsf{loop2}(\mathsf{s}(y)) \to \{{}^{1}\mathsf{loop2}(y)\} \tag{7.7}$$

$$\mathsf{double}(\mathsf{s}(y)) \to \{{}^{1}\mathsf{s}(\mathsf{s}(\mathsf{double}(y)))\} \tag{7.8}$$

$$\mathsf{double}(0) \to \{{}^{1}0\} \tag{7.9}$$

$$\mathsf{triple}(\mathsf{s}(y)) \to \{{}^{1}\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{triple}(y))))\} \tag{7.10}$$

$$\mathsf{triple}(0) \to \{{}^{1}0\} \tag{7.11}$$

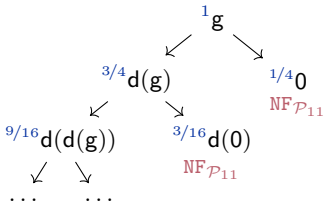
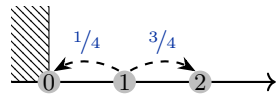



Figure 7.1: Infinite $\to_{\mathcal{P}_{11}}$-RT.




Count $\|t\| = |t|_{\mathsf{g}}$

Figure 7.2: $\mathcal{P}_{\mathsf{dup}}$ illustrated as an RWP.

Example 7.0.3. As another example, consider the PTRSs $\mathcal{P}_{11} = \{(7.12)\}$, $\mathcal{P}_{\mathsf{dup}} = \{(7.12), (7.14)\}$, and $\mathcal{P}_{12} = \{(7.12), (7.13), (7.15)\}$. Here, $\mathcal{P}_{\mathsf{dup}}$ is the PTRS that we already considered in Counterexample 4.2.1.

$$\mathsf{g} \to \{{}^{3/4}\mathsf{d}(\mathsf{g}),\ {}^{1/4}0\} \tag{7.12}$$

$$\mathsf{d}(x) \to \{{}^{1}0\} \tag{7.13}$$

$$\mathsf{d}(x) \to \{{}^{1}\mathsf{c}(x,x)\} \tag{7.14}$$

$$\mathsf{d}(\mathsf{d}(x)) \to \{{}^{1}\mathsf{c}(x,\mathsf{g})\} \tag{7.15}$$

$\mathcal{P}_{\mathsf{dup}}$ is not $\mathtt{AST}$. If we always apply (7.14) directly after (7.12) (in a non-innermost step), then this corresponds to the rule

$$\mathsf{g} \to \{{}^{3/4}\mathsf{c}(\mathsf{g},\mathsf{g}),\ {}^{1/4}0\}$$

which represents a positively drifting random walk on the number of $\mathsf{g}$'s in a term, see Figure 7.2. $\mathcal{P}_{12}$ is not $\mathtt{AST}$ either, because if we always apply (7.15) after two applications of (7.12), this corresponds to

$$\mathsf{g} \to \{{}^{9/16}\mathsf{c}(\mathsf{g},\mathsf{g}),\ {}^{3/16}0,\ {}^{1/4}0\}$$

which is also a positively drifting random walk towards non-termination.

However, in innermost evaluations, the $\mathsf{d}$-rule (7.14) can only duplicate normal forms, i.e., we cannot apply the rule (7.14) directly after (7.12), and hence $\mathtt{AST}(\xrightarrow{\mathsf{i}}_{\mathcal{P}_{\mathsf{dup}}})$ holds. We also have $\mathtt{AST}(\xrightarrow{\mathsf{i}}_{\mathcal{P}_{12}})$, as (7.15) is not applicable in innermost evaluations.

### Contributions

We present the following contributions in this chapter. The results marked with !★! are novel, while the remaining contributions can be found in the following peer-reviewed publications: ★ [KG23, KDG24, KG24, KG26].

- ★ Introduction of *annotated dependency pairs* (ADPs) as a novel extension of dependency pairs for PTRSs to handle the analysis of $\mathtt{AST}$ (Section 7.2).
- ★ Development of the *annotated dependency pair framework*, a modular divide-and-conquer framework for proving $\mathtt{AST}$ of PTRSs, using processors to decompose problems into simpler sub-problems (Section 7.3).

!★! The processor for *removing irrelevant ADPs* that replaces irrelevant ADPs by trivially terminating ones for `iAST` (Theorem 7.3.5).

★ The *dependency graph processor* that splits problems based on strongly connected components of the dependency graph (Section 7.4.1).

★ The *usable terms processor* that removes annotations from non-usable terms (Section 7.4.2).

★ The *usable rules processor* that deactivates non-usable rules (Section 7.4.3).

★ The *probability removal processor* that transforms probabilistic ADP problems into non-probabilistic DP problems when all rules are non-probabilistic (Section 7.4.4).

★ Discussion on the possibility of switching from full rewriting to innermost rewriting within the ADP framework (Section 7.4.5).

!★! The *switch back processor* that returns from an ADP problem to its underlying PTRS, e.g., to apply the decision procedure for the subclass RNT (Section 7.4.6).

★ The *reduction pair processor* that uses barycentric algebras to prove termination by comparing left-hand sides to the value of the annotated subterms in the corresponding right-hand sides (Section 7.5.1).

★ The *subterm criterion processor* that proves termination by comparing arguments using the subterm relation (Section 7.5.2).

★ The *rewriting processor* that applies rewrite rules to perform necessary rewrite steps (Section 7.6.1).

★ The *instantiation processor* that instantiates variables in ADPs to make them more concrete (Section 7.6.2).

★ The *forward instantiation processor* that works like the instantiation processor, but propagates instantiations forward instead of backward (Section 7.6.3).

★ The *rule overlap instantiation processor* that works like the instantiation processor, but considers the possible rewrite steps below annotations instead of the ADP steps at annotations (Section 7.6.4).

## Structure

We recapitulate ordinary dependency pairs for termination of TRSs in Section 7.1 and our novel adaptation for `AST` of PTRSs called annotated dependency pairs in Section 7.2. The general annotated dependency pair framework is then explained in Section 7.3. This framework is able to combine different termination techniques using *processors* that simplify ADP problems into simpler sub-problems. We first present processors based on *structural* properties in Section 7.4, then present processors based on *ranking functions* in Section 7.5, and finally present *transformational* processors

in Section 7.6. After presenting the details of all the different processors, we give a comprehensive table containing the most important information for each processor in Section 7.7. We end this chapter by performing an experimental evaluation in Section 7.8, and discussing related work and giving important open problems for future work in Section 7.9.

SECTION 7.1

## The Dependency Pair Framework

We first briefly recapitulate the idea of the dependency pair framework for TRSs.

The dependency pair framework (DP framework) is used to analyze (innermost) termination of TRSs, see, e.g., [AG00, GTS04, GTSF06, HM05, HM07] for further details. Remember the definition of defined symbols and constructors. With respect to some TRS $\mathcal{R}$, the signature $\Sigma = \mathcal{D} \uplus \mathcal{C}$ is decomposed into *defined symbols* $\mathcal{D}$ and *constructors* $\mathcal{C}$, such that $f \in \mathcal{D}$ if and only if $f = \mathrm{root}(\ell)$ for some $\ell \to r \in \mathcal{R}$. The defined symbols represent executable functions while the constructors represent data. For the remaining chapters, we assume that the signature $\Sigma$ and the decomposition $\Sigma = \mathcal{D} \uplus \mathcal{C}$ are fixed at the start of the analysis.

Example 7.1.1. Consider $\mathcal{R}_{\mathsf{ex}} = \{(7.16), (7.17)\}$. Here, $\mathcal{D} = \{\mathsf{f}, \mathsf{g}\}$ and $\mathcal{C} = \{\mathsf{s}, \mathsf{c}\}$.

$$\mathsf{f}(\mathsf{s}(x)) \to \mathsf{c}(\mathsf{f}(\mathsf{g}(x))) \qquad (7.16) \qquad\qquad \mathsf{g}(x) \to \mathsf{s}(x) \qquad (7.17)$$

For every $f \in \mathcal{D}$ we introduce a fresh *annotated symbol* $f^\sharp$ of the same arity. The symbols $f^\sharp$ are also called *tuple symbols* (or *marked symbols*) in the literature. The annotated symbols are used to compare the arguments of two subsequent "function calls". For example, if we want to prove termination of a TRS containing only the rule $\mathsf{plus}(\mathsf{s}(x), y) \to \mathsf{s}(\mathsf{plus}(x, y))$, then we need to show that the arguments of the defined function symbol $\mathsf{plus}$ "decrease" w.r.t. some well-founded order in every recursive call. To avoid handling tuples of arguments, we use the annotated symbol $\mathsf{plus}^\sharp$ to combine the arguments of $\mathsf{plus}$, i.e., we compare $\mathsf{plus}^\sharp(\mathsf{s}(x), y)$ with $\mathsf{plus}^\sharp(x, y)$. More precisely, for every rule $\ell \to r$ and every (not necessarily proper) subterm $t$ of $r$ with defined root symbol, one obtains a *dependency pair* (DP) $\ell^\sharp \to t^\sharp$ that represents two subsequent "function calls" in a rewrite sequence, where we have to compare the arguments.

Definition 7.1.2 – Dependency Pair.

*For every $f \in \mathcal{D}$ we introduce a fresh annotated symbol $f^\sharp$ of the same arity. Let $\mathcal{D}^\sharp$ denote the set of all annotated symbols, $\Sigma^\sharp = \mathcal{D}^\sharp \uplus \Sigma$, and $\mathcal{T}^\sharp = \mathcal{T}(\Sigma^\sharp, \mathcal{V})$. For any $t = f(t_1, \ldots, t_n) \in \mathcal{T}$ with $f \in \mathcal{D}$, let $t^\sharp = f^\sharp(t_1, \ldots, t_n)$.*
*For every rule $\ell \to r$ and every (not necessarily proper) subterm $t \trianglelefteq r$ with defined root symbol, one obtains a dependency pair (DP) $\ell^\sharp \to t^\sharp$. By $\mathcal{DP}(\mathcal{R})$ we denote the set of all dependency pairs of $\mathcal{R}$.*

Example 7.1.3. Continuing Example 7.1.1, we have $\mathcal{DP}(\mathcal{R}_{\mathsf{ex}}) = \{(7.18), (7.19)\}$.

$$\mathsf{f}^\sharp(\mathsf{s}(x)) \to \mathsf{f}^\sharp(\mathsf{g}(x)) \qquad (7.18) \qquad\qquad \mathsf{f}^\sharp(\mathsf{s}(x)) \to \mathsf{g}^\sharp(x) \qquad (7.19)$$

Instead of only considering the dependency pairs of a TRS, we have to consider the original rewrite rules as well, as these rules may be used to evaluate the arguments of dependency pairs. The goal is to prove the absence of infinite *chains* where a chain represents a sequence of subsequent "function calls" in evaluations. Again, here we can

use the original rewrite rules to evaluate the arguments of the recursive function calls.

Definition 7.1.4 – Chain.

*Let $\mathcal{D}$ be a set of DPs, and $\mathcal{R}$ be a TRS. A (possibly infinite) sequence $t^\sharp_0, t^\sharp_1, t^\sharp_2, \dots$ with $t^\sharp_j \xrightarrow{\varepsilon}_{\mathcal{D}} \circ \to^*_{\mathcal{R}} t^\sharp_{j+1}$ for all $j$ is a $(\mathcal{D},\mathcal{R})$-chain. Here, "$\xrightarrow{\varepsilon}_{\mathcal{D}}$" denotes rewrite steps with DPs from $\mathcal{D}$ at the root of the term.*

We refer to steps with $\xrightarrow{\varepsilon}_{\mathcal{D}}$ as **a**-*steps* (**a** for ***a**nnotation*, as we rewrite at the position of an annotated symbol), and we call steps with $\to_{\mathcal{R}}$ **n**-*steps* (**n** for ***n**o-annotation*, as we rewrite at the position of a symbol without annotation). The latter are used to evaluate the arguments of an annotated function symbol. So an infinite chain consists of an infinite number of **a**-steps with a finite number of **n**-steps in between.

Example 7.1.5. For $\mathcal{R}_{\mathsf{ex}}$ from Example 7.1.3, the sequence $\mathsf{f}^\sharp(\mathsf{s}(x)), \mathsf{f}^\sharp(\mathsf{s}(x)), \dots$ is an infinite $(\mathcal{DP}(\mathcal{R}_{\mathsf{ex}}),\mathcal{R}_{\mathsf{ex}})$-chain, as we have $\underline{\mathsf{f}}^\sharp(\mathsf{s}(x)) \xrightarrow{\varepsilon}_{\mathcal{DP}(\mathcal{R}_{\mathsf{ex}})} \mathsf{f}^\sharp(\underline{\mathsf{g}}(x)) \to^*_{\mathcal{R}_{\mathsf{ex}}} \mathsf{f}^\sharp(\mathsf{s}(x))$.

To analyze innermost termination, we have to make sure that chains are evaluated innermost as well.

Definition 7.1.6 – Innermost Chain.

*Let $\mathcal{D}$ be a set of DPs, and $\mathcal{R}$ be a TRS. A (possibly infinite) sequence $t^\sharp_0, t^\sharp_1, t^\sharp_2, \dots$ with $t^\sharp_j \xrightarrow{\mathtt{i}\,\varepsilon}_{\mathcal{D},\mathcal{R}} \circ \xrightarrow{\mathtt{i}}{}^*_{\mathcal{R}} t^\sharp_{j+1}$ for all $j$ is an innermost $(\mathcal{D},\mathcal{R})$-chain. Here, "$\xrightarrow{\mathtt{i}\,\varepsilon}_{\mathcal{D},\mathcal{R}}$" is the restriction of $\xrightarrow{\varepsilon}_{\mathcal{D}}$ to rewrite steps where the used redex is in $\mathtt{NF}_{\mathcal{R}}$.*

Example 7.1.7. The $(\mathcal{DP}(\mathcal{R}_{\mathsf{ex}}),\mathcal{R}_{\mathsf{ex}})$-chain from Example 7.1.5 is innermost as well.

The DP framework works on *DP problems* $(\mathcal{D},\mathcal{R})$ where $\mathcal{D}$ is a finite set of DPs and $\mathcal{R}$ is a finite TRS. Such a DP problem is (innermost) terminating if there exists no infinite (innermost) chain.

Definition 7.1.8 – DP Problem.

*A DP problem is a pair $(\mathcal{D},\mathcal{R})$ of a finite set of DPs $\mathcal{D}$ and a finite TRS $\mathcal{R}$.*
*A DP problem $(\mathcal{D},\mathcal{R})$ is called terminating (or $\mathtt{SN}(\mathcal{D},\mathcal{R})$) if there is no infinite $(\mathcal{D},\mathcal{R})$-chain. It is innermost terminating (or $\mathtt{iSN}(\mathcal{D},\mathcal{R})$) if there is no infinite innermost $(\mathcal{D},\mathcal{R})$-chain.*

The main result on DPs is the *chain criterion* which states that there is no infinite $\to_{\mathcal{R}}$-sequence iff there is no infinite $(\mathcal{DP}(\mathcal{R}),\mathcal{R})$-chain (i.e., $\mathtt{SN}(\mathcal{R}) \iff \mathtt{SN}(\mathcal{DP}(\mathcal{R}),\mathcal{R})$), and analogously for innermost rewriting.

Theorem 7.1.9 – Chain Criterion, [AG00].

Let $\mathcal{R}$ be a finite TRS. Then

$$\mathtt{SN}(\to_{\mathcal{R}}) \iff \mathtt{SN}(\mathcal{DP}(\mathcal{R}),\mathcal{R}) \quad \text{and} \quad \mathtt{SN}(\xrightarrow{\mathtt{i}}_{\mathcal{R}}) \iff \mathtt{iSN}(\mathcal{DP}(\mathcal{R}),\mathcal{R}).$$

*Proof.* See [AG00]. *Proof QED*

Example 7.1.10. The TRS $\mathcal{R}_{\mathsf{ex}}$ is not terminating as we have the infinite $\to_{\mathcal{R}_{\mathsf{ex}}}$-sequence

$$\underline{\mathsf{f}}(\mathsf{s}(x)) \to_{\mathcal{R}_{\mathsf{ex}}} \mathsf{c}(\mathsf{f}(\underline{\mathsf{g}}(x))) \to_{\mathcal{R}_{\mathsf{ex}}} \mathsf{c}(\underline{\mathsf{f}}(\mathsf{s}(x))) \to \cdots .$$

The corresponding infinite $(\mathcal{DP}(\mathcal{R}_{\mathsf{ex}}), \mathcal{R}_{\mathsf{ex}})$-chain is $\mathsf{f}^\sharp(\mathsf{s}(x))$, $\mathsf{f}^\sharp(\mathsf{s}(x))$, …, where

$$\underline{\mathsf{f}^\sharp}(\mathsf{s}(x)) \xrightarrow{\varepsilon}_{\mathcal{DP}(\mathcal{R}_{\mathsf{ex}})} \mathsf{f}^\sharp(\underline{\mathsf{g}}(x)) \to_{\mathcal{R}_{\mathsf{ex}}} \mathsf{f}^\sharp(\mathsf{s}(x)).$$

The DP framework is a *divide-and-conquer* approach, which applies *DP processors* Proc of the form $\mathrm{Proc}(\mathcal{D}, \mathcal{R}) = \{(\mathcal{D}_1, \mathcal{R}_1), \ldots, (\mathcal{D}_n, \mathcal{R}_n)\}$, where $\mathcal{D}, \mathcal{D}_1, \ldots, \mathcal{D}_n$ are (finite) sets of DPs and $\mathcal{R}, \mathcal{R}_1, \ldots, \mathcal{R}_n$ are TRSs, to transform DP problems into simpler sub-problems. A processor Proc is *sound* for termination (for innermost termination) if $(\mathcal{D}, \mathcal{R})$ is $\mathtt{SN}$ ($\mathtt{iSN}$) whenever $(\mathcal{D}_j, \mathcal{R}_j)$ is $\mathtt{SN}$ ($\mathtt{iSN}$) for all $1 \leq j \leq n$. It is *complete* for termination (for innermost termination) if $(\mathcal{D}_j, \mathcal{R}_j)$ is $\mathtt{SN}$ ($\mathtt{iSN}$) for all $1 \leq j \leq n$ whenever $(\mathcal{D}, \mathcal{R})$ is $\mathtt{SN}$ ($\mathtt{iSN}$).

So given a TRS $\mathcal{R}$, one starts with the initial DP problem $(\mathcal{DP}(\mathcal{R}), \mathcal{R})$ and applies sound (and preferably complete) DP processors repeatedly until all sub-problems are "solved", i.e., transformed into the empty set. This yields a modular framework for termination and innermost termination proofs, as different techniques can be used for different sub-problems $(\mathcal{D}_j, \mathcal{R}_j)$. To give an intuition for the framework, we recapitulate the three most important processors of the DP framework.

### 7.1.1 Dependency Graph Processor

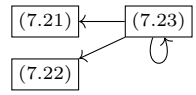
The (innermost) $(\mathcal{D}, \mathcal{R})$-*dependency graph* is a control flow graph that indicates which DPs can be used after each other in an (innermost) chain. Its set of nodes is $\mathcal{D}$ and there is an edge from $\ell_1^\sharp \to t_1^\sharp$ to $\ell_2^\sharp \to t_2^\sharp$ if there exist substitutions $\sigma_1, \sigma_2$ such that $t_1^\sharp\sigma_1 \to_{\mathcal{R}}^* \ell_2^\sharp\sigma_2$ ($t_1^\sharp\sigma_1 \xrightarrow{\mathrm{i}}_{\mathcal{R}}^* \ell_2^\sharp\sigma_2$ and $\ell_1^\sharp\sigma_1, \ell_2^\sharp\sigma_2 \in \mathtt{NF}_{\mathcal{R}}$ for the innermost dependency graph). Any infinite (innermost) $(\mathcal{D}, \mathcal{R})$-chain corresponds to an infinite path in the (innermost) dependency graph, and since the graph is finite, this infinite path must end in some strongly connected component (SCC).[1] Hence, it suffices to consider the SCCs of this graph independently.

> Theorem 7.1.11 – Dependency Graph Processor, [AG00, GTSF06].
>
> Let $(\mathcal{D}, \mathcal{R})$ be a DP Problem, and let $\mathcal{D}_1, \ldots, \mathcal{D}_n$ be the SCCs of the (innermost) $(\mathcal{D}, \mathcal{R})$-dependency graph. Then $\mathrm{Proc}_{\mathtt{DG}}(\mathcal{D}, \mathcal{R}) = \{(\mathcal{D}_1, \mathcal{R}), \ldots, (\mathcal{D}_n, \mathcal{R})\}$ is sound and complete for (innermost) termination.

Example 7.1.12. Consider $\mathcal{R}_{\mathsf{fgf}} = \{(7.20)\}$ with $\mathcal{DP}(\mathcal{R}_{\mathsf{fgf}}) = \{(7.21), (7.22), (7.23)\}$. The (innermost) $(\mathcal{DP}(\mathcal{R}_{\mathsf{fgf}}), \mathcal{R}_{\mathsf{fgf}})$-dependency graph is on the right.

$$\mathsf{f}(\mathsf{f}(\mathsf{g}(x))) \to \mathsf{f}(\mathsf{g}(\mathsf{f}(\mathsf{g}(\mathsf{f}(x))))) \quad (7.20)$$

$$\begin{aligned}
&\mathsf{f}^\sharp(\mathsf{f}(\mathsf{g}(x))) \to \mathsf{f}^\sharp(\mathsf{g}(\mathsf{f}(\mathsf{g}(\mathsf{f}(x))))) && (7.21)\\
&\mathsf{f}^\sharp(\mathsf{f}(\mathsf{g}(x))) \to \mathsf{f}^\sharp(\mathsf{g}(\mathsf{f}(x))) && (7.22)\\
&\mathsf{f}^\sharp(\mathsf{f}(\mathsf{g}(x))) \to \mathsf{f}^\sharp(x) && (7.23)
\end{aligned}$$




Here, $\mathrm{Proc}_{\mathtt{DG}}(\mathcal{DP}(\mathcal{R}_{\mathsf{fgf}}), \mathcal{R}_{\mathsf{fgf}})$ yields the DP problem $(\{(7.23)\}, \mathcal{R}_{\mathsf{fgf}})$.

While the exact (innermost) dependency graph is not computable in general, there exist several techniques to over-approximate it automatically, see, e.g., [AG00, GTSF06, HM05]. A basic approximation technique that we will also use for the transformational processors in Section 7.6 works as follows. To find out whether there is an edge from

[1] Here, a set $\mathcal{D}'$ of DPs is an *SCC* if it is a maximal cycle, i.e., it is a maximal set such that for any $\ell_1^\sharp \to t_1^\sharp$ and $\ell_2^\sharp \to t_2^\sharp$ in $\mathcal{D}'$ there is a non-empty path from $\ell_1^\sharp \to t_1^\sharp$ to $\ell_2^\sharp \to t_2^\sharp$ which only traverses nodes from $\mathcal{D}'$.

$\ell_1^\sharp \to t_1^\sharp$ to $\ell_2^\sharp \to t_2^\sharp$ in the innermost dependency graph we compute the skeleton $\mathrm{Cap}_\mathcal{R}(t_1^\sharp)$ of $t_1^\sharp$ that remains unchanged when reducing $t_1^\sharp\sigma_1$ to $\ell_2^\sharp\sigma_2$ by innermost steps for substitutions $\sigma_1$ that map all variables to normal forms and arbitrary substitutions $\sigma_2$. $\mathrm{Cap}_\mathcal{R}(t_1^\sharp)$ results from replacing all those subterms of $t_1^\sharp$ whose root is a defined symbol of $\mathcal{R}$ by different fresh variables. Here, multiple occurrences of the same subterm are also replaced by pairwise different variables. So if $\mathsf{f} \in \mathcal{D}$ and $\mathsf{c} \in \mathcal{C}$, then $\mathrm{Cap}_\mathcal{R}(\mathsf{c}(\mathsf{f}(x),\mathsf{f}(x))) = \mathsf{c}(x_1,x_2)$. Afterwards, we check whether the skeleton $\mathrm{Cap}_\mathcal{R}(t_1^\sharp)$ and $\ell_2^\sharp$ (after renaming the variables of $\ell_2^\sharp$ apart) are unifiable by a substitution $\sigma$ where $\ell_1^\sharp\sigma$ and $\ell_2^\sharp\sigma$ are in $\mathtt{ANF}_\mathcal{R}$ (and thus, also in $\mathtt{NF}_\mathcal{R}$). If they are, then we create an edge from $\ell_1^\sharp \to t_1^\sharp$ to $\ell_2^\sharp \to t_2^\sharp$ in the estimated innermost dependency graph. If they cannot be unified, then there is no innermost rewrite sequence from an instantiation of $t_1^\sharp$ to an instantiation of $\ell_2^\sharp$ such that both instantiated left-hand sides $\ell_1^\sharp, \ell_2^\sharp$ are in argument normal form. Thus, this leads to a sound over-approximation of the innermost dependency graph.

To check whether we can reduce $t_1^\sharp\sigma_1$ to $\ell_2^\sharp\sigma_2$ in non-innermost chains, we add another transformation Ren that replaces all variable occurrences (including multiple occurrences of the same variable) by fresh variables. The reason for this additional transformation is that for non-innermost chains, variables in $t_1^\sharp$ may be instantiated by terms that are not in normal form. Hence, they can be evaluated to different terms during the reduction of the arguments of $t_1^\sharp$. Thus, we check unifiability of $\mathrm{Ren}(\mathrm{Cap}_\mathcal{R}(t_1^\sharp))$ and $\ell_2^\sharp$ for the (non-innermost) dependency graph.

### 7.1.2 Usable Rules Processor

The next processor removes rules that cannot be used for right-hand sides of dependency pairs when their variables are instantiated with normal forms.[2]

Theorem 7.1.13 – Usable Rules Processor, [GTSF06].

Let $\mathcal{R}$ be a TRS. For every $f \in \Sigma^\sharp$, let $\mathrm{Rules}_\mathcal{R}(f) = \{\ell \to r \in \mathcal{R} \mid \mathrm{root}(\ell) = f\}$. Moreover, for every $t \in \mathcal{T}(\Sigma^\sharp, \mathcal{V})$, the *usable rules* $\mathcal{U}_\mathcal{R}(t)$ of $t$ w.r.t. $\mathcal{R}$ are recursively defined as follows, where $\mathcal{R}' = \mathcal{R} \setminus \mathrm{Rules}_\mathcal{R}(f)$:

$$\mathcal{U}_\mathcal{R}(t) = \varnothing, \qquad \text{if } t \in \mathcal{V} \text{ or } \mathcal{R} = \varnothing$$

$$\mathcal{U}_\mathcal{R}(f(t_1,\dots,t_n)) = \mathrm{Rules}_\mathcal{R}(f) \cup \bigcup_{j=1}^{n} \mathcal{U}_{\mathcal{R}'}(t_j) \cup \bigcup_{\ell\to r \in \mathrm{Rules}_\mathcal{R}(f)} \mathcal{U}_{\mathcal{R}'}(r), \quad \text{otherwise,}$$

The *usable rules* for the DP problem $(\mathcal{D},\mathcal{R})$ are $\mathcal{U}(\mathcal{D},\mathcal{R}) = \bigcup_{\ell^\sharp\to t^\sharp\in\mathcal{D}} \mathcal{U}_\mathcal{R}(t^\sharp)$. Then $\mathrm{Proc}_{\mathtt{UR}}(\mathcal{D},\mathcal{R}) = \{(\mathcal{D},\mathcal{U}(\mathcal{D},\mathcal{R}))\}$ is sound but not complete for innermost termination.

Example 7.1.14. We continue the analysis from Example 7.1.12 for innermost rewriting. The application $\mathrm{Proc}_{\mathtt{UR}}(\{(7.23)\},\mathcal{R}_{\mathsf{fgf}})$ yields the problem $(\{(7.23)\},\varnothing)$, i.e., it removes all rules, because the right-hand side of (7.23) does not contain the defined symbol $\mathsf{f}$.

[2] See [GTS04] for a complete version of this processor. It extends DP problems by an additional set which stores the left-hand sides of all rules (including the non-usable ones) to determine whether a rewrite step is innermost. We omit this refinement here for readability.

### 7.1.3 Reduction Pair Processor

Finally, the *reduction pair processor* allows us to use term orders to remove certain DPs. As in Theorem 3.2.62 we only consider term orders based on $\Sigma$-interpretations. Remember that $(\mathfrak{I}, \succ)$ is *monotonic* if $\mathfrak{I}(t_1) \succ \mathfrak{I}(t_2)$ implies $\mathfrak{I}(C[t_1]) \succ \mathfrak{I}(C[t_2])$ for every context $C$. Additionally, $(\mathfrak{I}, \succ)$ is *weakly monotonic* if $\mathfrak{I}(t_1) \succcurlyeq \mathfrak{I}(t_2)$ implies $\mathfrak{I}(C[t_1]) \succcurlyeq \mathfrak{I}(C[t_2])$ for every context $C$. The reduction pair processor searches for a weakly monotonic, ordered $\Sigma^\sharp$-algebra $(\mathfrak{I}, \succ)$ such that all rules and DPs are ordered by $\succcurlyeq$, and it removes those DPs that are ordered by $\succ$.

Theorem 7.1.15 – Reduction Pair Processor, [GTSF06].

Let $(\mathcal{D}, \mathcal{R})$ be a DP Problem. Let $(\mathfrak{I}, \succ)$ be a weakly monotonic, ordered $\Sigma^\sharp$-algebra, and let $\mathcal{D} = \mathcal{D}_{\succcurlyeq} \uplus \mathcal{D}_{\succ}$ with $\mathcal{D}_{\succ} \neq \varnothing$ such that

$$\begin{array}{lll} (1) & \forall \ell^\sharp \to r^\sharp \in \mathcal{D}. & \mathfrak{I}(\ell^\sharp) \succcurlyeq \mathfrak{I}(r^\sharp), \\ (2) & \forall \ell^\sharp \to r^\sharp \in \mathcal{D}_{\succ}. & \mathfrak{I}(\ell^\sharp) \succ \mathfrak{I}(r^\sharp), \\ (3) & \forall \ell \to r \in \mathcal{R}. & \mathfrak{I}(\ell) \succcurlyeq \mathfrak{I}(r). \end{array}$$

Then $\mathrm{Proc}_{\mathtt{RP}}(\mathcal{D}, \mathcal{R}) = \{(\mathcal{D}_{\succcurlyeq}, \mathcal{R})\}$ is sound and complete for both termination and innermost termination.

The reduction pair processor allows us to use weakly monotonic polynomial interpretations that do not have to depend on all of their arguments, and SMT solvers can be used to search for an appropriate interpretation automatically.

Example 7.1.16. For $(\{(7.23)\}, \varnothing)$, one can use the reduction pair processor with the polynomial interpretation $\mathsf{f}_{\mathrm{Pol}}(x) = x + 1$ and $\mathsf{f}^\sharp_{\mathrm{Pol}}(x) = \mathsf{g}_{\mathrm{Pol}}(x) = x$. Then, $\mathrm{Proc}_{\mathtt{RP}}\big(\{(7.23)\}, \varnothing\big) = \{\big(\varnothing, \varnothing\big)\}$. As $\mathrm{Proc}_{\mathtt{DG}}(\varnothing, \varnothing) = \varnothing$ and all processors used are sound, this means that there is no infinite innermost chain for the initial DP problem $(\mathcal{DP}(\mathcal{R}_{\mathsf{fgf}}), \mathcal{R}_{\mathsf{fgf}})$ and thus, $\mathcal{R}_{\mathsf{fgf}}$ is innermost terminating. This implies that $\mathcal{R}_{\mathsf{fgf}}$ is terminating, since $\mathcal{R}_{\mathsf{fgf}}$ is non-overlapping, hence, innermost termination implies termination [Gra95].

SECTION 7.2

## Annotated Dependency Pairs

Next, we present an adaptation of the framework for AST that uses *annotated dependency pairs*, which are an extension of ordinary dependency pairs. We first define annotated dependency pairs formally. Then we explain how to perform innermost rewriting with ADPs (Section 7.2.1) and extend this to an arbitrary evaluation strategy (Section 7.2.2). Finally, we adapt the notion of chains to the probabilistic setting (Section 7.2.3).

Ordinary DPs do not suffice in the probabilistic setting. Similar to the dependency tuples for complexity analysis of TRSs [NEG13], we cannot consider each defined symbol in a right-hand side individually, but we have to consider all of them simultaneously.

Counterexample 7.2.1. A natural idea to define dependency pairs for a probabilistic rule $\ell \to \{{}^{p_1}r_1, \ldots, {}^{p_k}r_k\} \in \mathcal{P}$ would be (7.24) or (7.25):

$$\{\ell^\sharp \to \{{}^{p_1}r_1, \ldots, {}^{p_j}t_j^\sharp, \ldots, {}^{p_k}r_k\} \mid \exists j \in \{1, \ldots, k\}.\ t_j \trianglelefteq r_j \wedge \mathrm{root}(t_j) \in \mathcal{D}\} \tag{7.24}$$

$$\{\ell^\sharp \to \{{}^{p_1}t_1^\sharp, \ldots, {}^{p_k}t_k^\sharp\} \mid \forall j \in \{1, \ldots, k\}.\ t_j \trianglelefteq r_j \wedge \mathrm{root}(t_j) \in \mathcal{D}\} \tag{7.25}$$

For (7.25), if some $r_j$ does not contain any defined symbol, then instead of $t_j^\sharp$ we use

a fresh constructor $\mathsf{bot}$ that does not occur in $\mathcal{P}$. So in both (7.24) and (7.25), we replace $r_j$ by a single term $t_j^\sharp$ in the right-hand side, as in the non-probabilistic DP framework. A counterexample to such an approach is the PTRS $\mathcal{P}_{\mathsf{drift}}$ containing only the rule $\mathsf{g} \to \{^{3/4}\mathsf{c}(\mathsf{g},\mathsf{g}),\ ^{1/4}\mathsf{0}\}$, which is a positively drifting random walk. When using (7.24) or (7.25), we would just get the dependency pair $\mathsf{g}^\sharp \to \{^{3/4}\mathsf{g}^\sharp,\ ^{1/4}t\}$, where $t$ is either $\mathsf{0}$ or $\mathsf{bot}$. However, with this dependency pair we lose the information about the second call of the defined symbol $\mathsf{g}$, which is the reason for non-termination of the random walk. In fact, the PTRS $\mathcal{P}_{11} = \{(7.12)\} = \{\mathsf{g} \to \{^{3/4}\mathsf{d}(\mathsf{g}),\ ^{1/4}\mathsf{0}\}\}$ would yield the same DP $\mathsf{g}^\sharp \to \{^{3/4}\mathsf{g}^\sharp,\ ^{1/4}t\}$, although $\mathcal{P}_{11}$ is $\mathtt{AST}$, whereas $\mathcal{P}_{\mathsf{drift}}$ is not. Thus, using such a direct adaptation of dependency pairs to the probabilistic setting is unsound.

The counterexample $\mathcal{P}_{\mathsf{drift}}$ shows that having only one annotation in the right-hand side of each dependency pair is not expressive enough, but instead we need a rule where all defined symbols in the right-hand side are annotated, called an *annotated dependency pair*. As we showed in [KVG24], ADPs are not only useful to analyze probabilistic rewriting, but a suitable variant of ADPs can also be used to extend the DP framework to (non-probabilistic) *relative* rewriting.

We start with some basic definitions regarding the positions in a term, in order to obtain all positions of defined symbols and to replace certain symbols with their annotated or non-annotated version.[3]

Definition 7.2.2 – Annotations.
*For $t \in \mathcal{T}^\sharp$ and $\mathcal{X} \subseteq \Sigma^\sharp \cup \mathcal{V}$, let $\mathrm{pos}_{\mathcal{X}}(t)$ be all positions of $t$ with symbols or variables from $\mathcal{X}$. For a set of positions $\Phi \subseteq \mathrm{pos}_{\mathcal{D}\cup\mathcal{D}^\sharp}(t)$, let $\sharp_\Phi(t)$ be the variant of $t$ where the symbols at positions from $\Phi$ in $t$ are annotated, and all other annotations are removed. Thus, $\mathrm{pos}_{\mathcal{D}^\sharp}(\sharp_\Phi(t)) = \Phi$, and $\sharp_\varnothing(t)$ removes all annotations from $t$, where we often write $\flat(t)$ instead of $\sharp_\varnothing(t)$. For $t \in \mathcal{T}$, we write $\sharp_{\mathcal{D}}(t)$ instead of $\sharp_{\mathrm{pos}_{\mathcal{D}}(t)}(t)$, i.e., all defined symbols of $t$ are annotated. Moreover, let $\flat_\pi^\uparrow(t)$ result from removing all annotations from $t$ that are strictly above the position $\pi$.*

Example 7.2.3. For $\mathcal{P}_{\mathsf{dup}}$ with the rules $\mathsf{g} \to \{^{3/4}\mathsf{d}(\mathsf{g}),\ ^{1/4}\mathsf{0}\}$ and $\mathsf{d}(x) \to \{^{1}\mathsf{c}(x,x)\}$ we have $\sharp_{\{1\}}(\mathsf{d}(\mathsf{g})) = \sharp_{\{1\}}(\mathsf{d}^\sharp(\mathsf{g}^\sharp)) = \mathsf{d}(\mathsf{g}^\sharp)$, $\flat(\mathsf{d}^\sharp(\mathsf{g}^\sharp)) = \mathsf{d}(\mathsf{g})$, and $\flat_1^\uparrow(\mathsf{d}^\sharp(\mathsf{g}^\sharp)) = \mathsf{d}(\mathsf{g}^\sharp)$.

To transform the rules of a PTRS into ADPs, initially we annotate all defined symbols occurring in right-hand sides, as we need to consider all these defined symbols simultaneously. The left-hand side of the resulting ADP is the left-hand side of the original rule.

The original DP framework for TRSs initially starts with the DP problem $(\mathcal{D},\mathcal{R})$, where $\mathcal{D}$ is the set of all DPs and $\mathcal{R}$ contains the original rewrite rules. Instead of additionally considering the original rewrite rules in our ADP framework, every ADP has a flag $m \in \{\mathsf{t},\mathsf{f}\}$ to indicate whether this ADP may also be used as an ordinary probabilistic rewrite rule without annotations to rewrite at a position below an annotated symbol without removing that annotation. This flag will be modified and used by the processors of the ADP framework, similar to the ordinary rewrite rules $\mathcal{R}$ in the DP problem $(\mathcal{D},\mathcal{R})$. So an ADP can be used as a collection of all DPs of a rule, but it can also be used as the original rule itself. We call this the *duality* of annotated dependency pairs.

[3] Instead of using $\natural$ as an inverse operator to $\sharp$ (like in musical notation), we use $\flat$ as in the musical isomorphism from differential geometry.

Remark 7.2.4

Instead of the flag $m$, one could of course also regard pairs $(\mathcal{A}, \mathcal{P})$ where $\mathcal{A}$ is a set of ADPs and $\mathcal{P}$ contains the same ADPs as $\mathcal{A}$ but without any annotations. We did not choose this representation since it "duplicates" information (i.e., every non-annotated original rule can be directly obtained from the corresponding ADP in $\mathcal{A}$ by removing all annotations and thus, it is superfluous to store both the ADP and its corresponding rule). This is in contrast to ordinary DPs where the original rule cannot be reconstructed from the DPs. Moreover, a representation with pairs $(\mathcal{A}, \mathcal{P})$ has the disadvantage that a processor like the usable rules processor would have to delete rules from $\mathcal{P}$ which affects the innermost evaluation strategy and would result in a processor that is not complete.

Definition 7.2.5 – Annotated Dependency Pair, ADP Problem.

*An annotated dependency pair (ADP) has the form* $\ell \xhookrightarrow{m} \{{}^{p_1}r_1, \dots, {}^{p_k}r_k\}$, *where* $\ell \in \mathcal{T}$ *with* $\ell \notin \mathcal{V}$, $m \in \{\mathsf{t}, \mathsf{f}\}$, $\sum_{1 \le j \le k} p_j = 1$, *and for all* $1 \le j \le k$ *we have* $0 < p_j$ *and* $r_j \in \mathcal{T}^\sharp$ *with* $\mathcal{V}(r_j) \subseteq \mathcal{V}(\ell)$. *An ADP problem is a finite set of ADPs* $\mathcal{A}$.
*For a rule* $\ell \to \mu = \{{}^{p_1}r_1, \dots, {}^{p_k}r_k\}$, *its canonical annotated dependency pair is* $\mathcal{DP}(\ell \to \mu) = \ell \xhookrightarrow{\mathsf{t}} \{{}^{p_1}\sharp_{\mathrm{pos}_{\mathcal{D}}(r_1)}(r_1), \dots, {}^{p_k}\sharp_{\mathrm{pos}_{\mathcal{D}}(r_k)}(r_k)\}$. *The canonical ADPs of a PTRS* $\mathcal{P}$ *are* $\mathcal{DP}(\mathcal{P}) = \{\mathcal{DP}(\ell \to \mu) \mid \ell \to \mu \in \mathcal{P}\}$.

For an ADP problem $\mathcal{A}$, let $\mathtt{NF}_{\mathcal{A}}$ be the set of all terms $t \in \mathcal{T}^\sharp$ such that no subterm of $\flat(t)$ is an instance of a left-hand side of an ADP from $\mathcal{A}$ (with flag $\mathsf{t}$ or $\mathsf{f}$), and let $\mathtt{ANF}_{\mathcal{A}}$ be the set of all terms $t \in \mathcal{T}^\sharp$ whose proper subterms are all in $\mathtt{NF}_{\mathcal{A}}$ (the *argument normal forms* w.r.t. $\mathcal{A}$). Both sets only depend on the set of left-hand sides of the ADPs in $\mathcal{A}$, and they do not depend on annotations, i.e., $t \in \mathtt{NF}_{\mathcal{A}}$ iff $\flat(t) \in \mathtt{NF}_{\mathcal{A}}$, and $t \in \mathtt{ANF}_{\mathcal{A}}$ iff $\flat(t) \in \mathtt{ANF}_{\mathcal{A}}$. Analogously to Definition 3.2.29, for a PTRS $\mathcal{P}$ let $\mathtt{ANF}_{\mathcal{P}}$ be the set of all terms $t \in \mathcal{T}$ whose proper subterms are all in $\mathtt{NF}_{\mathcal{P}}$.

In the following, we fix an ADP problem $\mathcal{A}$. Note that in contrast to ordinary dependency pairs as in Section 7.1, due to the duality of ADPs, the left-hand sides of ADPs are not annotated and the matching used to perform rewriting with ADPs in Definitions 7.2.8 and 7.2.11 will ignore the annotations as well.

Example 7.2.6. For $\mathcal{P}_{\mathsf{dup}} = \{(7.12), (7.14)\}$ and $\mathcal{P}_{12} = \{(7.12), (7.13), (7.15)\}$ from Example 7.0.3, we obtain $\mathcal{DP}(\mathcal{P}_{\mathsf{dup}}) = \{(\sharp 7.12), (\sharp 7.14)\}$ and $\mathcal{DP}(\mathcal{P}_{12}) = \{(\sharp 7.12), (\sharp 7.13), (\sharp 7.15)\}$ with

$$\mathsf{g} \xhookrightarrow{\mathsf{t}} \{{}^{3/4}\mathsf{d}^\sharp(\mathsf{g}^\sharp), {}^{1/4}\mathsf{0}\} \quad (\sharp 7.12) \qquad \mathsf{d}(x) \xhookrightarrow{\mathsf{t}} \{{}^{1}\mathsf{c}(x, x)\} \quad (\sharp 7.14)$$

$$\mathsf{d}(x) \xhookrightarrow{\mathsf{t}} \{{}^{1}\mathsf{0}\} \quad (\sharp 7.13) \qquad \mathsf{d}(\mathsf{d}(x)) \xhookrightarrow{\mathsf{t}} \{{}^{1}\mathsf{c}(x, \mathsf{g}^\sharp)\} \quad (\sharp 7.15)$$

Similar to the DPs in the non-probabilistic setting, the annotated subterms of a right-hand side $r_j$ correspond to those subsequent "function calls" where we need to prove that the arguments are decreasing to conclude $\mathtt{AST}$. For example, for the ADP $\mathsf{g} \xhookrightarrow{\mathsf{t}} \{{}^{3/4}\mathsf{d}^\sharp(\mathsf{g}^\sharp), {}^{1/4}\mathsf{0}\}$ $(\sharp 7.12)$, we have to compare the annotated left-hand side $\mathsf{g}^\sharp$ to both $\mathsf{d}^\sharp(\mathsf{g})$ and $\mathsf{g}^\sharp$ from the first term in the right-hand side. When adapting the reduction pair processor to the probabilistic setting in Section 7.5.1, we will show how to compare a single term on the left-hand side with several annotated subterms on the right-hand side.

Example 7.2.7. For $\mathcal{P}_{\mathsf{alg}}$ from Example 7.0.2, the canonical ADPs are

$$\mathsf{loop1}(y) \hookrightarrow^{\mathsf{t}} \{{}^{1/2}\mathsf{loop1}^\sharp(\mathsf{double}^\sharp(y)),\ {}^{1/2}\mathsf{loop2}^\sharp(\mathsf{double}^\sharp(y))\} \quad (\sharp 7.5)$$

$$\mathsf{loop1}(y) \hookrightarrow^{\mathsf{t}} \{{}^{1/3}\mathsf{loop1}^\sharp(\mathsf{triple}^\sharp(y)),\ {}^{2/3}\mathsf{loop2}^\sharp(\mathsf{triple}^\sharp(y))\} \quad (\sharp 7.6)$$

$$\mathsf{loop2}(\mathsf{s}(y)) \hookrightarrow^{\mathsf{t}} \{{}^{1}\mathsf{loop2}^\sharp(y)\} \quad (\sharp 7.7)$$

$$\mathsf{double}(\mathsf{s}(y)) \hookrightarrow^{\mathsf{t}} \{{}^{1}\mathsf{s}(\mathsf{s}(\mathsf{double}^\sharp(y)))\} \quad (\sharp 7.8)$$

$$\mathsf{double}(\mathsf{0}) \hookrightarrow^{\mathsf{t}} \{{}^{1}\mathsf{0}\} \quad (\sharp 7.9)$$

$$\mathsf{triple}(\mathsf{s}(y)) \hookrightarrow^{\mathsf{t}} \{{}^{1}\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{triple}^\sharp(y))))\} \quad (\sharp 7.10)$$

$$\mathsf{triple}(\mathsf{0}) \hookrightarrow^{\mathsf{t}} \{{}^{1}\mathsf{0}\} \quad (\sharp 7.11)$$

When defining the rewrite relation for ADPs, the essential part is to determine how to handle annotations if we rewrite above or below them. We start by considering only innermost evaluation for simplicity. Afterwards, we consider rewriting with an arbitrary evaluation strategy as well.

### 7.2.1 Innermost Rewriting with ADPs

As in the classical non-probabilistic DP framework, our goal is to track reduction sequences with **a**-steps where the root symbol of the redex is annotated, and between two **a**-steps there can be several **n**-steps. Our rewrite relation for ADPs refines **a**-steps into (**at**)- and (**af**)-steps in order to distinguish between rewriting at an **a**nnotation by an ADP with flag **t**rue or **f**alse. Similarly, we also refine **n**-steps into (**nt**)- and (**nf**)-steps depending on the flag of the ADP that is used to rewrite at a **n**on-annotated position. Still, the (**at**)- and (**af**)-steps are the crucial ones, and when extending chains to the probabilistic setting, we prohibit infinite rewrite sequences with only finitely many **a**-steps. Similar to the non-probabilistic setting, this is needed for the reduction pair processor to guarantee termination if we orient the **a**-steps strictly with a well-founded order, while only orienting the **n**-steps weakly.

Rewriting with an ADP problem $\mathcal{A}$ is like ordinary probabilistic term rewriting while considering and modifying annotations that indicate where a non-AST evaluation may arise. If we rewrite a term $s$ at an innermost position $\pi$ with an ADP $\ell \to \{{}^{p_1}r_1, \dots, {}^{p_k}r_k\}$, then the following properties should hold:

(a) Due to the duality of ADPs, the matching of left-hand sides should ignore the annotations. So for example, we have to be able to rewrite both $\mathsf{d}(\mathsf{d}(x))$ and $\mathsf{d}^\sharp(\mathsf{d}^\sharp(x))$ with an ADP $\mathsf{d}(\mathsf{d}(x)) \hookrightarrow^{\mathsf{t}} \{{}^{1}\mathsf{c}(x, \mathsf{g}^\sharp)\}$ at the root.

(b) Annotations below the position $\pi$ should be removed, as terms below the redex have to be normal forms due to the innermost evaluation strategy.

(c) Annotations at positions orthogonal to $\pi$ should be kept.

(d) Annotations at positions strictly above $\pi$ should only be kept if the ADP has the flag $m = \mathsf{t}$.

(e) Annotations in the right-hand sides $r_1, \dots, r_k$ should only be kept if the redex was annotated at the root, i.e., if $\pi \in \mathrm{pos}_{\mathcal{D}^\sharp}(s)$.

This leads to the following probabilistic rewrite relation.

Definition 7.2.8 – Innermost Rewriting with ADPs.

*Let $\mathcal{A}$ be an ADP problem. A term $s \in \mathcal{T}^\sharp$ rewrites innermost with $\mathcal{A}$ to $\mu = \{{}^{p_1}t_1, \dots, {}^{p_k}t_k\}$ (denoted $s \hookrightarrow^{\mathsf{i}}_{\mathcal{A}} \mu$) if there are an $\ell \hookrightarrow^{m} \{{}^{p_1}r_1, \dots, {}^{p_k}r_k\} \in \mathcal{A}$,*

*a substitution $\sigma$, and a $\pi \in \mathrm{pos}_{\mathcal{D}\cup\mathcal{D}^\sharp}(s)$ such that $\flat(s|_\pi) = \ell\sigma \in \mathtt{ANF}_{\mathcal{A}}$, and for all $1 \le j \le k$ the term $t_j$ is defined as follows, depending on the flag $m$ and on whether $\pi \in \mathrm{pos}_{\mathcal{D}^\sharp}(s)$ holds:*

| | $\pi \in \mathrm{pos}_{\mathcal{D}^\sharp}(s)$ | $\pi \notin \mathrm{pos}_{\mathcal{D}^\sharp}(s)$ |
|---|---|---|
| $m = \mathsf{t}$ | $t_j = s[r_j\sigma]_\pi$ **(at)** | $t_j = s[\flat(r_j)\sigma]_\pi$ **(nt)** |
| $m = \mathsf{f}$ | $t_j = \flat_\pi^\uparrow(s[r_j\sigma]_\pi)$ **(af)** | $t_j = \flat_\pi^\uparrow(s[\flat(r_j)\sigma]_\pi)$ **(nf)** |

*We may indicate explicitly which case is used by writing, e.g., $s \hookrightarrow^{\mathsf{i},\mathbf{at}}_{\mathcal{A}} \mu$.*

The above definition indeed corresponds to the properties (a)-(e):

(a) When computing the matching substitution $\sigma$, we only consider the terms $\ell$ and $\flat(s|_\pi)$ without annotations.

(b) Since the substitution $\sigma$ is obtained by matching $\ell$ against $\flat(s|_\pi)$, it does not contain any annotations. Hence, the term $r_j\sigma$ or $\flat(r_j)\sigma$ that replaces $s|_\pi$ contains no annotations of $s|_\pi$ below the root.

(c) All annotations on positions orthogonal to $\pi$ are not affected when computing $t_j$.

(d) If $m = \mathsf{f}$, then $\flat_\pi^\uparrow$ removes all annotations at positions strictly above $\pi$.

(e) If $\pi \notin \mathrm{pos}_{\mathcal{D}^\sharp}(s)$, then we use $\flat(r_j)$ to remove all annotations from $r_j$.

Let us consider an example for each individual case of $\hookrightarrow^{\mathsf{i}}_{\mathcal{A}}$. For this, we fix an ADP problem $\widetilde{\mathcal{A}}$ containing the three ADPs $\mathsf{f}(x) \hookrightarrow^{\mathsf{t}} \{^1\mathsf{f}^\sharp(x)\}$, $\mathsf{g}(\mathsf{a}) \hookrightarrow^{\mathsf{t}} \{^1\mathsf{g}^\sharp(\mathsf{b})\}$, and $\mathsf{g}(\mathsf{a}) \hookrightarrow^{\mathsf{f}} \{^1\mathsf{g}^\sharp(\mathsf{b})\}$.

**Steps With Annotation and Flag True.** A step of the form **at** (**a**nnotation and **t**rue) is performed at the position of an annotation, i.e., this can potentially lead to a non-AST evaluation. Hence, all annotations from the right-hand side $r_j$ of the used ADP are kept during the rewrite step. The rewrite step $\underline{\mathsf{f}^\sharp}(\mathsf{g}^\sharp(\mathsf{b})) \hookrightarrow^{\mathsf{i},\mathbf{at}}_{\widetilde{\mathcal{A}}} \{^1\mathsf{f}^\sharp(\mathsf{g}(\mathsf{b}))\}$ is done using the ADP $\mathsf{f}(x) \hookrightarrow^{\mathsf{t}} \{^1\mathsf{f}^\sharp(x)\}$ at the root position $\pi = \varepsilon$ with the substitution $\sigma(x) = \mathsf{g}(\mathsf{b})$. We get $\flat(s|_\pi) = \flat(\mathsf{f}^\sharp(\mathsf{g}^\sharp(\mathsf{b}))|_\varepsilon) = \mathsf{f}(\mathsf{g}(\mathsf{b})) = \ell\sigma$ and $t_1 = s[r_1\sigma]_\pi = \mathsf{f}^\sharp(\mathsf{g}(\mathsf{b}))$. Note that annotations below the redex are removed, i.e., we obtain $\mathsf{g}(\mathsf{b})$ instead of $\mathsf{g}^\sharp(\mathsf{b})$. This is not a problem, as in innermost evaluations terms below the redex are in normal form anyway.

**Steps With Annotation and Flag False.** A step of the form **af** (**a**nnotation and **f**alse) is similar but due to the flag $m = \mathsf{f}$, this ADP cannot be used below an annotation in a non-AST evaluation. Hence, we remove all annotations above the used redex. Using the ADP $\mathsf{g}(\mathsf{a}) \hookrightarrow^{\mathsf{f}} \{^1\mathsf{g}^\sharp(\mathsf{b})\}$ on the term $\mathsf{f}^\sharp(\mathsf{g}^\sharp(\mathsf{a}))$ at position 1 yields $\mathsf{f}^\sharp(\underline{\mathsf{g}^\sharp}(\mathsf{a})) \hookrightarrow^{\mathsf{i},\mathbf{af}}_{\widetilde{\mathcal{A}}} \{^1\mathsf{f}(\mathsf{g}^\sharp(\mathsf{b}))\}$, i.e., we remove the annotation of $\mathsf{f}^\sharp$ at the root.

**Steps Without Annotation and Flag True.** A step of the form **nt** (**n**o annotation and **t**rue) is performed at the position of a subterm without annotation. Hence, the subterm cannot lead to a non-AST evaluation, but this rewrite step may be needed for an annotation at a position above. As an example, one could rewrite the non-annotated subterm $\mathsf{g}(\mathsf{a})$ in $\mathsf{f}^\sharp(\underline{\mathsf{g}}(\mathsf{a}))$ $\hookrightarrow^{\mathsf{i},\mathbf{nt}}_{\widetilde{\mathcal{A}}} \{^1\mathsf{f}^\sharp(\mathsf{g}(\mathsf{b}))\}$ using the ADP $\mathsf{g}(\mathsf{a}) \hookrightarrow^{\mathsf{t}} \{^1\mathsf{g}^\sharp(\mathsf{b})\}$.

**Steps Without Annotation and Flag False.** Finally, a step of the form **nf** (**n**o annotation and **f**alse) is irrelevant for non-AST evaluations, because the redex is not annotated and due to $m = \mathsf{f}$, afterwards one cannot rewrite an annotated term at a position above. For example, using the ADP $\mathsf{g}(\mathsf{a}) \overset{\mathsf{f}}{\hookrightarrow} \{{}^{1}\mathsf{g}^{\sharp}(\mathsf{b})\}$ instead to rewrite the non-annotated subterm $\mathsf{g}(\mathsf{a})$ yields $\mathsf{f}^{\sharp}(\underline{\mathsf{g}}(\mathsf{a})) \overset{\mathsf{i}}{\hookrightarrow}^{\mathbf{nf}}_{\widetilde{\mathcal{A}}} \{{}^{1}\mathsf{f}(\mathsf{g}(\mathsf{b}))\}$. The case **nf** is needed for innermost rewriting to ensure that normal forms always remain the same, even if we remove or add annotations in ADPs or if we modify their flag $m$. The flag $\mathsf{f}$ and steps of the form **nf** will allow us to obtain a usable rules processor which is *complete* (see Theorem 7.4.14), in contrast to the incomplete usable rules processor from Theorem 7.1.13 in the non-probabilistic setting.

### 7.2.2 Full Rewriting with ADPs

Next, we consider rewriting with ADPs w.r.t. an arbitrary evaluation strategy. As shown in Example 7.0.3, we have $\mathtt{AST}(\overset{\mathsf{i}}{\to}_{\mathcal{P}_{12}})$, but not $\mathtt{AST}(\to_{\mathcal{P}_{12}})$. To simulate the corresponding non-AST reduction from Example 7.0.3 with ADPs, it is crucial that if the variables of an ADP are instantiated by annotated terms, then these annotations must be regarded when rewriting with this ADP. For example, if one uses the ADP $\mathsf{d}(\mathsf{d}(x)) \overset{\mathsf{t}}{\hookrightarrow} \{{}^{1}\mathsf{c}(x, \mathsf{g}^{\sharp})\}$ (♯7.15) to rewrite the redex $\mathsf{d}^{\sharp}(\mathsf{d}^{\sharp}(\mathsf{g}^{\sharp}))$ (whose subterm $\mathsf{g}^{\sharp}$ starts a non-AST evaluation), then it should still be possible to continue the evaluation of the subterm $\mathsf{g}^{\sharp}$ afterwards which was "completely inside" the substitution of the applied rewrite step. (We will illustrate this in detail in Counterexample 7.2.15.)

So for full rewriting with ADPs, we have to keep certain annotations below the used redex. To this end, we use *variable reposition functions (VRFs)*[4] to relate positions of variables in the left-hand side of an ADP to those positions of the same variables in the right-hand sides where we want to keep the annotations of the instantiated variables. So for an ADP $\ell \overset{m}{\hookrightarrow} \mu$ with $\ell|_{\pi} = x$, we indicate which occurrence of $x$ in $r \in \mathrm{Supp}(\mu)$ should keep the annotations if one rewrites an instance of $\ell$ where the subterm at position $\pi$ contains annotations.

Definition 7.2.9 – Variable Reposition Functions.

*Let $\ell \overset{m}{\hookrightarrow} \{{}^{p_1}r_1, \dots, {}^{p_k}r_k\}$ be an ADP. A family of functions $(\varphi_j)_{1 \le j \le k}$ with $\varphi_j : \mathrm{pos}_{\mathcal{V}}(\ell) \to \mathrm{pos}_{\mathcal{V}}(r_j) \uplus \{\bot\}$ is called a family of variable reposition functions (VRF) for the ADP if for all $1 \le j \le k$ we have $\ell|_{\pi} = r_j|_{\varphi_j(\pi)}$ whenever $\varphi_j(\pi) \neq \bot$.*

Example 7.2.10. Let us consider the ADP $\mathsf{a}(x, y) \overset{\mathsf{t}}{\hookrightarrow} \{{}^{1}\mathsf{b}(x)\}$. Here, there are two possible VRFs, viz., $\varphi_1(1) = 1$ and $\varphi_1(2) = \bot$, or $\varphi_1(1) = \bot$ and $\varphi_1(2) = \bot$. We have to map the position of the variable $y$ to $\bot$, because $y$ does not occur on the right-hand side. The position of the variable $x$ can either be mapped to $\bot$ as well, or to the position 1, which is the position of $x$ on the right-hand side. This indicates whether we rewrite a term like $\mathsf{a}(\mathsf{f}^{\sharp}, \mathsf{g}^{\sharp})$ to $\mathsf{b}(\mathsf{f})$ or to $\mathsf{b}(\mathsf{f}^{\sharp})$. See Figure 7.3 for a graphical representation of these VRFs.

Note that VRFs do not allow us to duplicate annotations, i.e., we cannot use ADPs to analyze AST of duplicating rules like $\mathsf{d}(x) \to \{{}^{1}\mathsf{c}(x, x)\}$. A probabilistic rule or ADP $\ell \to \mu$ is *non-duplicating* (ND) if all rules in $\{\ell \to r \mid r \in \mathrm{Supp}(\mu)\}$ are, and a PTRS or ADP problem is non-duplicating if all of its rules are (disregarding the flag for ADPs).

[4] VRFs were introduced in [KVG24] when adapting ADPs to full *relative* rewriting. However, due to the probabilistic setting, our definition here is slightly different.

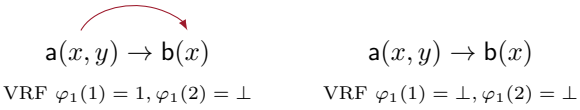



Figure 7.3: The two VRFs for the ADP $\mathsf{a}(x,y) \overset{\mathsf{t}}{\hookrightarrow} \{^1\mathsf{b}(x)\}$.

A VRF indicates at which positions the annotations should be kept. However, since ADPs do not have to be left-linear, this movement of annotations from the left- to the right-hand side is a bit more involved. Note that our VRFs do not have to be injective, i.e., for the ADP $\mathsf{c}(x,x) \overset{\mathsf{t}}{\hookrightarrow} \{^1\mathsf{d}(x)\}$ a valid VRF would be $\varphi_1$ with $\varphi_1(1) = 1$ and $\varphi_1(2) = 1$. Now consider the term $\mathsf{c}(\mathsf{g}(\mathsf{f}^\sharp), \mathsf{g}^\sharp(\mathsf{f}))$. Which annotations should be kept when rewriting at the root position with the ADP $\mathsf{c}(x,x) \overset{\mathsf{t}}{\hookrightarrow} \{^1\mathsf{d}(x)\}$? Since both the subterm $\mathsf{g}^\sharp(\mathsf{f})$ and the subterm $\mathsf{f}^\sharp$ may lead to a non-`AST` evaluation, this should still be possible after the rewrite step, since both subterms are completely "inside" the substitution used for matching. Hence, our rewrite relation allows for the step $\underline{\mathsf{c}}(\mathsf{g}(\mathsf{f}^\sharp), \mathsf{g}^\sharp(\mathsf{f})) \hookrightarrow_{\mathcal{A}} \{^1\mathsf{d}(\mathsf{g}^\sharp(\mathsf{f}^\sharp))\}$, i.e., we consider all terms corresponding to the same variable when applying an ADP and *merge* their annotations. In this way, we obtain a term that still contains all annotations that may lead to non-`AST` evaluations.

So whenever there is a position $\rho \in \mathrm{pos}_{\mathcal{V}}(\ell)$ of a variable in the left-hand side $\ell$ of an ADP and $\rho$ corresponds to a position $\varphi_j(\rho) \neq \bot$ of the same variable in the right-hand side $r_j$, then all annotations below the position $\rho$ in the redex $s|_\pi$ must be kept below the position $\varphi_j(\rho)$ in the resulting term $r_j\sigma$. In other words, whenever there is a position $\tau$ with an annotated symbol at position $\rho.\tau$ in the redex $s|_\pi$, then the symbol at position $\varphi_j(\rho).\tau$ in the resulting term $r_j\sigma$ should also be annotated. We collect all these positions in a set $\Psi_j$.

In our example with the ADP $\mathsf{c}(x,x) \overset{\mathsf{t}}{\hookrightarrow} \{^1\mathsf{d}(x)\}$, we have $\ell = \mathsf{c}(x,x)$ and $r_1 = \mathsf{d}(x)$. The redex $s|_\pi$ is $\mathsf{c}(\mathsf{g}(\mathsf{f}^\sharp), \mathsf{g}^\sharp(\mathsf{f}))$ and thus, for the matching substitution $\sigma$ with $\ell\sigma = \flat(s|_\pi)$, we obtain $r_1\sigma = \mathsf{d}(\mathsf{g}(\mathsf{f}))$. To find out which symbols should be annotated in $r_1\sigma$, we consider the VRF $\varphi_1(1) = 1$ and $\varphi_1(2) = 1$ which maps variable positions $\rho$ of $\ell$ to variable positions of $r_1$. For $\rho = 1$, the redex $\mathsf{c}(\mathsf{g}(\mathsf{f}^\sharp), \mathsf{g}^\sharp(\mathsf{f}))$ has an annotation at the position $\rho.\tau$ with $\tau = 1$. Hence, the position $\varphi_1(\rho).\tau = 1.1$ in $r_1\sigma$ should be annotated. Moreover, for $\rho' = 2$, the redex $\mathsf{c}(\mathsf{g}(\mathsf{f}^\sharp), \mathsf{g}^\sharp(\mathsf{f}))$ has an annotation at the position $\rho'.\tau'$ with $\tau' = \varepsilon$. Hence, the position $\varphi_1(\rho').\tau' = 1$ in $r_1\sigma$ should be annotated as well. Thus, for $\Psi_1 = \{1.1, 1\}$ we obtain $\sharp_{\Psi_1}(r_1\sigma) = \sharp_{\{1.1,1\}}(\mathsf{d}(\mathsf{g}(\mathsf{f}))) = \mathsf{d}(\mathsf{g}^\sharp(\mathsf{f}^\sharp))$.

Now we can define (full) rewriting with ADPs. As in the definition of innermost rewriting (Definition 7.2.8), all annotations of $r_j$ should be deleted whenever $\pi \notin \mathrm{pos}_{\mathcal{D}^\sharp}(s)$. Thus, we use $\sharp_{\Psi_j}(r_j\sigma)$ which removes all annotations from $r_j$ (and only adds certain annotations on the terms introduced by $\sigma$). In contrast, if $\pi \in \mathrm{pos}_{\mathcal{D}^\sharp}(s)$, then we use $\sharp_{\mathrm{pos}_{\mathcal{D}^\sharp}(r_j) \cup \Psi_j}(r_j\sigma)$ in order to keep the annotations of $r_j$.

Definition 7.2.11 – Rewriting with ADPs.

*Let $\mathcal{A}$ be an ADP problem. A term $s \in \mathcal{T}^\sharp$ rewrites with $\mathcal{A}$ to $\mu = \{^{p_1}t_1, \ldots, ^{p_k}t_k\}$ (denoted $s \hookrightarrow_{\mathcal{A}} \mu$) if there are an $\ell \overset{m}{\hookrightarrow} \{^{p_1}r_1, \ldots, ^{p_k}r_k\} \in \mathcal{A}$, a VRF $(\varphi_j)_{1 \leq j \leq k}$ for this ADP, a substitution $\sigma$, and a $\pi \in \mathrm{pos}_{\mathcal{D} \cup \mathcal{D}^\sharp}(s)$ such that $\flat(s|_\pi) = \ell\sigma$, and for all $1 \leq j \leq k$ the term $t_j$ is defined as follows, depending on the flag $m$ and on*

*whether* $\pi \in \mathrm{pos}_{\mathcal{D}^\sharp}(s)$ *holds:*

| | $\pi \in \mathrm{pos}_{\mathcal{D}^\sharp}(s)$ | $\pi \notin \mathrm{pos}_{\mathcal{D}^\sharp}(s)$ |
|---|---|---|
| $m = \mathtt{t}$ | $t_j = s[\sharp_{\mathrm{pos}_{\mathcal{D}^\sharp}(r_j)\cup\Psi_j}(r_j\sigma)]_\pi$ **(at)** | $t_j = s[\sharp_{\Psi_j}(r_j\sigma)]_\pi$ **(nt)** |
| $m = \mathtt{f}$ | $t_j = \flat^{\uparrow}_\pi(s[\sharp_{\mathrm{pos}_{\mathcal{D}^\sharp}(r_j)\cup\Psi_j}(r_j\sigma)]_\pi)$ **(af)** | $t_j = \flat^{\uparrow}_\pi(s[\sharp_{\Psi_j}(r_j\sigma)]_\pi)$ **(nf)** |

*Here,* $\Psi_j = \{\varphi_j(\rho).\tau \mid \rho \in \mathrm{pos}_{\mathcal{V}}(\ell),\ \varphi_j(\rho) \neq \bot,\ \rho.\tau \in \mathrm{pos}_{\mathcal{D}^\sharp}(s|_\pi)\}$.

Thus, innermost rewriting is the special case of Definition 7.2.11 where we require that $\flat(s|_\pi) = \ell\sigma$ is in argument normal form and where we use the VRF which maps all variable positions to $\bot$ (i.e., all annotations below the redex are removed).

### 7.2.3 Chains

For *chains* in the probabilistic setting we now consider specific RTs, called *chain trees*. Chain trees are defined analogously to RTs, but the crucial requirement is that every infinite path of the tree must contain infinitely many steps of the forms **at** or **af**.

Definition 7.2.12 – Chain Tree.
*Let* $\mathcal{A}$ *be an ADP problem. A labeled tree* $\mathfrak{T} = (N, E)$ *is an* $\mathcal{A}$*-chain tree (*$\mathcal{A}$*-CT) if the nodes* $v \in N$ *are labeled by pairs* ${}^{p_v}t_v$ *of a non-zero probability* $0 < p_v \leq 1$ *and a term* $t_v \in \mathcal{T}^\sharp$*, where the root* $\mathfrak{r}$ *is always labeled with the probability* $p_\mathfrak{r} = 1$*. For each inner node* $v \in N$ *with the successors* $w_1, \ldots, w_k \in N$*, the edge relation represents a reduction step with the relation* $\hookrightarrow_\mathcal{A}$*, i.e.,* $t_v \hookrightarrow_\mathcal{A} \{{}^{p_{w_1}/p_v}t_{w_1}, \ldots, {}^{p_{w_k}/p_v}t_{w_k}\}$*. An* $\mathcal{A}$*-CT comes with a labeling that fixes, for every inner node* $v$*, the ADP and the kind of step used for this rewrite step, denoted by* $\mathrm{Step}(v) \in \mathcal{A} \times \{\mathbf{at}, \mathbf{af}, \mathbf{nt}, \mathbf{nf}\}$*. By* $\Gamma \subseteq N$ *we denote the set of all inner nodes* $v$ *whose step* $\mathrm{Step}(v)$ *uses Case* **at** *or* **af***, and by* $\overline{\Gamma}$ *we denote the set of all other inner nodes. Additionally, every infinite path in* $\mathfrak{T}$ *has to contain infinitely many nodes from* $\Gamma$ *(called the liveness property).* $\mathfrak{T}$ *is an innermost chain tree if the edges represent innermost rewrite steps. As for RTs, we say that an* $\mathcal{A}$*-CT converges (or terminates) with probability* $|\mathfrak{T}| = \sum_{v \in \mathrm{Leaf}^\mathfrak{T}} p_v$.

With this notion of chain trees, we can now define $\mathtt{AST}$ for ADP problems.

Definition 7.2.13 – AST for ADP Problems.
*An ADP problem* $\mathcal{A}$ *is called almost surely terminating (*$\mathtt{AST}(\hookrightarrow_\mathcal{A})$*) if we have* $|\mathfrak{T}| = 1$ *for every* $\mathcal{A}$*-CT* $\mathfrak{T}$*. It is called almost surely innermost terminating (*$\mathtt{AST}(\overset{\mathtt{i}}{\hookrightarrow}_\mathcal{A})$*) if we have* $|\mathfrak{T}| = 1$ *for every innermost* $\mathcal{A}$*-CT* $\mathfrak{T}$.

Since innermost rewriting with $\mathcal{A}$ is the special case of Definition 7.2.11 that uses the VRF which maps all variable positions to $\bot$ (and requires $\flat(s|_\pi) \in \mathtt{ANF}_\mathcal{A}$), every innermost $\mathcal{A}$-CT is also an $\mathcal{A}$-CT. Hence, $\mathtt{AST}(\hookrightarrow_\mathcal{A})$ implies $\mathtt{AST}(\overset{\mathtt{i}}{\hookrightarrow}_\mathcal{A})$ for every ADP problem $\mathcal{A}$.

Remark 7.2.14
With a slight abuse of notation, we write $\mathtt{AST}(\overset{\mathtt{i}}{\hookrightarrow}_\mathcal{A})$ even though we do not only consider $\overset{\mathtt{i}}{\hookrightarrow}_\mathcal{A}$-RTs but $\overset{\mathtt{i}}{\hookrightarrow}_\mathcal{A}$-CTs instead. So whenever we write $P(\to)$ where $\to \in \{\hookrightarrow_\mathcal{A}, \overset{\mathtt{i}}{\hookrightarrow}_\mathcal{A}\}$ for an ADP problem $\mathcal{A}$, we consider $\to$-CTs instead of $\to$-RTs.

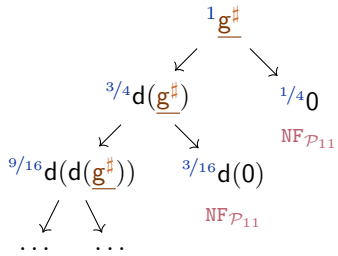



Figure 7.4: Innermost $\mathcal{DP}(\mathcal{P}_{11})$-chain tree.

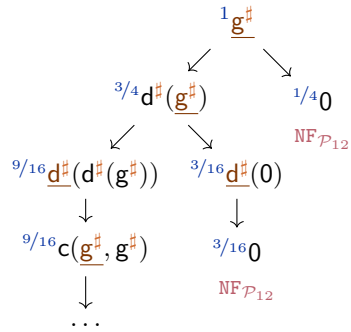



Figure 7.5: $\mathcal{DP}(\mathcal{P}_{12})$-chain tree.

> The difference is the liveness property of $\to$-CTs that is not required for $\to$-RTs.

The corresponding $\mathcal{DP}(\mathcal{P}_{11})$-chain tree $\mathfrak{T}$ for the $\to_{\mathcal{P}_{11}}$-RT from Figure 7.1 is shown in Figure 7.4. Here, we have $|\mathfrak{T}| = 1$.

Counterexample 7.2.15. To see why VRFs are needed to obtain a sound chain criterion for full rewriting, consider $\mathcal{P}_{12}$ with $\mathcal{DP}(\mathcal{P}_{12}) = \{(\sharp 7.12), (\sharp 7.13), (\sharp 7.15)\}$ from Example 7.2.6:

$$\mathsf{g} \overset{\mathrm{t}}{\hookrightarrow} \{^{3/4}\mathsf{d}^\sharp(\mathsf{g}^\sharp), {}^{1/4}0\} \quad (\sharp 7.12) \qquad \mathsf{d}(\mathsf{d}(x)) \overset{\mathrm{t}}{\hookrightarrow} \{^{1}\mathsf{c}(x, \mathsf{g}^\sharp)\} \quad (\sharp 7.15)$$
$$\mathsf{d}(x) \overset{\mathrm{t}}{\hookrightarrow} \{^{1}0\} \quad (\sharp 7.13)$$

As noted in Example 7.0.3, $\mathtt{AST}(\overset{\mathrm{i}}{\to}_{\mathcal{P}_{12}})$ holds, but $\mathtt{AST}(\to_{\mathcal{P}_{12}})$ does not. For $\mathcal{DP}(\mathcal{P}_{12})$, applying two rewrite steps with the ADP ($\sharp 7.12$) to $\mathsf{g}^\sharp$ would result in a chain tree with the leaves $^{9/16}\mathsf{d}^\sharp(\mathsf{d}^\sharp(\mathsf{g}^\sharp))$, $^{3/16}\mathsf{d}^\sharp(0)$ (which can be extended by the child $^{3/16}0$), and $^{1/4}0$. Now it is important that the next application of the ADP ($\sharp 7.15$) to $^{9/16}\mathsf{d}^\sharp(\mathsf{d}^\sharp(\mathsf{g}^\sharp))$ allows us to keep the annotation of its argument $\mathsf{g}^\sharp$, i.e., it can yield $^{9/16}\mathsf{c}(\mathsf{g}^\sharp, \mathsf{g}^\sharp)$. The corresponding chain tree is depicted in Figure 7.5. Otherwise, i.e., if it could only yield $^{9/16}\mathsf{c}(\mathsf{g}, \mathsf{g}^\sharp)$, the number of $\mathsf{g}^\sharp$-symbols in such a chain tree could never exceed one. Since non-annotated symbols like $\mathsf{g}$ do not result in any **at**- or **af**-steps, for all such chain trees $\mathfrak{T}$ we would have $|\mathfrak{T}| = 1$, i.e., then the chain criterion would falsely conclude that $\mathcal{P}_{12}$ is $\mathtt{AST}$.

For innermost rewriting, we now indeed obtain a sound and complete chain criterion as in the non-probabilistic setting, i.e., to analyze $\mathtt{AST}(\overset{\mathrm{i}}{\to}_{\mathcal{P}})$ for a PTRS $\mathcal{P}$ one can analyze $\mathtt{AST}(\overset{\mathrm{i}}{\hookrightarrow}_{\mathcal{DP}(\mathcal{P})})$ of its canonical ADP problem instead.

> Theorem 7.2.16 – Chain Criterion for $\mathtt{AST}(\overset{\mathrm{i}}{\to}_{\mathcal{P}})$.
> Let $\mathcal{P}$ be a finite PTRS. Then $\mathtt{AST}(\overset{\mathrm{i}}{\to}_{\mathcal{P}}) \iff \mathtt{AST}(\overset{\mathrm{i}}{\hookrightarrow}_{\mathcal{DP}(\mathcal{P})})$.

To prove the innermost chain criterion, we define the set $\mathrm{pos}_{\mathcal{D}\wedge\neg\mathtt{NF}_{\mathcal{P}}}(t)$ of all positions of subterms of $t$ that may be used as a redex now or in future rewrite steps, because the subterm has a defined root symbol and is not in normal form w.r.t. $\mathcal{P}$. This set is needed, as we cannot simply annotate every defined symbol in the soundness proof of the innermost chain criterion. Instead, we can (and it suffices to) only guarantee that all the symbols in $\mathrm{pos}_{\mathcal{D}\wedge\neg\mathtt{NF}_{\mathcal{P}}}(t)$ are annotated.

Definition 7.2.17 – $\mathrm{pos}_{\mathcal{D}\wedge\neg\mathtt{NF}_\mathcal{P}}(t)$.

*Let $\mathcal{P}$ be a PTRS. For a term $t \in \mathcal{T}$ we define*

$$\mathrm{pos}_{\mathcal{D}\wedge\neg\mathtt{NF}_\mathcal{P}}(t) = \{\pi \mid \pi \in \mathrm{pos}_\mathcal{D}(t), t|_\pi \notin \mathtt{NF}_\mathcal{P}\}.$$

Example 7.2.18. Consider the following PTRS $\mathcal{P}$ over a signature with $\mathcal{D} = \{\mathsf{f}, \mathsf{g}\}$ and $\mathcal{C} = \{\mathsf{a}, \mathsf{b}\}$ with the rules $\mathsf{f}(x, \mathsf{a}) \to \{^1\mathsf{b}\}$ and $\mathsf{g}(\mathsf{a}) \to \{^1\mathsf{a}\}$. For the term $t = \mathsf{f}(\mathsf{g}(\mathsf{a}), \mathsf{g}(\mathsf{b}))$ we have $\mathrm{pos}_\mathcal{D}(t) = \{\varepsilon, 1, 2\}$ and $\mathrm{pos}_{\mathcal{D}\wedge\neg\mathtt{NF}_\mathcal{P}}(t) = \{\varepsilon, 1\}$.

Finally, for two (possibly annotated) terms $s, t$ we define $s \overset{\flat}{=} t$ if $\flat(s) = \flat(t)$. Remember that for innermost rewriting, we do not use any VRFs, hence, we can ignore them in the proofs for innermost rewriting.

*Proof of Theorem 7.2.16.* We prove both directions individually. *Proof*

In the following, we will often implicitly use that for an annotated term $t \in \mathcal{T}^\sharp$, we have $\flat(t) \in \mathtt{ANF}_\mathcal{P}$ iff $t \in \mathtt{ANF}_{\mathcal{DP}(\mathcal{P})}$ since a rewrite rule and its corresponding canonical annotated dependency pair have the same left-hand side.

▶ *Proof*($\Longleftarrow$, *Soundness*): Assume that $\mathtt{AST}(\overset{\mathtt{i}}{\to}_\mathcal{P})$ does not hold. Then, there exists an $\overset{\mathtt{i}}{\to}_\mathcal{P}$-RT $\mathfrak{T} = (N, E)$ whose root is labeled with $t$ for some term $t \in \mathcal{T}$ that converges with probability $< 1$. We will construct an innermost $\mathcal{DP}(\mathcal{P})$-CT $\mathfrak{T}' = (N, E)$ with the same underlying tree structure such that $p_v^\mathfrak{T} = p_v^{\mathfrak{T}'}$ for all $v \in N$, where all the inner nodes are in $\Gamma$. Since the tree structure and the probabilities are the same, we then get $|\mathfrak{T}| = |\mathfrak{T}'|$. So there exists an innermost $\mathcal{DP}(\mathcal{P})$-CT $\mathfrak{T}'$ that converges with probability $< 1$ and $\mathtt{AST}(\overset{\mathtt{i}}{\to}_{\mathcal{DP}(\mathcal{P})})$ does not hold either.

We construct the terms $t'_v$ for the nodes of the $\mathcal{DP}(\mathcal{P})$-CT inductively such that for all inner nodes $v \in N \setminus \mathrm{Leaf}$ with children nodes $vE = \{w_1, \dots, w_k\}$ we have $t'_v \overset{\mathtt{i}}{\hookrightarrow}_{\mathcal{DP}(\mathcal{P})} \{^{p_{w_1}/p_v}t'_{w_1}, \dots, ^{p_{w_k}/p_v}t'_{w_k}\}$ and use Case **at** so that all nodes are in $\Gamma$. Let $W \subseteq N$ be the set of nodes $v$ where we have already defined the corresponding term $t'_v$. During our construction, we ensure that for all $v \in W$ we have

$$t_v \overset{\flat}{=} t'_v \text{ and } \mathrm{pos}_{\mathcal{D}\wedge\neg\mathtt{NF}_\mathcal{P}}(t_v) \subseteq \mathrm{pos}_{\mathcal{D}^\sharp}(t'_v). \tag{7.26}$$

This means that the corresponding term $t_v$ for the node $v$ in $\mathfrak{T}$ has the same structure as the term $t'_v$ in $\mathfrak{T}'$, and additionally, all the possible redexes in $t_v$ are annotated in $t'_v$, see Figure 7.6. We label the root of $\mathfrak{T}'$ with $\sharp_\mathcal{D}(t)$. Here, we have $t \overset{\flat}{=} \sharp_\mathcal{D}(t)$ and $\mathrm{pos}_{\mathcal{D}\wedge\neg\mathtt{NF}_\mathcal{P}}(t) \subseteq \mathrm{pos}_\mathcal{D}(t) = \mathrm{pos}_{\mathcal{D}^\sharp}(\sharp_\mathcal{D}(t))$. As long as there is still an inner node $v \in W$ such that its successors are not contained in $W$, we do the following. Let $vE = \{w_1, \dots, w_k\}$ be the set of its successors. We need to define the corresponding terms $t'_{w_1}, \dots, t'_{w_k}$ for the nodes $w_1, \dots, w_k$. Since $v$ is not a leaf, we have $t_v \overset{\mathtt{i}}{\to}_\mathcal{P} \{^{p_{w_1}/p_v}t_{w_1}, \dots, ^{p_{w_k}/p_v}t_{w_k}\}$. This means that there is a rule $\ell \to \{^{p_1}r_1, \dots, ^{p_k}r_k\} \in \mathcal{P}$, a position $\pi$, and a substitution $\sigma$ such that $t_v|_\pi = \ell\sigma \in \mathtt{ANF}_\mathcal{P}$. Furthermore, we have $t_{w_j} = t_v[r_j\sigma]_\pi$ for all $1 \le j \le k$. So the node $w_j$ in $\mathfrak{T}$ is labeled by $^{p_v \cdot p_j}t_v[r_j\sigma]_\pi$ for all $1 \le j \le k$.

The corresponding ADP for the rule is $\ell \overset{\mathtt{t}}{\hookrightarrow} \{^{p_1}\sharp_\mathcal{D}(r_1), \dots, ^{p_k}\sharp_\mathcal{D}(r_k)\}$. Furthermore, $\pi \in \mathrm{pos}_{\mathcal{D}\wedge\neg\mathtt{NF}_\mathcal{P}}(t_v) \subseteq_{(IH)} \mathrm{pos}_{\mathcal{D}^\sharp}(t'_v)$ and $t_v \overset{\flat}{=}_{(IH)} t'_v$. Hence, we can rewrite $t'_v$ with $\ell \overset{\mathtt{t}}{\hookrightarrow} \{^{p_1}\sharp_\mathcal{D}(r_1), \dots, ^{p_k}\sharp_\mathcal{D}(r_k)\}$, using the position $\pi$ and the substitution $\sigma$, and Case **at** applies. We get $t'_v \overset{\mathtt{i}}{\hookrightarrow}_{\mathcal{DP}(\mathcal{P})} \{^{p_1}t'_{w_1}, \dots, ^{p_k}t'_{w_k}\}$ with $t'_{w_j} = t'_v[\sharp_\mathcal{D}(r_j)\sigma]_\pi$. This means that we have $t_{w_j} \overset{\flat}{=} t'_{w_j}$. It remains to prove $\mathrm{pos}_{\mathcal{D}\wedge\neg\mathtt{NF}_\mathcal{P}}(t_{w_j}) \subseteq \mathrm{pos}_{\mathcal{D}^\sharp}(t'_{w_j})$ for all $1 \le j \le k$. For all positions $\tau \in \mathrm{pos}_{\mathcal{D}\wedge\neg\mathtt{NF}_\mathcal{P}}(t_{w_j}) = \mathrm{pos}_{\mathcal{D}\wedge\neg\mathtt{NF}_\mathcal{P}}(t_v[r_j\sigma]_\pi)$ that

are orthogonal to or above $\pi$, we have $\tau \in \mathrm{pos}_{\mathcal{D}\wedge\neg\mathtt{NF}_\mathcal{P}}(t_v) \subseteq_{(IH)} \mathrm{pos}_{\mathcal{D}^\sharp}(t'_v)$, and all annotations orthogonal to or above $\pi$ remain in $t'_{w_j}$ as they were in $t'_v$. For all positions $\tau \in \mathrm{pos}_{\mathcal{D}\wedge\neg\mathtt{NF}_\mathcal{P}}(t_{w_j}) = \mathrm{pos}_{\mathcal{D}\wedge\neg\mathtt{NF}_\mathcal{P}}(t_v[r_j\sigma]_\pi)$ that are below $\pi$, we know that, due to innermost evaluation, the root of any subterm below $\pi$ that is not in normal form must lie inside $r_j$ (as the terms introduced by $\sigma$ are in normal form), and thus $\tau \in \mathrm{pos}_{\mathcal{D}^\sharp}(t'_{w_j})$, as all defined symbols of $r_j$ are annotated in $t'_{w_j} = t'_v[\sharp_\mathcal{D}(r_j)\sigma]_\pi$.

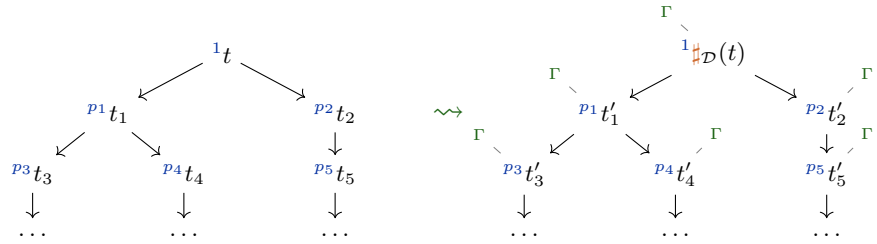



Figure 7.6: Construction for the soundness proof of Theorem 7.2.16.

▶ *Proof*($\Longrightarrow$, *Completeness*): Assume that $\mathtt{AST}(\overset{\mathtt{i}}{\hookrightarrow}_{\mathcal{DP}(\mathcal{P})})$ does not hold. Then, there exists an innermost $\mathcal{DP}(\mathcal{P})$-CT $\mathfrak{T} = (N, E)$ whose root is labeled with $t$ for some annotated term $t \in \mathcal{T}^\sharp$ that converges with probability $< 1$. We will construct an $\overset{\mathtt{i}}{\to}_\mathcal{P}$-RT $\mathfrak{T}' = (N, E)$ with the same underlying tree structure such that $p_v^\mathfrak{T} = p_v^{\mathfrak{T}'}$ for all $v \in N$. Since the tree structure and the probabilities are the same, we then get $|\mathfrak{T}'| = |\mathfrak{T}|$. Therefore, there exists an $\overset{\mathtt{i}}{\to}_\mathcal{P}$-RT $\mathfrak{T}'$ that converges with probability $< 1$. Hence, $\mathtt{AST}(\overset{\mathtt{i}}{\to}_\mathcal{P})$ does not hold either.

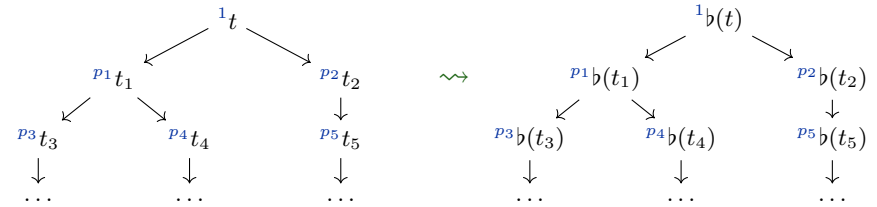



Figure 7.7: Construction for the completeness proof of Theorem 7.2.16.

For every node $v \in N$, the label of $v$ in $\mathfrak{T}'$ is $^{p_v^\mathfrak{T}}\flat(t_v)$, where $t_v$ is the term for the node $v$ in $\mathfrak{T}$, i.e., we remove all annotations, see Figure 7.7. We only have to show that $\mathfrak{T}'$ is indeed a valid RT, i.e., that the edge relation represents valid rewrite steps with $\overset{\mathtt{i}}{\to}_\mathcal{P}$, but this follows directly from the fact that if we remove all annotations in Definition 7.2.8, then we get the ordinary innermost probabilistic term rewriting relation again. *QED*

For the chain criterion for full AST, we have to impose a restriction on the initial PTRS, namely that the PTRS must be ND. As mentioned before, our VRFs (and the corresponding full rewrite relation for ADPs) cannot handle duplicating rules correctly. For example, for the duplicating ADP $\mathtt{d}(x) \overset{\mathtt{t}}{\hookrightarrow} \{^1\mathtt{c}(x, x)\}$ ($\sharp$7.14), we have three different VRFs which map position 1 to either $\bot$, 1, or 2, but we cannot map it to both positions 1 and 2. Hence, $\mathcal{DP}(\mathcal{P}_\mathsf{dup})$ is AST although $\mathcal{P}_\mathsf{dup}$ is not, as $\mathtt{d}^\sharp(\mathtt{g}^\sharp)$ only rewrites to $\{^1\mathtt{c}(\mathtt{g}^\sharp, \mathtt{g})\}$ or $\{^1\mathtt{c}(\mathtt{g}, \mathtt{g}^\sharp)\}$, but the annotation cannot be duplicated. Thus, the chain criterion would be unsound for duplicating PTRSs like $\mathcal{P}_\mathsf{dup}$.

Alternatively, one could modify the definition of the rewrite relation $\hookrightarrow_{\mathcal{A}}$ and use *generalized* VRFs (GVRFs) instead of VRFs which can duplicate annotations. This would yield a sound and complete chain criterion for $\mathtt{AST}$ of possibly duplicating PTRSs, but then one would also have to consider this modified definition of $\hookrightarrow_{\mathcal{A}}$ for the processors of the ADP framework in Section 7.3. Unfortunately, the most important processors would become unsound when defining the rewrite relation $\hookrightarrow_{\mathcal{A}}$ via GVRFs (see Ex. 7.5.4 and 7.4.4). Therefore, we use VRFs instead and restrict ourselves to non-duplicating PTRSs for the soundness of the chain criterion for full probabilistic rewriting.[5]

**Theorem 7.2.19** – Chain Criterion for AST.
Let $\mathcal{P}$ be a finite PTRS. Then $\mathtt{AST}(\to_{\mathcal{P}}) \implies \mathtt{AST}(\hookrightarrow_{\mathcal{DP}(\mathcal{P})})$. If $\mathcal{P}$ is $\mathtt{ND}$, then we even have $\mathtt{AST}(\to_{\mathcal{P}}) \iff \mathtt{AST}(\hookrightarrow_{\mathcal{DP}(\mathcal{P})})$.

*Proof.* We show both directions individually. *Proof*

▶ *Proof*($\Longleftarrow$, *Soundness*): Let $\mathcal{P}$ be $\mathtt{ND}$ and assume that $\mathtt{AST}(\to_{\mathcal{P}})$ does not hold. Then, there exists a $\to_{\mathcal{P}}$-RT $\mathfrak{T} = (N, E)$ whose root is labeled with $t$ for some term $t \in \mathcal{T}$ that converges with probability $< 1$. We use almost the same construction as in the soundness proof of Theorem 7.2.16, but here we can simply annotate all the defined symbols. Therefore, there exists a $\mathcal{DP}(\mathcal{P})$-CT $\mathfrak{T}'$ that converges with probability $< 1$ and $\mathtt{AST}(\hookrightarrow_{\mathcal{DP}(\mathcal{P})})$ does not hold either.

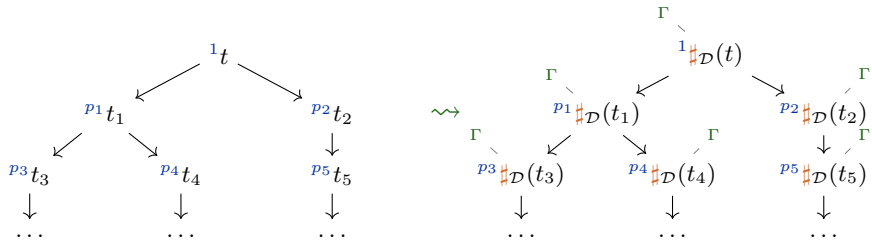



Figure 7.8: Construction for the soundness proof of Theorem 7.2.19.

For every node $v \in N$, the label of $v$ in $\mathfrak{T}'$ is $^{p_v^{\mathfrak{T}}}\sharp_{\mathcal{D}}(t_v)$, where $t_v$ is the term for the node $v$ in $\mathfrak{T}$. The annotations ensure that we rewrite with Case **at** so that the node $v$ is contained in $\Gamma$, see Figure 7.8. We only have to show that the edge relation represents valid rewrite steps with $\hookrightarrow_{\mathcal{DP}(\mathcal{P})}$. Let $v \in N \setminus \mathrm{Leaf}$ and $vE = \{w_1, \dots, w_k\}$ be the set of its successors. Since $v$ is not a leaf, we have $t_v \to_{\mathcal{P}} \{^{p_{w_1}/p_v}t_{w_1}, \dots, ^{p_{w_k}/p_v}t_{w_k}\}$. This means that there is a rule $\ell \to \{^{p_1}r_1, \dots, ^{p_k}r_k\} \in \mathcal{P}$, a position $\pi$, and a substitution $\sigma$ such that $t_v|_\pi = \ell\sigma$. Furthermore, we have $t_{w_j} = t_v[r_j\sigma]_\pi$ for all $1 \le j \le k$.
The corresponding ADP for the rule is $\ell \overset{\mathtt{t}}{\hookrightarrow} \{^{p_1}\sharp_{\mathcal{D}}(r_1), \dots, ^{p_k}\sharp_{\mathcal{D}}(r_k)\}$. Furthermore, $\pi \in \mathrm{pos}_{\mathcal{D}^\sharp}(\sharp_{\mathcal{D}}(t_v))$ as all defined symbols are annotated in $\sharp_{\mathcal{D}}(t_v)$. Hence, we can rewrite $\sharp_{\mathcal{D}}(t_v)$ with $\ell \overset{\mathtt{t}}{\hookrightarrow} \{^{p_1}\sharp_{\mathcal{D}}(r_1), \dots, ^{p_k}\sharp_{\mathcal{D}}(r_k)\}$, using the position $\pi$ and the substitution $\sigma$, and Case **at** applies. Furthermore, we take some VRF $(\varphi_j)_{1\le j\le k}$ that is surjective on the positions of the variables in the right-hand side, i.e., for all $1 \le j \le k$ and all positions $\tau \in \mathrm{pos}_{\mathcal{V}}(r_j)$ there exists a $\tau' \in \mathrm{pos}_{\mathcal{V}}(\ell)$ such that $\varphi_j(\tau') = \tau$. Such a VRF must exist, since $\mathcal{P}$ is $\mathtt{ND}$. We have $\sharp_{\mathcal{D}}(t_v) \hookrightarrow_{\mathcal{DP}(\mathcal{P})} \{^{p_1}\sharp_{\mathcal{D}}(t_{w_1}), \dots, ^{p_k}\sharp_{\mathcal{D}}(t_{w_k})\}$ since

[5] A related restriction is needed when analyzing (non-probabilistic) relative termination due to the VRFs [KVG24].

by rewriting with Case **at** we get $\sharp_{\mathcal{D}}(t_v)[\sharp_{\mathrm{pos}_{\mathcal{D}^\sharp}(\sharp_{\mathcal{D}}(r_j))\cup\Psi_j}(r_j\sigma)]_\pi = \sharp_{\mathcal{D}}(t_{w_j})$ with $\Psi_j$ defined as in Definition 7.2.11. Note that since the used VRF is surjective on the variable positions of the right-hand side, we do not remove any annotation in the substitution. Furthermore, we annotated all defined symbols in $r_j$. Thus, we obtain $\sharp_{\mathcal{D}}(t_{w_j})$ where all defined symbols are annotated again.

► *Proof* ($\Longrightarrow$, *Completeness*): Analogous to completeness for iAST, since removing all annotations in Definition 7.2.11 yields the ordinary probabilistic rewrite relation $\to_{\mathcal{P}}$ again. This direction does not need ND. *QED*

Example 7.2.20. As an example, the PTRS $\mathcal{P}_{\mathsf{alg}}$ from Example 7.0.2 is ND. Hence, we can not only analyze $\mathtt{AST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}_{\mathsf{alg}}})$ but also $\mathtt{AST}(\to_{\mathcal{P}_{\mathsf{alg}}})$ via the canonical ADP problem $\mathcal{DP}(\mathcal{P}_{\mathsf{alg}})$ from Example 7.2.7.

> Remark 7.2.21
> In the chain criterion for non-probabilistic DPs, it suffices to regard only instantiations where all terms below an annotated symbol are terminating. The reason is the *minimality property*, i.e., whenever a term starts an infinite rewrite sequence, then it also starts an infinite sequence, where all proper subterms of every used redex are terminating. However, we have seen in Counterexample 5.3.1 that such a minimality property does not hold for AST.

An ADP problem $\mathcal{A}$ without annotations always satisfies $\mathtt{AST}(\hookrightarrow_{\mathcal{A}})$ and $\mathtt{AST}(\overset{\mathtt{i}}{\hookrightarrow}_{\mathcal{A}})$, because then no rewrite step with $\hookrightarrow_{\mathcal{A}}$ increases the number of annotations (recall that VRFs cannot duplicate annotations). Hence, any term with $n$ annotations only starts rewrite sequences with at most $n$ steps of the form **at** or **af**, i.e., all $\mathcal{A}$-CTs are finite. We call such an ADP problem *solved*.

SECTION 7.3

## The ADP Framework for AST

The idea of the DP framework for non-probabilistic TRSs is to apply *DP processors* repeatedly which transform a DP problem into simpler sub-problems as we have seen in Section 7.1. The same idea is used in the ADP framework, which yields the first *modular* approach for proving $\mathtt{AST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}})$ and $\mathtt{AST}(\to_{\mathcal{P}})$ (the latter for ND PTRSs) of probabilistic TRSs automatically. An *ADP processor* Proc has the form $\mathrm{Proc}(\mathcal{A}) = \{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$ for ADP problems $\mathcal{A}, \mathcal{A}_1, \ldots, \mathcal{A}_n$. Proc is *sound for* AST if $\mathtt{AST}(\hookrightarrow_{\mathcal{A}})$ holds whenever $\mathtt{AST}(\hookrightarrow_{\mathcal{A}_j})$ holds for all $1 \le j \le n$. It is *complete for* AST if $\mathtt{AST}(\hookrightarrow_{\mathcal{A}_j})$ holds for all $1 \le j \le n$ whenever $\mathtt{AST}(\hookrightarrow_{\mathcal{A}})$ holds. Analogously, Proc is *sound* or *complete* for iAST if the same implication holds for $\mathtt{AST}(\overset{\mathtt{i}}{\hookrightarrow}_{\mathcal{A}})$. Thus, one starts with the canonical ADP problem and applies sound (and preferably complete) ADP processors repeatedly until all remaining ADP problems are solved. This implies that the canonical ADP problem satisfies $\mathtt{AST}(\hookrightarrow_{\mathcal{DP}(\mathcal{P})})$ or $\mathtt{AST}(\overset{\mathtt{i}}{\hookrightarrow}_{\mathcal{DP}(\mathcal{P})})$ and by the chain criterion, the original PTRS satisfies $\mathtt{AST}(\to_{\mathcal{P}})$ (provided that $\mathcal{P}$ is ND) or $\mathtt{AST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}})$, respectively. This idea is visualized in Figure 7.9.

In Section 7.4–7.6, we adapt the main processors from Section 7.1 to our new ADP framework for $\mathtt{AST}(\hookrightarrow_{\mathcal{A}})$ and $\mathtt{AST}(\overset{\mathtt{i}}{\hookrightarrow}_{\mathcal{A}})$ in addition to several other processors that either originate from the non-probabilistic DP framework, or are specifically designed for probabilistic rewriting. Whenever the respective processor can be used for proving $\mathtt{AST}(\hookrightarrow_{\mathcal{A}})$, we explain the variant of the processor for full rewriting first, and afterwards

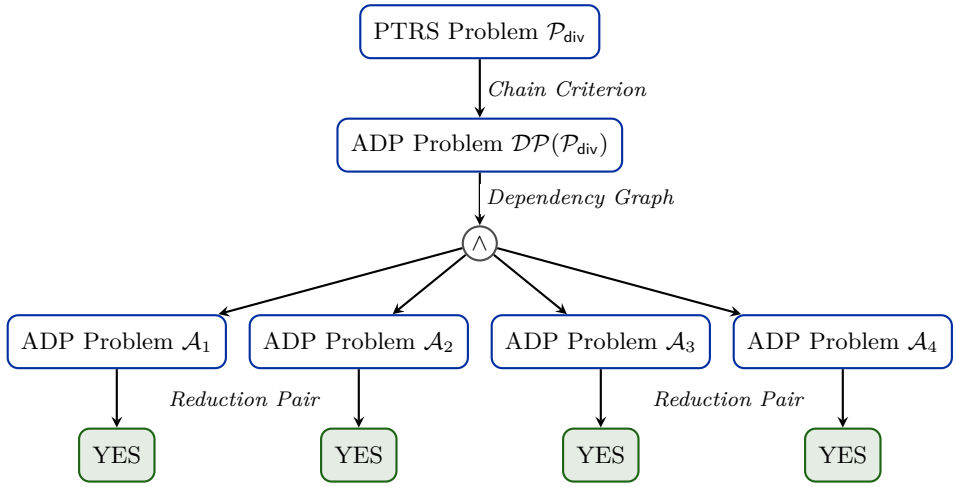



Figure 7.9: A schematic proof tree of the ADP framework for AST. It shows how to prove $\mathtt{AST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}_{\mathsf{div}}})$ for the PTRS $\mathcal{P}_{\mathsf{div}}$ from Counterexample 7.0.1.

we show how one can improve the processor for $\mathtt{AST}(\xrightarrow{\mathtt{i}}_{\mathcal{A}})$.

ADP processors that only remove annotations and change flags of ADPs from t to f are always complete.

Theorem 7.3.1 – Completeness of ADP Processors.

Let $\mathcal{A}$ be an ADP problem and let Proc be an ADP processor such that for every $\mathcal{A}' \in \mathrm{Proc}(\mathcal{A})$ we have: whenever there is an ADP $\ell \xrightarrow{m'} \{{}^{p_1}r'_1, \ldots, {}^{p_k}r'_k\} \in \mathcal{A}'$, then there is an ADP $\ell \xrightarrow{m} \{{}^{p_1}r_1, \ldots, {}^{p_k}r_k\} \in \mathcal{A}$ such that $\flat(r'_j) = \flat(r_j)$ and $\mathrm{pos}_{\mathcal{D}^\sharp}(r'_j) \subseteq \mathrm{pos}_{\mathcal{D}^\sharp}(r_j)$ for all $1 \leq j \leq k$, and such that $m' = \mathtt{t}$ implies $m = \mathtt{t}$. Moreover, for every ADP $\ell \xrightarrow{m} \mu \in \mathcal{A}$, the ADP problem $\mathcal{A}'$ contains an ADP with the same left-hand side $\ell$. Then Proc is complete for both $\mathtt{AST}(\hookrightarrow_{\mathcal{A}})$ and $\mathtt{AST}(\xrightarrow{\mathtt{i}}_{\mathcal{A}})$.

*Proof.* Each (innermost) $\mathcal{A}'$-CT $\mathfrak{T}'$ can be transformed into an (innermost) $\mathcal{A}$-CT $\mathfrak{T}$ with $|\mathfrak{T}'| = |\mathfrak{T}|$ by performing the respective steps with $\ell \xrightarrow{m} \{{}^{p_1}r_1, \ldots, {}^{p_k}r_k\} \in \mathcal{A}$ instead of $\ell \xrightarrow{m'} \{{}^{p_1}r'_1, \ldots, {}^{p_k}r'_k\} \in \mathcal{A}'$. Since we have $\flat(r'_j) = \flat(r_j)$ and $\mathrm{pos}_{\mathcal{D}^\sharp}(r'_j) \subseteq \mathrm{pos}_{\mathcal{D}^\sharp}(r_j)$ for all $1 \leq j \leq k$, and $m = \mathtt{t}$ if $m' = \mathtt{t}$, $\mathfrak{T}$ has the same terms as $\mathfrak{T}'$ but possibly with more annotations. Since every infinite path in $\mathfrak{T}'$ contains infinitely many rewrite steps with Case **at** or **af**, this ensures that $\mathfrak{T}$ satisfies the liveness property as well. For innermost rewriting, note that $\mathcal{A}$ and $\mathcal{A}'$ have the same left-hand sides, hence $\mathtt{ANF}_{\mathcal{A}} = \mathtt{ANF}_{\mathcal{A}'}$, so every step that is innermost w.r.t. $\mathcal{A}'$ is also innermost w.r.t. $\mathcal{A}$. *Proof* *QED*

The above proof structure will often be used for the following soundness proofs as well. Given some $\mathcal{A}$-CT $\mathfrak{T}$, we use an $\mathcal{A}' \in \mathrm{Proc}(\mathcal{A})$ and construct an $\mathcal{A}'$-CT $\mathfrak{T}'$ such that $|\mathfrak{T}'| \leq |\mathfrak{T}|$. If the processor Proc removes annotations, then one has to ensure that the liveness property of $\mathfrak{T}'$ holds again.

Moreover, the following two lemmas are important for the soundness proofs in general. The first lemma states that if we partition the annotated nodes of a CT $\mathfrak{T}$ into $\Gamma_1 \uplus \Gamma_2$ such that every subtree of $\mathfrak{T}$ that uses annotated steps only from $\Gamma_1$ converges with

probability 1, then it suffices to consider subtrees where every infinite path has an infinite number of annotated rewrite steps with nodes from $\Gamma_2$ (i.e., infinitely many rewrite steps with Case **at** or **af** at nodes from $\Gamma_2$). This is a direct consequence of the Sequential Execution Lemma for AST (Lemma 5.1.3).

Lemma 7.3.2 – Γ-Partition Lemma.
*Let $\mathcal{A}$ be an ADP problem and let $\mathfrak{T}$ be an $\mathcal{A}$-CT that converges with probability $< 1$. Let $\Gamma^{\mathfrak{T}} = \Gamma_1 \uplus \Gamma_2$ be a partition of the annotated nodes of $\mathfrak{T}$ such that every subtree of $\mathfrak{T}$ whose inner nodes all belong to $\overline{\Gamma}^{\mathfrak{T}} \cup \Gamma_1$ converges with probability 1. Then there exists a grounded subtree $\mathfrak{T}'$ of $\mathfrak{T}$ (i.e., a subtree that contains the root of $\mathfrak{T}$) that converges with probability $< 1$ such that every infinite path uses an infinite number of nodes from $\Gamma_2$. In particular, $\mathfrak{T}'$ is again an (innermost) $\mathcal{A}$-CT if $\mathfrak{T}$ is one.*

*Proof.* The Γ-Partition Lemma is an instance of the Sequential Execution Lemma (Lemma 5.1.3). In particular, an $\mathcal{A}$-CT $\mathfrak{T}$ is an RT for the underlying ADP rewrite relation ($\hookrightarrow_{\mathcal{A}}$, or $\overset{\mathtt{i}}{\hookrightarrow}_{\mathcal{A}}$ for innermost chain trees), and by assumption we have $|\mathfrak{T}| < 1$. Every inner node of $\mathfrak{T}$ performs either an annotated rewrite step (Case **at** or **af**, i.e., a node in $\Gamma^{\mathfrak{T}}$) or a non-annotated one (a node in $\overline{\Gamma}^{\mathfrak{T}}$), so the inner nodes of $\mathfrak{T}$ are partitioned into $\Gamma^{\mathfrak{T}} \uplus \overline{\Gamma}^{\mathfrak{T}}$. Using $\Gamma^{\mathfrak{T}} = \Gamma_1 \uplus \Gamma_2$, we set $N_1 = \overline{\Gamma}^{\mathfrak{T}} \cup \Gamma_1$ and $N_2 = \Gamma_2$, which partitions the inner nodes of $\mathfrak{T}$ into $N_1 \uplus N_2$. By assumption, every subtree of $\mathfrak{T}$ that only contains inner nodes from $N_1 = \overline{\Gamma}^{\mathfrak{T}} \cup \Gamma_1$ converges with probability 1, so the prerequisites of the Sequential Execution Lemma are satisfied. We thus obtain a subtree $\mathfrak{T}'$ of $\mathfrak{T}$ with $|\mathfrak{T}'| < 1$ such that every infinite path of $\mathfrak{T}'$ contains an infinite number of nodes from $N_2 = \Gamma_2$. This subtree is grounded, as the subtree constructed in the proof of Lemma 5.1.3 keeps the root of $\mathfrak{T}$. *QED*

The second crucial lemma is the *Starting Lemma*. It shows that w.l.o.g., we can assume that the root of a CT is labeled with $t$ for a term $t \in \mathcal{T}^{\sharp}$ with $\mathrm{pos}_{\mathcal{D}^{\sharp}}(t) = \{\pi\}$ for some position $\pi$ and $\flat(t|_\pi) = s\theta$ for a substitution $\theta$ and an ADP $s \overset{m}{\longrightarrow} \mu \in \mathcal{A}$, and that the root of the CT performs its step with this ADP at the position $\pi$. For innermost rewriting, we can even assume $\pi = \varepsilon$, so that $t$ is an instantiated left-hand side where only the root is annotated. For full rewriting, this stronger form does not hold (see Counterexample 7.3.4 below). The proof for this lemma requires the above Γ-Partition Lemma. This can be seen analogously to the witness theorems for PTRSs (Theorem 5.3.2 and Theorem 5.3.3), adjusted to ADPs.

Lemma 7.3.3 – Starting Lemma.
*If an ADP problem $\mathcal{A}$ is not AST, then there exists an $\mathcal{A}$-CT $\mathfrak{T}$ with $|\mathfrak{T}| < 1$ that starts with $t$ for a term $t \in \mathcal{T}^{\sharp}$ where $\mathrm{pos}_{\mathcal{D}^{\sharp}}(t) = \{\pi\}$ for some position $\pi$ and $\flat(t|_\pi) = s\theta$ for a substitution $\theta$ and an ADP $s \overset{m}{\longrightarrow} \mu \in \mathcal{A}$, such that the root of $\mathfrak{T}$ performs a step with this ADP at the position $\pi$. If $\mathtt{AST}(\overset{\mathtt{i}}{\hookrightarrow}_{\mathcal{A}})$ does not hold, then there exists such an innermost $\mathcal{A}$-CT $\mathfrak{T}$ with $\pi = \varepsilon$, where $\flat(t) \in \mathtt{ANF}_{\mathcal{A}}$.*

*Proof.* Assume that every $\mathcal{A}$-CT $\mathfrak{T}$ converges with probability 1 if it starts with $t$ where $\mathrm{pos}_{\mathcal{D}^{\sharp}}(t) = \{\pi\}$ for some position $\pi$ and the root of $\mathfrak{T}$ performs its step at the position $\pi$. We now prove that then also every $\mathcal{A}$-CT $\mathfrak{T} = (N, E)$ that starts with $t$ for some arbitrary annotated term $t$ converges with probability 1, and thus $\mathcal{A}$ is AST. We prove the claim by induction on the number of annotations in the initial term $t$. *Proof*

If $t$ contains no annotation, then $\mathfrak{T}$ is trivially finite (it cannot contain an infinite path, since there are no nodes in $\Gamma$) and hence, it converges with probability 1. Next, let $t$ contain exactly one annotation, and let $F$ be the set of all nodes $v \in \Gamma$ such that no proper ancestor of $v$ is in $\Gamma$. Before a node from $F$ is reached, all steps are of the form **nt** or **nf**, so they do not create annotations, and they cannot duplicate annotations, since we use VRFs and not GVRFs. The nodes that do not have a proper ancestor in $\Gamma$ form a finitely branching tree without infinite paths (due to the liveness property), which is finite by KÖNIG's Lemma. Thus, $|\mathfrak{T}|$ is the sum of the probabilities of the leaves of $\mathfrak{T}$ in this finite tree and of the convergence probabilities $|\mathfrak{T}_v| \cdot p_v$ of the subtrees $\mathfrak{T}_v$ at the nodes $v \in F$, where the probabilities of these leaves and nodes add up to 1. For every $v \in F$, the term $t_v$ contains exactly one annotation (at least one, since $v \in \Gamma$), and $v$ performs its step at the position of this annotation. So by our assumption, we have $|\mathfrak{T}_v| = 1$ for all $v \in F$, and hence $|\mathfrak{T}| = 1$.

Next, we regard the induction step, and assume for a contradiction that for a term $t$ with $n > 1$ annotations, there is a CT $\mathfrak{T} = (N, E)$ that converges with probability $< 1$. Here, our induction hypothesis is that every $\mathcal{A}$-CT $\mathfrak{T}$ that starts with $t'$, where $t'$ contains $m$ annotations for some $1 \leq m < n$, converges with probability 1. Let $\Pi_1 = \{\tau\}$ and $\Pi_2 = \{\chi \in \mathrm{pos}_{\mathcal{D}^\sharp}(t) \mid \chi \neq \tau\}$ for some $\tau \in \mathrm{pos}_{\mathcal{D}^\sharp}(t)$ and consider the two terms $\sharp_{\Pi_1}(t)$ and $\sharp_{\Pi_2}(t)$, which both contain strictly fewer than $n$ annotations. By our induction hypothesis, we know that every $\mathcal{A}$-CT that starts with $\sharp_{\Pi_1}(t)$ or $\sharp_{\Pi_2}(t)$ converges with probability 1. Let $\mathfrak{T}_1 = (N, E)$ be the tree that starts with $\sharp_{\Pi_1}(t)$ and uses the same ADPs, the same positions, and the same VRFs as in $\mathfrak{T}$. Analogously, let $\mathfrak{T}_2$ be the tree that starts with $\sharp_{\Pi_2}(t)$ and uses the same ADPs, positions, and VRFs as $\mathfrak{T}$. For every node $v$, let $t_v$, $t_v^{(1)}$, and $t_v^{(2)}$ be its terms in $\mathfrak{T}$, $\mathfrak{T}_1$, and $\mathfrak{T}_2$. By induction on the depth of $v$, we have $t_v \overset{\flat}{=} t_v^{(1)} \overset{\flat}{=} t_v^{(2)}$ and $\mathrm{pos}_{\mathcal{D}^\sharp}(t_v) = \mathrm{pos}_{\mathcal{D}^\sharp}(t_v^{(1)}) \cup \mathrm{pos}_{\mathcal{D}^\sharp}(t_v^{(2)})$, since the set $\Psi_j$ in Definition 7.2.11 is the union of the contributions of the single annotated positions of the redex, and the annotations of the right-hand side are created iff the position of the redex is annotated.

We can partition the $\Gamma^{\mathfrak{T}}$-nodes (i.e., the $\Gamma$-nodes of $\mathfrak{T}$) into $\Gamma_1^{\mathfrak{T}} = \Gamma^{\mathfrak{T}_1}$ and $\Gamma_2^{\mathfrak{T}} = \Gamma^{\mathfrak{T}} \setminus \Gamma_1^{\mathfrak{T}}$ (i.e., the $\Gamma$-nodes of $\mathfrak{T}_1$, and the rest). Note that $\mathfrak{T}_1$ itself may not be an $\mathcal{A}$-CT again, since there might exist paths without an infinite number of $\Gamma^{\mathfrak{T}_1}$-nodes, but obviously every subtree $\mathfrak{T}_1'$ of $\mathfrak{T}_1$ such that every infinite path has an infinite number of $\Gamma^{\mathfrak{T}_1}$-nodes is an $\mathcal{A}$-CT again. Moreover, by extending such a subtree to be grounded, i.e., adding the initial path from the root of $\mathfrak{T}_1$ to the root of $\mathfrak{T}_1'$ together with all siblings of the nodes on this path as leaves, we created an $\mathcal{A}$-CT that starts with $\sharp_{\Pi_1}(t)$, and hence by our induction hypothesis, converges with probability 1. Thus, this also holds for $\mathfrak{T}_1'$.

We want to use the $\Gamma$-Partition Lemma (Lemma 7.3.2) for the tree $\mathfrak{T}$. For this, we have to show that every subtree $\mathfrak{T}_1'$ of $\mathfrak{T}$ that only contains inner nodes from $\overline{\Gamma}^{\mathfrak{T}} \cup \Gamma_1^{\mathfrak{T}}$ converges with probability 1. But since $\mathfrak{T}_1'$ only contains inner nodes from $\overline{\Gamma}^{\mathfrak{T}} \cup \Gamma_1^{\mathfrak{T}}$ it must either contain infinitely many $\Gamma_1^{\mathfrak{T}}$-nodes (and by the previous paragraph it converges with probability 1), or it contains only finitely many $\Gamma_1^{\mathfrak{T}}$-nodes, hence must be finite itself, and converges with probability 1.

We have shown that the conditions for the $\Gamma$-Partition Lemma (Lemma 7.3.2) are satisfied. Thus, we can apply the $\Gamma$-Partition Lemma to obtain a grounded subtree $\mathfrak{T}'$ of $\mathfrak{T}$ with $|\mathfrak{T}'| < 1$ such that on every infinite path, we have an infinite number of $\Gamma_2^{\mathfrak{T}}$ nodes. Now consider the restriction of $\mathfrak{T}_2$ to the nodes of $\mathfrak{T}'$. Again, all local properties for an $\mathcal{A}$-CT are satisfied. Additionally, this time we know that every infinite path has an

infinite number of nodes from $\Gamma_2^{\mathfrak{T}}$ in $\mathfrak{T}'$, and since $\mathrm{pos}_{\mathcal{D}^\sharp}(t_v) = \mathrm{pos}_{\mathcal{D}^\sharp}(t_v^{(1)}) \cup \mathrm{pos}_{\mathcal{D}^\sharp}(t_v^{(2)})$, every node from $\Gamma_2^{\mathfrak{T}} = \Gamma^{\mathfrak{T}} \setminus \Gamma^{\mathfrak{T}_1}$ is also in $\Gamma^{\mathfrak{T}_2}$. Hence, we also know that the liveness property is satisfied. This means that this restriction of $\mathfrak{T}_2$ is an $\mathcal{A}$-CT that starts with $\sharp_{\Pi_2}(t)$ and converges with probability $|\mathfrak{T}'| < 1$. This is our desired contradiction, which proves the induction step.

So we have proven that we can restrict ourselves to CTs that start with $t$ for a term $t \in \mathcal{T}^\sharp$ where $\mathrm{pos}_{\mathcal{D}^\sharp}(t) = \{\pi\}$ for some position $\pi$ and whose root performs its step at the position $\pi$ when analyzing $\mathtt{AST}(\hookrightarrow_{\mathcal{A}})$ and $\mathtt{AST}(\overset{\mathrm{i}}{\hookrightarrow}_{\mathcal{A}})$, since the trees constructed in this proof perform the same rewrite steps on terms with the same flattened subterms, so that they are innermost if $\mathfrak{T}$ is innermost. The step at the root of such a CT is of the form **at** or **af** with some ADP $s \overset{m}{\hookrightarrow} \mu \in \mathcal{A}$, i.e., $\flat(t|_\pi) = s\theta$ for a substitution $\theta$, where $\flat(t|_\pi) = s\theta \in \mathtt{ANF}_{\mathcal{A}}$ for innermost CTs.

It remains to show that for $\mathtt{AST}(\overset{\mathrm{i}}{\hookrightarrow}_{\mathcal{A}})$, we can additionally assume $\pi = \varepsilon$. So let $\mathfrak{T}$ be an innermost $\mathcal{A}$-CT with $|\mathfrak{T}| < 1$ that starts with $t$ where $\mathrm{pos}_{\mathcal{D}^\sharp}(t) = \{\pi\}$, and whose root performs its step at the position $\pi$. Then all annotated positions of every term $t_v$ in $\mathfrak{T}$ are at or below $\pi$: a step at or below $\pi$ only creates annotations at or below its position, a step at a position orthogonal to $\pi$ is of the form **nt** or **nf** and creates no annotations, and a step at a position strictly above $\pi$ is of the form **nt** or **nf** and removes all annotations below its position, since innermost rewrite steps do not keep annotations below the redex. Hence, if a node $v$ performs a step at a position strictly above $\pi$, then the terms of its children contain no annotations, so that the subtree $\mathfrak{T}_v$ contains no nodes from $\Gamma$ and is finite by the liveness property and König's Lemma. For every node $v$ we have $|\mathfrak{T}_v| = \sum_{w \in vE} {}^{p_w}\!/_{p_v} \cdot |\mathfrak{T}_w|$ for every inner node $v$, and $|\mathfrak{T}_v| = 1$ if $v$ is a leaf or performs a step at a position strictly above $\pi$. Moreover, for every node $v$, let $\nu(v)$ be the node that we reach from $v$ if we repeatedly go from a node that performs a step at a position orthogonal to $\pi$ to one of its children $w$ with minimal $|\mathfrak{T}_w|$. This process terminates, since an infinite path that only performs steps at positions orthogonal to $\pi$ would not contain any nodes from $\Gamma$. We have $|\mathfrak{T}_{\nu(v)}| \leq |\mathfrak{T}_v|$, and $t_{\nu(v)}|_\pi = t_v|_\pi$, since steps at positions orthogonal to $\pi$ do not change the subterm at position $\pi$.

Now we construct an innermost $\mathcal{A}$-CT $\mathfrak{T}'$ that starts with $t|_\pi$, whose nodes are nodes of $\mathfrak{T}$. The root of $\mathfrak{T}'$ is the root of $\mathfrak{T}$, and each node $v$ of $\mathfrak{T}'$ is labeled with ${}^{p'_v}t_v|_\pi$ for a probability $p'_v$, where $p'_\mathfrak{t} = 1$. If $v$ is a leaf of $\mathfrak{T}$ or performs a step at a position strictly above $\pi$, then $v$ is a leaf of $\mathfrak{T}'$. Otherwise, $v$ performs a step at a position $\pi.\kappa$, and its children in $\mathfrak{T}'$ are the nodes $\nu(w)$ for its children $w \in vE = \{w_1, \ldots, w_h\}$ in $\mathfrak{T}$, with $p'_{\nu(w)} = p'_v \cdot {}^{p_w}\!/_{p_v}$. Since all annotations of $t_v$ are at or below $\pi$, we have $t_v|_\pi \overset{\mathrm{i}}{\hookrightarrow}_{\mathcal{A}} \{{}^{p_{w_1}/p_v}t_{w_1}|_\pi, \ldots, {}^{p_{w_h}/p_v}t_{w_h}|_\pi\}$ with the same ADP, the same case, and the same substitution at the position $\kappa$, and $t_w|_\pi = t_{\nu(w)}|_\pi$ for all $w \in vE$. In particular, $v$ is a node from $\Gamma$ in $\mathfrak{T}'$ iff it is a node from $\Gamma$ in $\mathfrak{T}$. Every infinite path of $\mathfrak{T}'$ results from an infinite path of $\mathfrak{T}$ by omitting finitely many nodes between any two of its nodes, which perform steps at positions orthogonal to $\pi$ and are thus not in $\Gamma$. Hence, $\mathfrak{T}'$ satisfies the liveness property, so it is an innermost $\mathcal{A}$-CT that starts with $t|_\pi$, where $\mathrm{pos}_{\mathcal{D}^\sharp}(t|_\pi) = \{\varepsilon\}$ and the root performs its step at the position $\varepsilon$. Finally, for every inner node $v$ of $\mathfrak{T}'$ we have $\sum_{w \in vE} p'_{\nu(w)} \cdot |\mathfrak{T}_{\nu(w)}| \leq p'_v \cdot \sum_{w \in vE} {}^{p_w}\!/_{p_v} \cdot |\mathfrak{T}_w| = p'_v \cdot |\mathfrak{T}_v|$, and every leaf $v$ of $\mathfrak{T}'$ has $|\mathfrak{T}_v| = 1$. So by induction on $d \in \mathbb{N}$, the sum of $p'_v$ for all leaves $v$ of $\mathfrak{T}'$ up to depth $d$ plus the sum of $p'_v \cdot |\mathfrak{T}_v|$ for all inner nodes $v$ of $\mathfrak{T}'$ at depth $d$ is at most $|\mathfrak{T}|$. Hence, $|\mathfrak{T}'| \leq |\mathfrak{T}| < 1$. *QED*

Counterexample 7.3.4. For full rewriting, we cannot assume $\pi = \varepsilon$ in the Starting Lemma (Lemma 7.3.3), since a step above the annotation may merge two subterms

whose annotations complement each other. Let $\mathsf{e}$ be a constant and $\mathsf{p}$ be a binary constructor, let $u = \mathsf{p}(\mathsf{a}^\sharp(\mathsf{c}), \mathsf{a}(\mathsf{c}^\sharp))$, and consider the ADP problem $\mathcal{A}$ with the ADPs

$$\mathsf{c} \overset{\mathsf{t}}{\hookrightarrow} \{{}^{1/2}\mathsf{e},\ {}^{1/2}\mathsf{h}^\sharp\} \qquad \mathsf{h} \overset{\mathsf{t}}{\hookrightarrow} \{{}^{1}u\} \qquad \mathsf{g}(\mathsf{p}(x,x)) \overset{\mathsf{t}}{\hookrightarrow} \{{}^{1}\mathsf{g}(x)\}$$
$$\mathsf{a}(\mathsf{e}) \overset{\mathsf{t}}{\hookrightarrow} \{{}^{1}u\} \qquad \mathsf{a}(x) \overset{\mathsf{t}}{\hookrightarrow} \{{}^{1}x\}.$$

$\mathcal{A}$ is not AST: In the term $\mathsf{g}(\mathsf{h}^\sharp)$, we rewrite $\mathsf{h}^\sharp$ with an **at**-step and obtain $\mathsf{g}(u) = \mathsf{g}(\mathsf{p}(\mathsf{a}^\sharp(\mathsf{c}), \mathsf{a}(\mathsf{c}^\sharp)))$. Then we apply the g-ADP at the root with the VRF that maps both positions of $x$ in its left-hand side to the position of $x$ in its right-hand side. This **nt**-step merges the annotations of the two copies of $\mathsf{a}(\mathsf{c})$ and yields $\mathsf{g}(\mathsf{a}^\sharp(\mathsf{c}^\sharp))$. Next, the c-ADP yields $\mathsf{g}(\mathsf{a}^\sharp(\mathsf{e}))$ and $\mathsf{g}(\mathsf{a}^\sharp(\mathsf{h}^\sharp))$ with probability $1/2$ each. From $\mathsf{g}(\mathsf{a}^\sharp(\mathsf{e}))$, an **at**-step with $\mathsf{a}(\mathsf{e}) \overset{\mathsf{t}}{\hookrightarrow} \{{}^{1}u\}$ yields $\mathsf{g}(u)$. From $\mathsf{g}(\mathsf{a}^\sharp(\mathsf{h}^\sharp))$, we rewrite $\mathsf{h}^\sharp$ to obtain $\mathsf{g}(\mathsf{a}^\sharp(u))$, and an **at**-step with $\mathsf{a}(x) \overset{\mathsf{t}}{\hookrightarrow} \{{}^{1}x\}$, whose VRF keeps the annotations of $x$, yields $\mathsf{g}(u)$ as well. Repeating these steps leads to an $\mathcal{A}$-CT without leaves where every infinite path contains infinitely many nodes from $\Gamma$, i.e., to an $\mathcal{A}$-CT that converges with probability 0.

However, every $\mathcal{A}$-CT $\mathfrak{T}$ that starts with a term $t$ with $\mathrm{pos}_{\mathcal{D}^\sharp}(t) = \{\varepsilon\}$ converges with probability 1. Before the first step at the annotated root, $\mathfrak{T}$ only performs steps of the form **nt** strictly below the root, which do not change the annotations, so as in the proof of Lemma 7.3.3, this part of $\mathfrak{T}$ is finite by the liveness property and König's Lemma. A step at the annotated root with the g-ADP or with $\mathsf{a}(x) \overset{\mathsf{t}}{\hookrightarrow} \{{}^{1}x\}$ yields a term without annotations, and then the remaining subtree is finite. Otherwise, it yields $\mathsf{e}$, $\mathsf{h}^\sharp$, or $u$. Then all further terms are ground and do not contain $\mathsf{g}$, so the merging g-ADP is never applied again, and since the VRFs of all other ADPs are injective, no annotations are merged. New annotations are only created in copies of $u$, each of which contains one $\mathsf{c}^\sharp$ and one $\mathsf{a}^\sharp$ with the argument $\mathsf{c}$. Such a $\mathsf{c}^\sharp$ leads to a new copy of $u$ only if the c-ADP yields $\mathsf{h}^\sharp$, and such an $\mathsf{a}^\sharp$ only if its argument $\mathsf{c}$ is rewritten to $\mathsf{e}$, which happens with probability $1/2$ in both cases; otherwise, the annotation disappears without creating new ones. Hence, such a run behaves like a symmetric random walk, and we have $|\mathfrak{T}| = 1$.

As a first and particularly simple processor, we show how to remove ADPs that are irrelevant for the termination analysis. Recall that the annotations on the right-hand sides of an ADP mark the subterms that may start a non-AST evaluation, and that an ADP with flag $\mathsf{f}$ is never used to rewrite below an annotation while keeping that annotation. Hence, for innermost rewriting, an ADP $\ell \overset{\mathsf{f}}{\hookrightarrow} \mu$ with flag $\mathsf{f}$ whose right-hand sides contain no annotations can never contribute to a non-iAST evaluation: its right-hand sides introduce no annotations, due to its flag $\mathsf{f}$ it erases the annotations above the rewrite position instead of keeping them, and every innermost rewrite step removes the annotations below the rewrite position. The only reason to keep such ADPs in an ADP problem is to preserve the set of normal forms, so that the innermost evaluation strategy is not affected. Therefore, we replace each such ADP by an ADP with the same left-hand side and a trivial right-hand side.

Theorem 7.3.5 – Removing Irrelevant ADPs Processor for iAST.

Let $\mathcal{A}$ be an ADP problem and let

$$\mathit{Irr}(\mathcal{A}) = \{\ell \overset{\mathsf{f}}{\hookrightarrow} \mu \in \mathcal{A} \mid \mathrm{pos}_{\mathcal{D}^\sharp}(r) = \varnothing \text{ for all } r \in \mathrm{Supp}(\mu)\}$$

be the set of all ADPs with flag $\mathsf{f}$ and without any annotations. Let $\mathrm{Proc}^{\mathsf{i}}_{\mathsf{irr}}$ be the processor that replaces every $\ell \overset{\mathsf{f}}{\hookrightarrow} \mu \in \mathit{Irr}(\mathcal{A})$ by $\ell \overset{\mathsf{f}}{\hookrightarrow} \{{}^{1}\mathsf{stop}\}$ for a fresh constant

stop, i.e.,

$$\mathrm{Proc}^{\mathbf{i}}_{\mathtt{irr}}(\mathcal{A}) = \{(\mathcal{A} \setminus \mathit{Irr}(\mathcal{A})) \cup \{\ell \xhookrightarrow{\mathsf{f}} \{^1\mathsf{stop}\} \mid \ell \xhookrightarrow{\mathsf{f}} \mu \in \mathit{Irr}(\mathcal{A})\}\}.$$

Then $\mathrm{Proc}^{\mathbf{i}}_{\mathtt{irr}}$ is sound and complete for $\mathtt{iAST}$.

Here, we add the fresh constant stop to the set $\mathcal{C}$ of constructors of the (otherwise fixed) signature. This does not affect $\mathtt{AST}(\xhookrightarrow{\mathbf{i}}_{\mathcal{A}})$, since no ADP of $\mathcal{A}$ contains stop, so that one can replace stop by a fresh variable in every innermost $\mathcal{A}$-CT.

*Proof.* Let $\mathcal{A}''$ be the ADP problem with $\mathrm{Proc}^{\mathbf{i}}_{\mathtt{irr}}(\mathcal{A}) = \{\mathcal{A}''\}$. Replacing each $\ell \xhookrightarrow{\mathsf{f}} \mu \in \mathit{Irr}(\mathcal{A})$ by $\ell \xhookrightarrow{\mathsf{f}} \{^1\mathsf{stop}\}$ keeps $\ell$ as a left-hand side, hence preserves the set of normal forms and thus the innermost evaluation strategy. Moreover, $\mathit{Irr}(\mathcal{A}'')$ consists of the new ADPs with the right-hand side $\{^1\mathsf{stop}\}$, so we have $\mathcal{A}'' \setminus \mathit{Irr}(\mathcal{A}'') = \mathcal{A} \setminus \mathit{Irr}(\mathcal{A})$ and $\mathtt{ANF}_{\mathcal{A}''} = \mathtt{ANF}_{\mathcal{A}}$. We show the following claim for all ADP problems $\mathcal{A}_1$ and $\mathcal{A}_2$ with $\mathcal{A}_1 \setminus \mathit{Irr}(\mathcal{A}_1) \subseteq \mathcal{A}_2$ and $\mathtt{ANF}_{\mathcal{A}_1} \subseteq \mathtt{ANF}_{\mathcal{A}_2}$:

$$\text{For every innermost } \mathcal{A}_1\text{-CT } \mathfrak{T} \text{ there is an innermost } \mathcal{A}_2\text{-CT } \mathfrak{T}' \text{ with } |\mathfrak{T}'| \leq |\mathfrak{T}|. \tag{7.27}$$

Then (7.27) for $\mathcal{A}_1 = \mathcal{A}$ and $\mathcal{A}_2 = \mathcal{A}''$ (for $\mathcal{A}_1 = \mathcal{A}''$ and $\mathcal{A}_2 = \mathcal{A}$) implies that $\mathrm{Proc}^{\mathbf{i}}_{\mathtt{irr}}$ is sound (complete) for $\mathtt{iAST}$.

To prove (7.27), let $\mathfrak{T}$ be an innermost $\mathcal{A}_1$-CT. The idea is that $\mathfrak{T}'$ performs the same steps as $\mathfrak{T}$, except for the steps with ADPs from $\mathit{Irr}(\mathcal{A}_1)$ and the steps at positions where annotations cannot occur anymore due to such a step. For two terms $t, t' \in \mathcal{T}^\sharp$ and a set $H$ of pairwise orthogonal positions of both $t$ and $t'$, we write $t \approx_H t'$ if

(a) replacing the subterms at all positions from $H$ by a fresh variable $y$ in $t$ and in $t'$ yields two terms $\overline{t}$ and $\overline{t'}$ with $\overline{t} \overset{\flat}{=} \overline{t'}$,

(b) $t$ has no annotation at a position that is above, at, or below a position from $H$, and

(c) every position that is orthogonal to all positions from $H$ is annotated in $t$ iff it is annotated in $t'$.

So $t \approx_\varnothing t'$ holds iff $t = t'$. We assign a term $t'_v$ and a set $H_v$ with $t_v \approx_{H_v} t'_v$ to every node $v$ of $\mathfrak{T}$, where $t'_{\mathfrak{r}} = t_{\mathfrak{r}}$ and $H_{\mathfrak{r}} = \varnothing$ for the root $\mathfrak{r}$. Let $v$ be an inner node with $vE = \{w_1, \ldots, w_k\}$ that performs its step with the ADP $\ell \xhookrightarrow{m} \{^{p_1}r_1, \ldots, ^{p_k}r_k\} \in \mathcal{A}_1$, the position $\pi$, and the substitution $\sigma$, i.e., $\flat(t_v|_\pi) = \ell\sigma \in \mathtt{ANF}_{\mathcal{A}_1}$. We call $v$ *mirrored* if this ADP is not in $\mathit{Irr}(\mathcal{A}_1)$ and $\pi$ is orthogonal to all positions from $H_v$, and *skipped* otherwise.

- If $v$ is mirrored, then we have $t_v|_\pi = t'_v|_\pi$ by (a) and (c), the ADP is contained in $\mathcal{A}_2$, and $\ell\sigma \in \mathtt{ANF}_{\mathcal{A}_2}$. So we have $t'_v \xhookrightarrow{\mathbf{i}}_{\mathcal{A}_2} \{^{p_1}t'_{w_1}, \ldots, ^{p_k}t'_{w_k}\}$ with the same ADP, the same case, and the same substitution at the position $\pi$, and we define $H_{w_j} = H_v$. Then $t_{w_j} \approx_{H_{w_j}} t'_{w_j}$ holds, as both steps replace the subterm at the position $\pi$ by the same term, and if $m = \mathsf{f}$, then both remove the annotations strictly above $\pi$.
- If $v$ is skipped, then we define $t'_{w_j} = t'_v$ for all $1 \leq j \leq k$. If $\pi$ is at or below a position from $H_v$, then $\pi \notin \mathrm{pos}_{\mathcal{D}^\sharp}(t_v)$ and $t_v$ has no annotations above $\pi$ by (b), so the step only replaces the subterm at the position $\pi$ by a term without annotations, and we define $H_{w_j} = H_v$. If $\pi$ is strictly above a position from $H_v$,

then $\pi \notin \mathrm{pos}_{\mathcal{D}^\sharp}(t_v)$ and $t_v$ has no annotations above $\pi$ by (b), so $t_{w_j}$ has no annotations above, at, or below $\pi$, and we define $H_{w_j} = \{\pi\} \cup \{\tau \in H_v \mid \tau \perp \pi\}$. Otherwise, the ADP is in $Irr(\mathcal{A}_1)$ and $\pi$ is orthogonal to all positions from $H_v$. Then we have $m = \mathsf{f}$ and $r_j$ contains no annotations, so again $t_{w_j}$ has no annotations above, at, or below $\pi$, and we define $H_{w_j} = H_v \cup \{\pi\}$. In all three cases, we obtain $t_{w_j} \approx_{H_{w_j}} t'_{w_j}$, since innermost rewrite steps do not keep annotations below the redex.

Every infinite path of $\mathfrak{T}$ contains infinitely many mirrored nodes from $\Gamma$. To see this, note that by (b), every skipped node from $\Gamma$ performs its step with an ADP from $Irr(\mathcal{A}_1)$. Such a step removes the annotation at its position and creates no annotations, and the steps of the nodes from $\overline{\Gamma}$ do not create annotations either. So if an infinite path contained only finitely many mirrored nodes from $\Gamma$, then after the last of them, the number of annotations would never increase and it would decrease with every further node from $\Gamma$. Thus, the path would contain only finitely many nodes from $\Gamma$, which contradicts the liveness property.
As in the proof of Lemma 7.3.3, for every node $v$ let $\nu(v)$ be the node that we reach from $v$ if we repeatedly go from a skipped node to one of its children $w$ with minimal $|\mathfrak{T}_w|$. This process terminates, since no infinite path consists only of skipped nodes, and we have $|\mathfrak{T}_{\nu(v)}| \leq |\mathfrak{T}_v|$ and $t'_{\nu(v)} = t'_v$. Now the nodes of $\mathfrak{T}'$ are nodes of $\mathfrak{T}$, where each node $v$ of $\mathfrak{T}'$ is labeled with ${}^{p'_v}t'_v$ for a probability $p'_v$. The root of $\mathfrak{T}'$ is $\nu(\mathfrak{r})$ with $p'_{\nu(\mathfrak{r})} = 1$. If a node $v$ of $\mathfrak{T}'$ is a leaf of $\mathfrak{T}$, then it is a leaf of $\mathfrak{T}'$. Otherwise, $v$ is mirrored, and its children in $\mathfrak{T}'$ are the nodes $\nu(w)$ for its children $w \in vE$ in $\mathfrak{T}$, with $p'_{\nu(w)} = p'_v \cdot p_w/p_v$. Due to $t'_{\nu(w)} = t'_w$, the edges of $\mathfrak{T}'$ represent the innermost rewrite steps with $\mathcal{A}_2$ from above, and $v$ is a node from $\Gamma$ in $\mathfrak{T}'$ iff it is a node from $\Gamma$ in $\mathfrak{T}$. Every infinite path of $\mathfrak{T}'$ results from an infinite path of $\mathfrak{T}$ by omitting skipped nodes, so it contains infinitely many nodes from $\Gamma$. Hence, $\mathfrak{T}'$ is an innermost $\mathcal{A}_2$-CT. Finally, for every inner node $v$ of $\mathfrak{T}'$ we have $\sum_{w \in vE} p'_{\nu(w)} \cdot |\mathfrak{T}_{\nu(w)}| \leq p'_v \cdot \sum_{w \in vE} p_w/p_v \cdot |\mathfrak{T}_w| = p'_v \cdot |\mathfrak{T}_v|$, and every leaf $v$ of $\mathfrak{T}'$ has $|\mathfrak{T}_v| = 1$. So by induction on $d \in \mathbb{N}$, the sum of $p'_v$ for all leaves $v$ of $\mathfrak{T}'$ up to depth $d$ plus the sum of $p'_v \cdot |\mathfrak{T}_v|$ for all inner nodes $v$ of $\mathfrak{T}'$ at depth $d$ is at most $|\mathfrak{T}_{\nu(\mathfrak{r})}| \leq |\mathfrak{T}|$. Hence, $|\mathfrak{T}'| \leq |\mathfrak{T}|$. *QED*

We will see applications of this processor all over the remainder of this chapter.

For full rewriting, however, the ADPs in $Irr(\mathcal{A})$ need not be irrelevant: the VRF of a step with such an ADP can keep the annotations below the rewrite position, so that a step with a non-left-linear ADP from $Irr(\mathcal{A})$ can still merge the annotations of several subterms. Such a merge is the crucial step of Counterexample 7.3.4, which shows that we cannot assume $\pi = \varepsilon$ in the Starting Lemma for full rewriting, and the merging ADP of this counterexample becomes irrelevant if we change its flag to $\mathsf{f}$. Hence, $\mathrm{Proc}^{\mathsf{i}}_{\mathtt{irr}}$ is unsound for AST, as the following counterexample shows.

Counterexample 7.3.6. Reconsider the ADP problem $\mathcal{A}$ from Counterexample 7.3.4 and let $\mathcal{A}'$ result from $\mathcal{A}$ by changing the flag of the $\mathsf{g}$-ADP to $\mathsf{f}$, so that $Irr(\mathcal{A}') = \{\mathsf{g}(\mathsf{p}(x,x)) \overset{\mathsf{f}}{\hookrightarrow} \{{}^1\mathsf{g}(x)\}\}$. Then $\mathcal{A}'$ is not AST: The $\mathcal{A}$-CT from Counterexample 7.3.4 that converges with probability 0 only applies the $\mathsf{g}$-ADP at the root of the term, where there are no annotations above the redex. So this tree is also an $\mathcal{A}'$-CT, where the steps with the $\mathsf{g}$-ADP are **nf**-steps that still merge the annotations of the two copies of $\mathsf{a}(\mathsf{c})$.

However, $\mathrm{Proc}^{\mathsf{i}}_{\mathtt{irr}}(\mathcal{A}') = \{\mathcal{A}''\}$ with $\mathcal{A}'' = (\mathcal{A}' \setminus Irr(\mathcal{A}')) \cup \{\mathsf{g}(\mathsf{p}(x,x)) \overset{\mathsf{f}}{\hookrightarrow} \{{}^1\mathsf{stop}\}\}$ is

$\mathtt{AST}$, because the $\mathtt{g}$-rule cannot be used to merge annotations anymore, which would be needed for a CT that converges with probability $< 1$.

The processor to remove irrelevant ADPs is especially useful in combination with the usable rules processor (Section 7.4.3), which turns the flags of all non-usable ADPs to $\mathtt{f}$ and thereby often produces irrelevant ADPs, and as a preprocessing step before switching back to PTRSs (Section 7.4.6). As both of these processors are only introduced later in this chapter, the real power of removing irrelevant ADPs can only be understood once these other processors are known. For example, Example 7.4.19 shows how $\mathrm{Proc}^{\mathtt{i}}_{\mathtt{irr}}$ is used together with the usable rules and probability removal processors to prove $\mathtt{AST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}_{\mathtt{ag}}})$.

In the remainder of this chapter, we introduce the remaining processors of the ADP framework. However, due to the modularity of the framework itself, one can easily extend it by developing new processors in the future. We first present *structural* processors that split the problem based on the connection between the different defined symbols (Section 7.4). Afterwards, we present processors based on algebras (Section 7.5), similar to the technique that searches for a single ranking function on the whole system. Note that one can also use these algebraic processors after having already applied the structural processors. Finally, we present *transformational* processors that try to transform existing rules into new ones that are easier to analyze (Section 7.6).

SECTION 7.4

## Structural Processors

In this section, we introduce processors based on structural properties of the ADP problem. For example, we can analyze which function calls can generate which other function calls, yielding a control flow graph that can be used to split the problem into simpler sub-problems, similar to the modularity results of Section 5.4. Instead of costly calls to an SMT solver or SAT solver, most of the processors in this section are quite fast to execute and only rely on simple graph algorithms, and thus, are used first in our implementation. In total, we consider five processors and one additional idea.

1. The *dependency graph processor* builds a control flow graph of which ADPs can lead to applications of which others at an annotated position, so that the problem can be decomposed into its strongly connected components and these can be analyzed separately.
2. The *usable terms processor* removes the annotation from individual subterms in right-hand sides whose instances can never rewrite to an instance of a left-hand side of an ADP with annotations, i.e., subterms that can never lead to a rewrite step at an annotation.
3. The *usable rules processor* sets the flag to $\mathtt{f}$ for rules that can never be applied below an annotation when their variables are instantiated with normal forms, making the context of such annotations irrelevant.
4. The *probability removal processor* lets us switch to the ordinary non-probabilistic DP framework whenever every ADP is trivial, i.e., of the form $\ell \xhookrightarrow{m} \{^1r\}$, so that $(\mathtt{i})\mathtt{AST}$ can be proved via (innermost) termination of a non-probabilistic DP problem.

5. The idea of moving from the ADP framework for full rewriting to the one for innermost evaluation, whose processors are more powerful. However, this idea turns out to be unsound on the level of ADP problems.

6. The *switching back to PTRSs processor* takes the opposite direction and returns from an ADP problem to its underlying PTRS, in order to apply techniques such as the decision procedure for the subclass RNT.

### 7.4.1 Dependency Graph Processor

As in the non-probabilistic setting, the $\mathcal{A}$-*dependency graph* is a control flow graph whose nodes are now the ADPs from $\mathcal{A}$. Since a term can contain multiple annotations, the definition is a bit more involved than in the non-probabilistic setting. An edge of the dependency graph indicates whether an ADP $\alpha$ may lead to an application of another ADP $\alpha'$ on an annotated subterm whose annotation was introduced by $\alpha$. This possibility is not related to the probabilities. Hence, here we use the *non-probabilistic variant* $\mathrm{np}(\mathcal{A}) = \{\ell \to \flat(r_j) \mid \ell \xhookrightarrow{\mathtt{t}} \{p_1 r_1, \dots, p_k r_k\} \in \mathcal{A}, 1 \le j \le k\}$, which is an ordinary TRS over the original signature $\Sigma$. For $\mathrm{np}(\mathcal{A})$ we only consider rules with the flag $\mathtt{t}$, as only they can be used for rewriting below annotations without removing them. We define $t \trianglelefteq^\sharp s$ if there is a $\pi \in \mathrm{pos}_{\mathcal{D}^\sharp}(s)$ and $t = \flat(s|_\pi)$, i.e., $t$ results from a subterm of $s$ with annotated root symbol by removing its annotations.

**Dependency Graph Processor for `AST`** We start with the dependency graph processor for full `AST`.

> Definition 7.4.1 – Dependency Graph.
> *Let $\mathcal{A}$ be an ADP problem. The $\mathcal{A}$-dependency graph has the nodes $\mathcal{A}$ and there is an edge from $\ell_1 \xhookrightarrow{m} \{p_1 r_1, \dots, p_k r_k\}$ to $\ell_2 \xhookrightarrow{m'} \mu_2$ if there are substitutions $\sigma_1, \sigma_2$ and a $t \trianglelefteq^\sharp r_j$ for some $1 \le j \le k$ with $t^\sharp\sigma_1 \to^*_{\mathrm{np}(\mathcal{A})} \ell_2^\sharp\sigma_2$.*

So there is an edge from an ADP $\alpha$ to an ADP $\alpha'$ if after a $\hookrightarrow_{\mathcal{A}}$-step of the form **at** or **af** with $\alpha$ at position $\pi$ there may eventually come another $\hookrightarrow_{\mathcal{A}}$-step of the form **at** or **af** with $\alpha'$ at a position $\tau$ on or below $\pi$, where the root symbol of the subterm at position $\tau$ was introduced by $\alpha$. Since every infinite path in an $\mathcal{A}$-CT contains infinitely many rewrite steps with Case **at** or **af**, every such path traverses a cycle of the dependency graph infinitely often. Thus, as in the non-probabilistic setting, it suffices to consider its strongly connected components separately. In the ADP framework, this means that we remove the annotations from all ADPs except the ones in the SCC that we want to analyze.

To automate the following processor, the same over-approximation techniques as for the non-probabilistic dependency graph can be used, e.g., to estimate whether $t^\sharp\sigma_1 \to^*_{\mathrm{np}(\mathcal{A})} \ell_2^\sharp\sigma_2$ holds for some $\sigma_1, \sigma_2$, one checks whether $\mathrm{Ren}(\mathrm{Cap}_{\mathcal{A}}(t^\sharp))$ and $\ell_2^\sharp$ are unifiable. As in Section 7.1, Ren replaces all variable occurrences by fresh variables and $\mathrm{Cap}_{\mathcal{A}}(t^\sharp)$ replaces all those subterms $f(\dots)$ of $t^\sharp$ by different fresh variables where $f$ occurs on the root position of some left-hand side of $\mathcal{A}$. One could also improve $\mathrm{Cap}_{\mathcal{A}}$ by only regarding the root symbols $f$ of left-hand sides of ADPs with the flag $m = \mathtt{t}$ as "defined". In the following, we extend $\flat$ to multi-distributions, ADPs, and ADP problems by removing the annotations of all occurring

terms. So $\flat(\{{}^{p_1}r_1, \dots, {}^{p_k}r_k\}) = \{{}^{p_1}\flat(r_1), \dots, {}^{p_k}\flat(r_k)\}$, $\flat(\ell \xrightarrow{m} \mu) = \ell \xrightarrow{m} \flat(\mu)$, and $\flat(\mathcal{A}) = \{\flat(\ell \xrightarrow{m} \mu) \mid \ell \xrightarrow{m} \mu \in \mathcal{A}\}$.

**Theorem 7.4.2** – Dependency Graph Processor for AST.

Let $\mathcal{A}$ be an ADP problem, and let $\mathcal{A}_1, \dots, \mathcal{A}_n$ be the SCCs of the $\mathcal{A}$-dependency graph. Then $\mathrm{Proc}_{\mathtt{DG}}(\mathcal{A}) = \{\mathcal{A}_1 \cup \flat(\mathcal{A} \setminus \mathcal{A}_1), \dots, \mathcal{A}_n \cup \flat(\mathcal{A} \setminus \mathcal{A}_n)\}$ is sound and complete for AST.

The soundness proof below only uses that the graph contains an edge whenever the condition of Definition 7.4.1 holds. Hence, Theorem 7.4.2 (and analogously Theorem 7.4.6 below) also holds for the SCCs of every graph with the nodes $\mathcal{A}$ that contains all edges of the $\mathcal{A}$-dependency graph, e.g., for the over-approximation via Ren and $\mathrm{Cap}_{\mathcal{A}}$.

*Proof.* We prove both directions individually. Let $\overline{X} = X \cup \flat(\mathcal{A} \setminus X)$ for $X \subseteq \mathcal{A}$.

▶ *Proof(Completeness)*: By Theorem 7.3.1.

▶ *Proof(Soundness)*: Let $\mathfrak{G}$ be the $\mathcal{A}$-dependency graph. Suppose that every $\overline{\mathcal{A}_i}$-CT converges with probability 1 for all $1 \leq i \leq n$. We prove that then also every $\mathcal{A}$-CT converges with probability 1. Let $\mathfrak{W} = \{\mathcal{A}_1, \dots, \mathcal{A}_n\} \cup \{\{\alpha\} \subseteq \mathcal{A} \mid \alpha \text{ is not in an SCC of } \mathfrak{G}\}$ be the set of all SCCs and all singleton sets of nodes that do not belong to any SCC. For two $X_1, X_2 \in \mathfrak{W}$ with $X_1 \neq X_2$ we say that $X_2$ is a *direct successor* of $X_1$ (denoted $X_1 >_{\mathfrak{G}} X_2$) if there exist nodes $\alpha \in X_1$ and $\beta \in X_2$ such that there is an edge from $\alpha$ to $\beta$ in $\mathfrak{G}$. Since the elements of $\mathfrak{W}$ are the SCCs and the remaining single nodes of $\mathfrak{G}$, the relation $>_{\mathfrak{G}}$ has no cycles. The core steps of this proof are the following:

1. We show that every ADP problem $\overline{X}$ with $X \in \mathfrak{W}$ is AST.
2. We show that composing SCCs maintains the AST property.
3. We show that for every $X \in \mathfrak{W}$, the ADP problem $\overline{\bigcup_{X >^*_{\mathfrak{G}} Y} Y}$ is AST.
4. We conclude that $\mathcal{A}$ must be AST.

**Step 1** Every ADP problem $\overline{X}$ with $X \in \mathfrak{W}$ is AST

We start by proving the following:

$$\text{Every ADP problem } \overline{X} \text{ with } X \in \mathfrak{W} \text{ is AST.} \tag{7.28}$$

To prove (7.28), note that if $X$ is an SCC, then it follows from our assumption that $\overline{X}$ is AST. If $X$ is a singleton set of a node that does not belong to any SCC, then assume for a contradiction that $\overline{X}$ is not AST. By Lemma 7.3.3 there exists an $\overline{X}$-CT $\mathfrak{T} = (N, E)$ that converges with probability $< 1$ and starts with $t$ where $\mathrm{pos}_{\mathcal{D}^\sharp}(t) = \{\pi_0\}$ and $\flat(t|_{\pi_0}) = s\theta$ for a substitution $\theta$ and some ADP $\alpha = s \xrightarrow{m} \{{}^{p_1}r_1, \dots, {}^{p_k}r_k\} \in \overline{X}$, such that the root of $\mathfrak{T}$ performs a step with $\alpha$ at the position $\pi_0$. If $\alpha \notin X$, then the resulting terms after the first rewrite step contain no annotations anymore and this cannot start a CT that converges with probability $< 1$. Hence, we have $\alpha \in X$ and thus, $X = \{\alpha\}$, since $X$ is a singleton set. Assume for a contradiction that there exists a node $v \in \Gamma$ in $\mathfrak{T}$ that is not the root and introduces new annotations. W.l.o.g., let $v$ be reachable from the root without traversing any other node that introduces new annotations. This means that for the corresponding term $t_v$ for node $v$ there is a $t' \trianglelefteq^\sharp t_v$ at position $\tau$ such that $t' = s\sigma'$ for some substitution $\sigma'$ (since $\alpha$ is the only

ADP in $\overline{X}$ that contains any annotations in the right-hand side). Let $(z_0, \dots, z_m)$ with $z_m = v$ be the path from the root to $v$ in $\mathfrak{T}$. The rewrite step at the root of $\mathfrak{T}$ must be $s\theta \hookrightarrow_{\overline{X}} \{^{p_1}r_1\theta, \dots, ^{p_k}r_k\theta\}$ at the position $\pi_0$. After that, we only use ADPs with the flag $\mathfrak{t}$ below the annotated position that will be used for the rewrite step at node $v$, as otherwise, the position $\tau$ would not be annotated in $t_v$. Therefore, we must have a $1 \leq j \leq k$ and a $t'' \trianglelefteq_\sharp r_j$ such that $t''^\sharp\theta \to^*_{\mathrm{np}(\mathcal{A})} s^\sharp\sigma'$, which means that there must be a self-loop for the only ADP in $X$, which is a contradiction to our assumption that $X$ is a singleton consisting of an ADP that is not in any SCC of $\mathfrak{G}$.
Now, we have proven that the $\overline{X}$-CT $\mathfrak{T}$ does not introduce new annotations. By definition of an $\mathcal{A}$-CT, every infinite path must contain an infinite number of nodes in $\Gamma$, i.e., nodes where we rewrite at an annotation. Thus, every path in $\mathfrak{T}$ must be finite, which means that $\mathfrak{T}$ is finite itself, as the tree is finitely branching. But every finite CT converges with probability 1, which is a contradiction to our assumption that $\mathfrak{T}$ converges with probability $< 1$.

**Step 2** Composing SCCs maintains the `AST` property

In the second step, we show that composing SCCs maintains the `AST` property. More precisely, we prove the following:

> Let $\hat{X} \subseteq \mathfrak{W}$ and $\hat{Y} \subseteq \mathfrak{W}$ such that there are no $X_1, X_2 \in \hat{X}$ and $Y \in \hat{Y}$ which satisfy both $X_1 >^*_{\mathfrak{G}} Y >^*_{\mathfrak{G}} X_2$ and $Y \notin \hat{X}$, and such that there are no $Y_1, Y_2 \in \hat{Y}$ and $X \in \hat{X}$ which satisfy both $Y_1 >^*_{\mathfrak{G}} X >^*_{\mathfrak{G}} Y_2$ and $X \notin \hat{Y}$. If both $\overline{\bigcup_{X \in \hat{X}} X}$ and $\overline{\bigcup_{Y \in \hat{Y}} Y}$ are `AST`, then $\overline{\bigcup_{X \in \hat{X}} X \cup \bigcup_{Y \in \hat{Y}} Y}$ is `AST`. (7.29)

To show (7.29), we assume that both $\overline{\bigcup_{X \in \hat{X}} X}$ and $\overline{\bigcup_{Y \in \hat{Y}} Y}$ are `AST`. Let $\overline{Z} = \overline{\bigcup_{X \in \hat{X}} X \cup \bigcup_{Y \in \hat{Y}} Y}$. The property in (7.29) for $\hat{X}$ and $\hat{Y}$ says that a path between two nodes from $\bigcup_{X \in \hat{X}} X$ that only traverses nodes from $Z$ must also be a path that only traverses nodes from $\bigcup_{X \in \hat{X}} X$, so that $\bigcup_{Y \in \hat{Y}} Y$ cannot be used to "create" new paths between two nodes from $\bigcup_{X \in \hat{X}} X$, and vice versa. Assume for a contradiction that $\overline{Z}$ is not `AST`. By Lemma 7.3.3 there exists a $\overline{Z}$-CT $\mathfrak{T} = (N, E)$ that converges with probability $< 1$ and starts with $t$ where $\mathrm{pos}_{\mathcal{D}^\sharp}(t) = \{\pi_0\}$ and $\flat(t|_{\pi_0}) = s\theta$ for a substitution $\theta$ and an ADP $s \xhookrightarrow{m} \mu \in \overline{Z}$, such that the root of $\mathfrak{T}$ performs a step with this ADP at the position $\pi_0$.
If $s \xhookrightarrow{m} \mu \notin \bigcup_{X \in \hat{X}} X \cup \bigcup_{Y \in \hat{Y}} Y$, then the resulting terms contain no annotations anymore and this cannot start a CT that converges with probability $< 1$. W.l.o.g., we may assume that the ADP that is used for the rewrite step at the root is in $\bigcup_{X \in \hat{X}} X$. Otherwise, we simply swap $\bigcup_{X \in \hat{X}} X$ with $\bigcup_{Y \in \hat{Y}} Y$ in the following. Let $\alpha_0$ be the ADP that the root of $\mathfrak{T}$ uses for its step at the position $\pi_0$. Then every annotation of a term $t_v$ with $v \neq \mathfrak{r}$ results from an annotation that was created by a right-hand side of an ADP at an earlier node in $\Gamma$. If an annotation that was created by an ADP $\beta$ is later used for a step of the form **at** or **af** with an ADP $\beta'$, then its subterm was only rewritten with ADPs with the flag $\mathfrak{t}$ strictly below it in between, so by Definition 7.4.1, there is an edge from $\beta$ to $\beta'$ in $\mathfrak{G}$. Hence, by induction, every ADP that is used for a step of the form **at** or **af** in $\mathfrak{T}$ is reachable from $\alpha_0$ in $\mathfrak{G}$.
We can partition the set $\Gamma$ of our $\overline{Z}$-CT $\mathfrak{T}$ into the sets

- $\Gamma_1 = \{v \in \Gamma \mid v$ together with the labeling and its successors represents a step with an ADP from $\bigcup_{X \in \hat{X}} X\}$

- $\Gamma_2 = \Gamma \setminus \Gamma_1$

Note that the labeling represents a step with an ADP from $\overline{Z} \setminus \bigcup_{X \in \hat{X}} X$ at nodes $v \in \Gamma_2$. Every $\overline{\bigcup_{Y \in \hat{Y}} Y}$-CT converges with probability 1, since $\bigcup_{Y \in \hat{Y}} Y$ is AST. Thus, also every $\overline{\bigcup_{Y \in \hat{Y}} Y \setminus \bigcup_{X \in \hat{X}} X}$-CT converges with probability 1 (as it contains fewer annotations than $\overline{\bigcup_{Y \in \hat{Y}} Y}$). Every subtree of $\mathfrak{T}$ whose inner nodes all belong to $\overline{\Gamma} \cup \Gamma_2$ is such a CT, since it only uses ADPs from $\bigcup_{X \in \hat{X}} X$ for steps of the form **nt** or **nf**, which do not depend on the annotations of the used ADP. Furthermore, we have $|\mathfrak{T}| < 1$ by our assumption. By the $\Gamma$-Partition Lemma (Lemma 7.3.2) (with the roles of $\Gamma_1$ and $\Gamma_2$ swapped) we can find a grounded subtree $\mathfrak{T}' = (N', E')$ of $\mathfrak{T}$ that is again a $\overline{Z}$-CT with $|\mathfrak{T}'| < 1$ such that every infinite path has an infinite number of $\Gamma_1$-nodes.
We now construct a $\overline{\bigcup_{X \in \hat{X}} X}$-CT $\mathfrak{T}'' = (N', E'')$ with $\Gamma_1 \cap \Gamma^{\mathfrak{T}'} \subseteq \Gamma^{\mathfrak{T}''}$ that has the same underlying tree structure, and corresponding nodes have the same probabilities as in $\mathfrak{T}'$. Hence, we then obtain $|\mathfrak{T}'| = |\mathfrak{T}''|$. Moreover, every infinite path in $\mathfrak{T}''$ contains infinitely many $\Gamma^{\mathfrak{T}''}$-nodes, since every path in $\mathfrak{T}'$ contains infinitely many $\Gamma_1$-nodes and we have $\Gamma_1 \cap \Gamma^{\mathfrak{T}'} \subseteq \Gamma^{\mathfrak{T}''}$. But this implies that $\mathfrak{T}''$ is indeed a $\overline{\bigcup_{X \in \hat{X}} X}$-CT with $|\mathfrak{T}''| < 1$, which is a contradiction to our assumption that $\overline{\bigcup_{X \in \hat{X}} X}$ is AST.

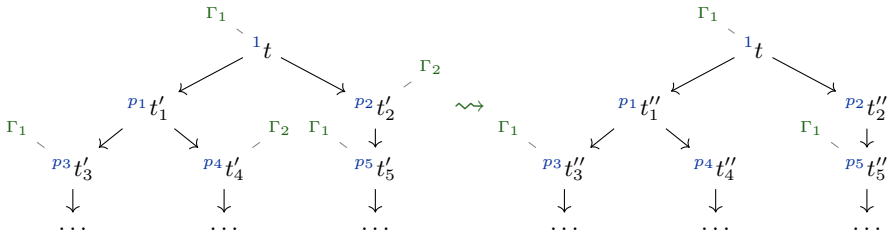



Figure 7.10: Construction for step 2.

The core idea of this construction is that annotations introduced by rewrite steps at a node $v \in \Gamma_2$ are not important for our computation. The reason is that if annotations are introduced using an ADP from $\bigcup_{Y \in \hat{Y}} Y$ that is not contained in $\bigcup_{X \in \hat{X}} X$, then by the prerequisite of (7.29), we know that such an ADP has no path in the dependency graph to an ADP in $\bigcup_{X \in \hat{X}} X$. Hence, by definition of the dependency graph, we are never able to use these terms for a rewrite step with an ADP from $\bigcup_{X \in \hat{X}} X$ to introduce new annotations. We can therefore apply the non-annotated ADP from $\bigcup_{Y \in \hat{Y}} Y$ instead.
We now construct the new labeling $L''$ for the $\overline{\bigcup_{X \in \hat{X}} X}$-CT $\mathfrak{T}''$ recursively. Let $W \subseteq N'$ be the set of nodes where we have already defined the labeling $L''$. Furthermore, for any term $t'_v$, let $\mathrm{Junk}_{\hat{X}}(t'_v)$ denote the positions of all annotated subterms $s \trianglelefteq_\sharp t'_v$ that can never be used for a rewrite step with an ADP from $\hat{X}$, as indicated by the dependency graph. To be precise, we define $\pi \in \mathrm{Junk}_{\hat{X}}(t'_v)$ if there is no $B \in \mathfrak{W}$ with $B >^*_{\mathfrak{G}} X$ for some $X \in \hat{X}$ such that there is an ADP $\ell_1 \xhookrightarrow{m} \{{}^{p_1}r_1, \ldots, {}^{p_k}r_k\} \in B$, and a substitution $\sigma$ with $\sharp_{\{\varepsilon\}}(t'_v|_\pi) \to^*_{\mathrm{np}(\mathcal{A})} \ell_1^\sharp \sigma$. During our construction, we ensure that for every $v \in W$ we have

$$t'_v \overset{\flat}{=} t''_v \text{ and } \mathrm{pos}_{\mathcal{D}^\sharp}(t'_v) \setminus \mathrm{Junk}_{\hat{X}}(t'_v) \subseteq \mathrm{pos}_{\mathcal{D}^\sharp}(t''_v). \tag{7.30}$$

We start by setting $t''_v = t'_v = t$ for the root $v$ of $\mathfrak{T}'$. Here, our property (7.30) is clearly satisfied. As long as there is still an inner node $v \in W$ such that its successors

are not contained in $W$, we do the following. Let $vE = \{w_1, \dots, w_k\}$ be the set of its successors. We need to define the corresponding terms for the nodes $w_1, \dots, w_k$ in $\mathfrak{T}''$. Since $v$ is not a leaf and $\mathfrak{T}'$ is a $\overline{Z}$-CT, we have $t'_v \hookrightarrow_{\overline{Z}} \{{}^{p_{w_1}/p_v}t'_{w_1}, \dots, {}^{p_{w_k}/p_v}t'_{w_k}\}$, and hence, we have to deal with the following two cases:

1. If we use an ADP from $\bigcup_{X \in \hat{X}} X$ in $\mathfrak{T}'$, then we perform the rewrite step with the same ADP, the same VRF $(\varphi_j)_{1 \le j \le k}$, the same position $\pi$, and the same substitution in $\mathfrak{T}''$. Since we have $t'_v \overset{\flat}{=}_{(IH)} t''_v$, we also get $t'_{w_j} \overset{\flat}{=} t''_{w_j}$ for all $1 \le j \le k$. Furthermore, since we rewrite at position $\pi$ it cannot be in $\mathrm{Junk}_{\hat{X}}(t'_v)$, and hence, if $\pi \in \mathrm{pos}_{\mathcal{D}^\sharp}(t'_v)$, then also $\pi \in \mathrm{pos}_{\mathcal{D}^\sharp}(t''_v)$ by (7.30). Thus, whenever we create annotations in the rewrite step in $\mathfrak{T}'$ (a step with **af** or **at**), then we do the same in $\mathfrak{T}''$ (the step is also an **af** or **at** step, respectively), and whenever we remove annotations in the rewrite step in $\mathfrak{T}''$ (a step with **af** or **nf**), then the same happened in $\mathfrak{T}'$ (the step is also an **af** or **nf** step). Therefore, we also get $\mathrm{pos}_{\mathcal{D}^\sharp}(t'_{w_j}) \setminus \mathrm{Junk}_{\hat{X}}(t'_{w_j}) \subseteq \mathrm{pos}_{\mathcal{D}^\sharp}(t''_{w_j})$ for all $1 \le j \le k$ and (7.30) is again satisfied.
2. If we use an ADP from $\overline{Z} \setminus \bigcup_{X \in \hat{X}} X$ in $\mathfrak{T}'$, and we use the ADP $\ell_1 \xhookrightarrow{m} \{{}^{p_1}r_1, \dots, {}^{p_k}r_k\}$, then we can use $\ell_1 \xhookrightarrow{m} \{{}^{p_1}\flat(r_1), \dots, {}^{p_k}\flat(r_k)\}$ instead, with the same VRF $(\varphi_j)_{1 \le j \le k}$, the same position $\pi$, and the same substitution. Note that if $\pi \in \mathrm{pos}_{\mathcal{D}^\sharp}(t'_v)$, then all positions of annotations introduced by the ADP are in $\mathrm{Junk}_{\hat{X}}(t'_{w_j})$ for all $1 \le j \le k$: the used ADP is in some $A \in \hat{Y}$ with $A \notin \hat{X}$, and it is reachable from $\alpha_0 \in X_1$ for some $X_1 \in \hat{X}$, i.e., $X_1 >^*_{\mathfrak{G}} A$. If an annotation introduced by this ADP were not in $\mathrm{Junk}_{\hat{X}}(t'_{w_j})$, then by Definition 7.4.1 we would also have $A >^*_{\mathfrak{G}} X_2$ for some $X_2 \in \hat{X}$, which contradicts the prerequisite of (7.29). Otherwise, we remove the annotations during the application of the rule anyway. Again, (7.30) is satisfied.

All in all, this shows that (7.29) holds.

**Step 3** The ADP problem $\overline{\bigcup_{X >^*_{\mathfrak{G}} Y} Y}$ is AST

Using (7.28) and (7.29), by induction on $>_{\mathfrak{G}}$ we now prove that

$$\text{for every } X \in \mathfrak{W}, \text{ the ADP problem } \overline{\textstyle\bigcup_{X >^*_{\mathfrak{G}} Y} Y} \text{ is } \mathtt{AST}. \tag{7.31}$$

Note that $>_{\mathfrak{G}}$ is well founded, since $\mathfrak{W}$ is finite and $>_{\mathfrak{G}}$ has no cycles.
For the base case, we consider an $X \in \mathfrak{W}$ that is minimal w.r.t. $>_{\mathfrak{G}}$. Hence, we have $\bigcup_{X >^*_{\mathfrak{G}} Y} Y = X$. By (7.28), $\overline{X}$ is AST.
For the induction step, we consider an $X \in \mathfrak{W}$ and assume that $\overline{\bigcup_{Y >^*_{\mathfrak{G}} Z} Z}$ is AST for every $Y \in \mathfrak{W}$ with $X >^+_{\mathfrak{G}} Y$. Let $\mathtt{Succ}(X) = \{Y \in \mathfrak{W} \mid X >_{\mathfrak{G}} Y\} = \{Y_1, \dots, Y_m\}$ be the set of all direct successors of $X$. The induction hypothesis states that $\overline{\bigcup_{Y_u >^*_{\mathfrak{G}} Z} Z}$ is AST for all $1 \le u \le m$. We first prove by induction that for all $1 \le u \le m$, $\overline{\bigcup_{1 \le i \le u} \bigcup_{Y_i >^*_{\mathfrak{G}} Z} Z}$ is AST.
In the inner induction base, we have $u = 1$ and hence $\overline{\bigcup_{1 \le i \le u} \bigcup_{Y_i >^*_{\mathfrak{G}} Z} Z} = \overline{\bigcup_{Y_1 >^*_{\mathfrak{G}} Z} Z}$.
By our outer induction hypothesis we know that $\overline{\bigcup_{Y_1 >^*_{\mathfrak{G}} Z} Z}$ is AST.
In the inner induction step, assume that the claim holds for some $1 \le u < m$. Then $\overline{\bigcup_{Y_{u+1} >^*_{\mathfrak{G}} Z} Z}$ is AST by our outer induction hypothesis and $\overline{\bigcup_{1 \le i \le u} \bigcup_{Y_i >^*_{\mathfrak{G}} Z} Z}$ is AST

by our inner induction hypothesis. By (7.29), we know that then $\overline{\bigcup_{1\leq i\leq u+1}\bigcup_{Y_i>^*_{\mathfrak{G}}Z} Z}$ is AST as well. The conditions for (7.29) are clearly satisfied, as we use the reflexive, transitive closure $>^*_{\mathfrak{G}}$ of the direct successor relation in both $\bigcup_{1\leq i\leq u}\bigcup_{Y_i>^*_{\mathfrak{G}}Z} Z$ and $\bigcup_{Y_{u+1}>^*_{\mathfrak{G}}Z} Z$.
Now we have shown that $\overline{\bigcup_{1\leq i\leq m}\bigcup_{Y_i>^*_{\mathfrak{G}}Z} Z}$ is AST. We know that $\overline{X}$ is AST by our assumption and that $\overline{\bigcup_{1\leq i\leq m}\bigcup_{Y_i>^*_{\mathfrak{G}}Z} Z}$ is AST. Hence, by (7.29) we obtain that $\overline{\bigcup_{X>^*_{\mathfrak{G}}Y} Y}$ is AST. Again, the conditions of (7.29) are satisfied, since $X$ is strictly greater w.r.t. $>^+_{\mathfrak{G}}$ than all $Z$ with $Y_i >^*_{\mathfrak{G}} Z$ for some $1 \leq i \leq m$.

**Step 4** $\mathcal{A}$ is AST

In (7.31) we have shown that $\overline{\bigcup_{X>^*_{\mathfrak{G}}Y} Y}$ for every $X \in \mathfrak{W}$ is AST. Let $X_1, \ldots, X_m \in \mathfrak{W}$ be the maximal elements of $\mathfrak{W}$ w.r.t. $>_{\mathfrak{G}}$. By induction, one can prove that $\overline{\bigcup_{1\leq i\leq u}\bigcup_{X_i>^*_{\mathfrak{G}}Y} Y}$ is AST for all $1 \leq u \leq m$ by (7.29), analogous to the previous induction. Again, the conditions of (7.29) are satisfied as we use the reflexive, transitive closure of $>_{\mathfrak{G}}$. In the end, we know that $\overline{\bigcup_{1\leq i\leq m}\bigcup_{X_i>^*_{\mathfrak{G}}Y} Y} = \mathcal{A}$ is AST and this ends the proof. *QED*

Example 7.4.3. Consider $\mathcal{P}_{\mathsf{alg}}$ and its canonical ADPs from Example 7.2.7. The $\mathcal{DP}(\mathcal{P}_{\mathsf{alg}})$-dependency graph is shown in Figure 7.11. Its SCCs are $\{(\sharp 7.5), (\sharp 7.6)\}$, $\{(\sharp 7.7)\}$, $\{(\sharp 7.8)\}$, and $\{(\sharp 7.10)\}$. For each SCC we create a separate ADP problem, where all annotations outside the SCC are removed. This leads to the ADP problems $\{(\sharp 7.5), (\sharp 7.6), \flat(\sharp 7.7)\text{–}\flat(\sharp 7.11)\}$, $\{(\sharp 7.7), \flat(\sharp 7.5), \flat(\sharp 7.6), \flat(\sharp 7.8)\text{–}\flat(\sharp 7.11)\}$, $\{(\sharp 7.8), \flat(\sharp 7.5)\text{–}\flat(\sharp 7.7), \flat(\sharp 7.9)\text{–}\flat(\sharp 7.11)\}$, and $\{(\sharp 7.10), \flat(\sharp 7.5)\text{–}\flat(\sharp 7.9), \flat(\sharp 7.11)\}$. To illustrate, the problem $\{(\sharp 7.5), (\sharp 7.6), \flat(\sharp 7.7)\text{–}\flat(\sharp 7.11)\}$ contains the ADPs

$$\mathsf{loop1}(y) \hookrightarrow^{\mathsf{t}} \{^{1/2}\mathsf{loop1}^\sharp(\mathsf{double}^\sharp(y)),\ ^{1/2}\mathsf{loop2}^\sharp(\mathsf{double}^\sharp(y))\} \qquad (\sharp 7.5)$$

$$\mathsf{loop1}(y) \hookrightarrow^{\mathsf{t}} \{^{1/3}\mathsf{loop1}^\sharp(\mathsf{triple}^\sharp(y)),\ ^{2/3}\mathsf{loop2}^\sharp(\mathsf{triple}^\sharp(y))\} \qquad (\sharp 7.6)$$

$$\mathsf{loop2}(\mathsf{s}(y)) \hookrightarrow^{\mathsf{t}} \{^{1}\mathsf{loop2}(y)\} \qquad (\flat(\sharp 7.7))$$

$$\mathsf{double}(\mathsf{s}(y)) \hookrightarrow^{\mathsf{t}} \{^{1}\mathsf{s}(\mathsf{s}(\mathsf{double}(y)))\} \qquad (\flat(\sharp 7.8))$$

$$\mathsf{double}(0) \hookrightarrow^{\mathsf{t}} \{^{1}0\} \qquad (\flat(\sharp 7.9))$$

$$\mathsf{triple}(\mathsf{s}(y)) \hookrightarrow^{\mathsf{t}} \{^{1}\mathsf{s}(\mathsf{s}(\mathsf{triple}(y)))\} \qquad (\flat(\sharp 7.10))$$

$$\mathsf{triple}(0) \hookrightarrow^{\mathsf{t}} \{^{1}0\} \qquad (\flat(\sharp 7.11))$$

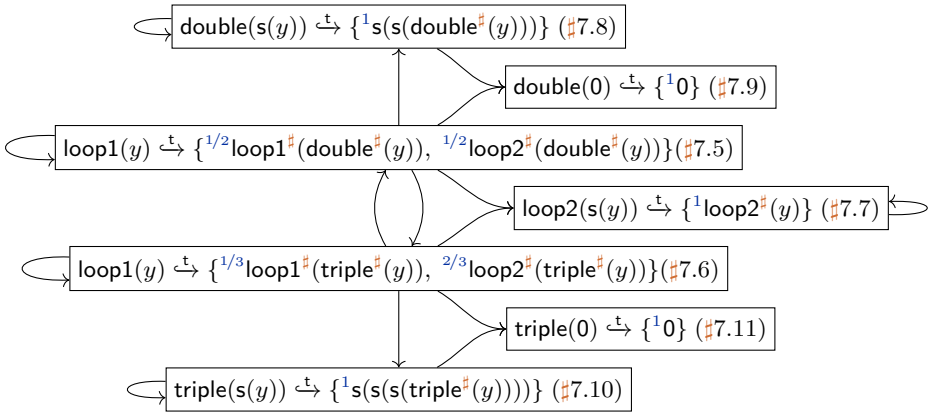



Figure 7.11: $\mathcal{DP}(\mathcal{P}_{\mathsf{alg}})$-dependency graph.

Counterexample 7.4.4. If we used GVRFs that can duplicate annotations, then the dependency graph processor would not be sound. The reason is that $\mathrm{Proc}_{\mathtt{DG}}$ maps ADP problems without annotations to the empty set. However, this would be unsound if we had GVRFs, because then the ADP problem $\mathcal{A}$ consisting of $\mathsf{a} \xhookrightarrow{\mathsf{t}} \{^1\mathsf{b}\}$ and $\mathsf{d}(x) \xhookrightarrow{\mathsf{t}} \{^1\mathsf{c}(x, \mathsf{d}(x))\}$ would not be $\mathtt{AST}$. Here, the use of GVRFs would lead to the $\mathcal{A}$-CT in Figure 7.12 with an infinite number of **at** steps that rewrite $\mathsf{a}^\sharp$ to $\mathsf{b}$.

$$^1\underline{\mathsf{d}}(\mathsf{a}^\sharp) \longrightarrow {}^1\mathsf{c}(\underline{\mathsf{a}}^\sharp, \mathsf{d}(\mathsf{a}^\sharp)) \longrightarrow {}^1\mathsf{c}(\mathsf{b}, \underline{\mathsf{d}}(\mathsf{a}^\sharp)) \longrightarrow \cdots$$

Figure 7.12: $\mathcal{A}$-CT with an infinite number of **at** steps.

**Dependency Graph Processor for $\mathtt{iAST}$** As in Section 7.1, we can integrate the restriction to innermost evaluation into the dependency graph. So instead of arbitrary rewrite steps, we now only allow innermost steps. Moreover, the substitutions $\sigma_1, \sigma_2$ need to be chosen in such a way that both ADPs can be applied in innermost reductions, i.e., we must have $\ell_1\sigma_1, \ell_2\sigma_2 \in \mathtt{ANF}_{\mathcal{A}}$.

Definition 7.4.5 – Innermost Dependency Graph.

*Let $\mathcal{A}$ be an ADP problem. The innermost $\mathcal{A}$-dependency graph has the nodes $\mathcal{A}$, and there is an edge from $\ell_1 \xhookrightarrow{m} \{^{p_1}r_1, \ldots, {}^{p_k}r_k\}$ to $\ell_2 \xhookrightarrow{m} \mu_2$ if there are substitutions $\sigma_1, \sigma_2$ and a $t \trianglelefteq_\sharp r_j$ for some $1 \leq j \leq k$ such that $t^\sharp\sigma_1 \xrightarrow{\mathtt{i}}{}^*_{\mathrm{np}(\mathcal{A})} \ell_2^\sharp\sigma_2$ and both $\ell_1\sigma_1$ and $\ell_2\sigma_2$ are in $\mathtt{ANF}_{\mathcal{A}}$.*

Again, the same over-approximation techniques as in Section 7.1 can also be used to estimate the innermost dependency graph in the probabilistic setting.

Theorem 7.4.6 – Dependency Graph Processor for $\mathtt{iAST}$.

Let $\mathcal{A}$ be an ADP problem, and let $\mathcal{A}_1, \ldots, \mathcal{A}_n$ be the SCCs of the innermost $\mathcal{A}$-dependency graph. Then $\mathrm{Proc}^{\mathtt{i}}_{\mathtt{DG}}(\mathcal{A}) = \{\mathcal{A}_1 \cup \flat(\mathcal{A} \setminus \mathcal{A}_1), \ldots, \mathcal{A}_n \cup \flat(\mathcal{A} \setminus \mathcal{A}_n)\}$ is sound and complete for $\mathtt{iAST}$.

*Proof.* We prove both directions individually. Let $\overline{X} = X \cup \flat(\mathcal{A} \setminus X)$ for $X \subseteq \mathcal{A}$.

▶*Proof*(*Completeness*): By Theorem 7.3.1.

▶*Proof*(*Soundness*): The proof is analogous to the proof for $\mathtt{AST}$ but uses the innermost dependency graph $\mathfrak{G}$ instead. Only Step 1. and 2. change slightly due to innermost evaluation.

Step 1. Reusing the notation from the proof for $\mathtt{AST}$, existence of a node $v \in \Gamma$ in $\mathfrak{T}$ that is not the root and where we introduce new annotations leads to a term $t' \trianglelefteq_\sharp t_v$ such that $t' = s\sigma'$ for some substitution $\sigma'$ and the ADP $s \xhookrightarrow{m} \mu \in X$, where $s\sigma' \in \mathtt{ANF}_{\mathcal{A}}$. The first rewrite step at the root is $s\theta \xhookrightarrow{\mathtt{i}}_{\overline{X}} \{^{p_1}r_1\theta, \ldots, {}^{p_k}r_k\theta\}$. Therefore, we must have a $1 \leq j \leq k$ and a $t'' \trianglelefteq_\sharp r_j$ such that $t''^\sharp\theta \xrightarrow{\mathtt{i}}{}^*_{\mathrm{np}(\mathcal{A})} s^\sharp\sigma'$ and $s\theta, s\sigma' \in \mathtt{ANF}_{\mathcal{A}}$, which means that there must be a self-loop for the only ADP in $X$, which is the desired contradiction.

Step 2. The construction in this step now considers innermost CTs. Hence, the definition of $\mathrm{Junk}_{\hat{X}}(t'_v)$ uses the innermost dependency graph and innermost evaluation. To be precise, we define $\pi \in \mathrm{Junk}_{\hat{X}}(t'_v)$ if there is no $B \in \mathfrak{W}$ with $B >^*_{\mathfrak{G}} X$ for some $X \in \hat{X}$ such that there is an ADP $\ell_1 \xhookrightarrow{m} \{^{p_1}r_1, \ldots, {}^{p_k}r_k\} \in B$, and a substitution $\sigma$

*Proof*

with $\sharp_{\{\varepsilon\}}(t'_v|_\pi) \xrightarrow{\mathtt{i}}{}^*_{\mathrm{np}(\mathcal{A})} \ell_1^\sharp\sigma$ and $\ell_1^\sharp\sigma \in \mathtt{ANF}_\mathcal{A}$. Due to the definition of the innermost dependency graph, we can still perform each rewrite step from $\mathfrak{T}'$ analogously in $\mathfrak{T}''$ with $\Gamma_1 \cap \Gamma^{\mathfrak{T}'} \subseteq \Gamma^{\mathfrak{T}''}$.

*QED*

Example 7.4.7. For $\mathcal{P}_{\mathsf{alg}}$, the innermost and the full $\mathcal{DP}(\mathcal{P}_{\mathsf{alg}})$-dependency graph coincide. Thus, for $\mathtt{AST}(\xhookrightarrow{\mathtt{i}}_\mathcal{A})$ we obtain the same ADP problems as in Example 7.4.3.

Example 7.4.8. For the duplicating PTRS $\mathcal{P}_{\mathsf{dup}}$ and its canonical ADPs $\mathcal{DP}(\mathcal{P}_{\mathsf{dup}}) = \{(\sharp 7.12), (\sharp 7.14)\}$ from Example 7.2.6 we can only analyze $\mathtt{AST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}_{\mathsf{dup}}})$. The innermost $\mathcal{DP}(\mathcal{P}_{\mathsf{dup}})$-dependency graph can be seen in Figure 7.13. As the only SCC $\{(\sharp 7.12)\}$ does not contain $(\sharp 7.14)$, we can remove all annotations from $(\sharp 7.14)$. However, since $(\sharp 7.14)$ already has no annotations, $\mathrm{Proc}^{\mathtt{i}}_{\mathtt{DG}}$ does not change $\mathcal{DP}(\mathcal{P}_{\mathsf{dup}})$.




Figure 7.13: Innermost $\mathcal{DP}(\mathcal{P}_{\mathsf{dup}})$-dependency graph.

### 7.4.2 Usable Terms Processor

The dependency graph processor either removes all annotations from an ADP or none. But an ADP can still contain terms $t^\sharp$ where no instance $t^\sharp\sigma_1$ rewrites to an instance $\ell^\sharp\sigma_2$ of a left-hand side $\ell$ of an ADP with annotations, i.e., it can only lead to ADPs outside the SCC we consider. The *usable terms processor* removes the annotation from the root of such *non-usable* terms like $\mathsf{d}^\sharp(\dots)$ in $\mathcal{DP}(\mathcal{P}_{\mathsf{dup}}) = \{\mathsf{g} \xhookrightarrow{\mathtt{t}} \{^{3/4}\mathsf{d}^\sharp(\mathsf{g}^\sharp), {}^{1/4}0\}\ (\sharp 7.12),\ \mathsf{d}(x) \xhookrightarrow{\mathtt{t}} \{^1\mathsf{c}(x,x)\}\ (\sharp 7.14)\}$. So instead of whole ADPs, here we consider the subterms in the right-hand sides of an ADP individually. Such a processor was not needed in the non-probabilistic setting, as ordinary dependency pairs only contain a single annotation.

**Usable Terms Processor for `AST`** We start with the processor for `AST`.

Theorem 7.4.9 – Usable Terms Processor for `AST`.

Let $\mathcal{A}$ be an ADP problem. We call $t \in \mathcal{T}^\sharp$ with $\mathrm{root}(t) \in \mathcal{D}^\sharp$ *usable* w.r.t. $\mathcal{A}$ if there are substitutions $\sigma_1, \sigma_2$ and an $\ell_2 \xhookrightarrow{m'} \mu_2 \in \mathcal{A}$ where $\mu_2$ contains an annotated symbol, such that $\sharp_{\{\varepsilon\}}(t)\sigma_1 \to^*_{\mathrm{np}(\mathcal{A})} \ell_2^\sharp\sigma_2$. Let $\Delta_\mathcal{A}(s) = \{\pi \in \mathrm{pos}_{\mathcal{D}^\sharp}(s) \mid s|_\pi \text{ is usable w.r.t. } \mathcal{A}\}$ and let

$$\mathcal{T}_{\mathtt{UT}}(\mathcal{A}) = \{\ell \xhookrightarrow{m} \{^{p_1}\sharp_{\Delta_\mathcal{A}(r_1)}(r_1), \dots, {}^{p_k}\sharp_{\Delta_\mathcal{A}(r_k)}(r_k)\} \mid \ell \xhookrightarrow{m} \{^{p_1}r_1, \dots, {}^{p_k}r_k\} \in \mathcal{A}\}$$

be the transformation that removes the annotations from the roots of all non-usable terms in the right-hand sides. Then $\mathrm{Proc}_{\mathtt{UT}}(\mathcal{A}) = \{\mathcal{T}_{\mathtt{UT}}(\mathcal{A})\}$ is sound and complete for `AST`.

*Proof.* We prove both directions individually.

*Proof*

▶ *Proof*(*Completeness*): By Theorem 7.3.1.

▶ *Proof*(*Soundness*): Let $\mathcal{A}$ be not `AST`. Then by Lemma 7.3.3 there exists an $\mathcal{A}$-CT $\mathfrak{T} = (N, E)$ that converges with probability $< 1$ that starts with $t$ and $\mathrm{pos}_{\mathcal{D}^\sharp}(t) = \{\pi_0\}$ for some position $\pi_0$. We will now create a $\mathcal{T}_{\mathtt{UT}}(\mathcal{A})$-CT $\mathfrak{T}' = (N, E')$, with the same underlying tree structure, such that $p_v^\mathfrak{T} = p_v^{\mathfrak{T}'}$ for all $v \in N$. Hence, we get

$|\mathfrak{T}'| = |\mathfrak{T}| < 1$, and thus $\mathcal{T}_{\mathtt{UT}}(\mathcal{A})$ is not `AST` either.
We construct the new labeling $L'$ for the $\mathcal{T}_{\mathtt{UT}}(\mathcal{A})$-CT $\mathfrak{T}'$ recursively. Let $W \subseteq N$ be the set of nodes where we have already defined the labeling $L'$. During our construction, we ensure that for every node $v \in W$ we have

$$t_v \overset{\flat}{=} t'_v \text{ and } \mathrm{pos}_{\mathcal{D}^\sharp}(t_v) \setminus \mathrm{Junk}(t_v) \subseteq \mathrm{pos}_{\mathcal{D}^\sharp}(t'_v). \tag{7.32}$$

Here, for any term $t_v$, let $\mathrm{Junk}(t_v)$ be the set of positions that can never be used for a rewrite step with an ADP that contains annotations. To be precise, we define $\pi \in \mathrm{Junk}(t_v)$ if there is no ADP $\ell_1 \overset{m}{\hookrightarrow} \{{}^{p_1}r_1, \ldots, {}^{p_k}r_k\} \in \mathcal{A}$ with annotations and substitution $\sigma$ such that $\sharp_{\{\varepsilon\}}(t_v|_\pi) \to^*_{\mathrm{np}(\mathcal{A})} \ell_1^\sharp\sigma$.
We start with the same term $t$ at the root. Here, our property (7.32) is clearly satisfied. As long as there is still an inner node $v \in W$ such that its successors are not contained in $W$, we do the following. Let $vE = \{w_1, \ldots, w_k\}$ be the set of its successors. We need to define the terms for the nodes $w_1, \ldots, w_k$ in $\mathfrak{T}'$. Since $v$ is not a leaf and $\mathfrak{T}$ is an $\mathcal{A}$-CT, we have $t_v \hookrightarrow_{\mathcal{A}} \{{}^{p_{w_1}/p_v}t_{w_1}, \ldots, {}^{p_{w_k}/p_v}t_{w_k}\}$. If we performed a step with $\hookrightarrow_{\mathcal{A}}$ using the ADP $\ell \overset{m}{\hookrightarrow} \{{}^{p_1}r_1, \ldots, {}^{p_k}r_k\}$, the VRF $(\varphi_j)_{1\le j\le k}$, the position $\pi$, and the substitution $\sigma$ in $\mathfrak{T}$, then we can use the ADP $\ell \overset{m}{\hookrightarrow} \{{}^{p_1}\sharp_{\Delta_{\mathcal{A}}(r_1)}(r_1), \ldots, {}^{p_k}\sharp_{\Delta_{\mathcal{A}}(r_k)}(r_k)\}$ with the same VRF $(\varphi_j)_{1\le j\le k}$, the same position $\pi$, and the same substitution $\sigma$. Now, we directly get $t_{w_j} \overset{\flat}{=} t'_{w_j}$ for all $1 \le j \le k$. To prove $\mathrm{pos}_{\mathcal{D}^\sharp}(t_{w_j}) \setminus \mathrm{Junk}(t_{w_j}) \subseteq \mathrm{pos}_{\mathcal{D}^\sharp}(t'_{w_j})$, note that if $\pi \in \mathrm{pos}_{\mathcal{D}^\sharp}(t_v) \cap \mathrm{Junk}(t_v)$, then $\ell \overset{m}{\hookrightarrow} \{{}^{p_1}r_1, \ldots, {}^{p_k}r_k\}$ contains no annotations by definition of $\mathrm{Junk}(t_v)$. Therefore, it does not matter whether we rewrite with case **at** or **nt** (**af** or **nf**). Otherwise, if $\pi \in \mathrm{pos}_{\mathcal{D}^\sharp}(t_v) \setminus \mathrm{Junk}(t_v)$, then the original rule contains the same terms with possibly more annotations, but all missing annotations are in $\mathrm{Junk}(t_v)$ by definition of $\sharp_{\Delta_{\mathcal{A}}(r_j)}(r_j)$. Thus, we get $\mathrm{pos}_{\mathcal{D}^\sharp}(t_{w_j}) \setminus \mathrm{Junk}(t_{w_j}) \subseteq \mathrm{pos}_{\mathcal{D}^\sharp}(t'_{w_j})$ for all $1 \le j \le k$.
Finally, in order to see that $\mathfrak{T}'$ is indeed a $\mathcal{T}_{\mathtt{UT}}(\mathcal{A})$-CT, we have to make sure that the liveness property (on every infinite path there are an infinite number of rewrite steps with Case **at** or **af**) is satisfied. Assume otherwise, i.e., that there exists an infinite path with no such rewrite steps. The same path exists in $\mathfrak{T}$, which is an $\mathcal{A}$-CT, so there are infinitely many rewrite steps with Case **at** or **af** in $\mathfrak{T}$. This means that every rewrite step with Case **at** or **af** at a node $v$ in this path is performed at a position in $\mathrm{Junk}(t_v)$. But since such rewrite steps decrease the number of annotations in the term, we cannot have infinitely many of them without rewriting at a position in $\mathrm{pos}_{\mathcal{D}^\sharp}(t_v) \setminus \mathrm{Junk}(t_v)$, which is a contradiction. *QED*

Example 7.4.10. For `AST`, $\mathrm{Proc}_{\mathtt{UT}}$ transforms $\{(\sharp 7.5), (\sharp 7.6), \flat(\sharp 7.7)\text{–}\flat(\sharp 7.11)\}$ from Example 7.4.3 with

$$\mathsf{loop1}(y) \overset{\mathrm{t}}{\hookrightarrow} \{{}^{1/2}\mathsf{loop1}^\sharp(\mathsf{double}^\sharp(y)),\ {}^{1/2}\mathsf{loop2}^\sharp(\mathsf{double}^\sharp(y))\} \tag{$\sharp$7.5}$$
$$\mathsf{loop1}(y) \overset{\mathrm{t}}{\hookrightarrow} \{{}^{1/3}\mathsf{loop1}^\sharp(\mathsf{triple}^\sharp(y)),\ {}^{2/3}\mathsf{loop2}^\sharp(\mathsf{triple}^\sharp(y))\} \tag{$\sharp$7.6}$$
$$\mathsf{loop2}(\mathsf{s}(y)) \overset{\mathrm{t}}{\hookrightarrow} \{{}^{1}\mathsf{loop2}(y)\} \tag{$\flat(\sharp$7.7)}$$
$$\mathsf{double}(\mathsf{s}(y)) \overset{\mathrm{t}}{\hookrightarrow} \{{}^{1}\mathsf{s}(\mathsf{s}(\mathsf{double}(y)))\} \tag{$\flat(\sharp$7.8)}$$
$$\mathsf{double}(0) \overset{\mathrm{t}}{\hookrightarrow} \{{}^{1}0\} \tag{$\flat(\sharp$7.9)}$$
$$\mathsf{triple}(\mathsf{s}(y)) \overset{\mathrm{t}}{\hookrightarrow} \{{}^{1}\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{triple}(y))))\} \tag{$\flat(\sharp$7.10)}$$
$$\mathsf{triple}(0) \overset{\mathrm{t}}{\hookrightarrow} \{{}^{1}0\} \tag{$\flat(\sharp$7.11)}$$

into $\{(\sharp 7.5'), (\sharp 7.6'), \flat(\sharp 7.7)\text{–}\flat(\sharp 7.11)\}$ with

$$\mathsf{loop1}(y) \overset{\mathrm{t}}{\hookrightarrow} \{{}^{1/2}\mathsf{loop1}^\sharp(\mathsf{double}(y)),\ {}^{1/2}\mathsf{loop2}(\mathsf{double}(y))\} \tag{$\sharp$7.5$'$}$$
$$\mathsf{loop1}(y) \overset{\mathrm{t}}{\hookrightarrow} \{{}^{1/3}\mathsf{loop1}^\sharp(\mathsf{triple}(y)),\ {}^{2/3}\mathsf{loop2}(\mathsf{triple}(y))\} \tag{$\sharp$7.6$'$}$$

The reason is that the left-hand sides of the only ADPs with annotations in the ADP problem have the root loop1. Thus, $\mathsf{loop2}^\sharp$-, $\mathsf{double}^\sharp$-, or $\mathsf{triple}^\sharp$-terms are not usable.

**Usable Terms Processor for iAST** As for the dependency graph processor, for $\mathtt{AST}(\overset{\mathtt{i}}{\hookrightarrow}_{\mathcal{A}})$ we only consider innermost evaluation in the definitions.

Theorem 7.4.11 – Usable Terms Processor for iAST.
Let $\ell_1 \in \mathcal{T}$ and $\mathcal{A}$ be an ADP problem. We call $t \in \mathcal{T}^\sharp$ with $\mathrm{root}(t) \in \mathcal{D}^\sharp$ *innermost usable* w.r.t. $\ell_1$ and $\mathcal{A}$ if there are substitutions $\sigma_1, \sigma_2$ and an $\ell_2 \overset{m'}{\hookrightarrow} \mu_2 \in \mathcal{A}$ where $\mu_2$ contains an annotated symbol, such that $\sharp_{\{\varepsilon\}}(t)\sigma_1 \overset{\mathtt{i}}{\to}^*_{\mathrm{np}(\mathcal{A})} \ell_2^\sharp\sigma_2$ and both $\ell_1\sigma_1$ and $\ell_2\sigma_2$ are in $\mathtt{ANF}_{\mathcal{A}}$. Let $\Delta^{\mathtt{i}}_{\ell,\mathcal{A}}(s) = \{\pi \in \mathrm{pos}_{\mathcal{D}^\sharp}(s) \mid s|_\pi$ is innermost usable w.r.t. $\ell$ and $\mathcal{A}\}$ and

$$\mathcal{T}^{\mathtt{i}}_{\mathtt{UT}}(\mathcal{A}) = \{\ell \overset{m}{\hookrightarrow} \{{}^{p_1}\sharp_{\Delta^{\mathtt{i}}_{\ell,\mathcal{A}}(r_1)}(r_1), \ldots, {}^{p_k}\sharp_{\Delta^{\mathtt{i}}_{\ell,\mathcal{A}}(r_k)}(r_k)\} \mid \ell \overset{m}{\hookrightarrow} \{{}^{p_1}r_1, \ldots, {}^{p_k}r_k\} \in \mathcal{A}\}.$$

Then $\mathrm{Proc}^{\mathtt{i}}_{\mathtt{UT}}(\mathcal{A}) = \{\mathcal{T}^{\mathtt{i}}_{\mathtt{UT}}(\mathcal{A})\}$ is sound and complete for iAST.

*Proof.* We prove both directions individually.

▶ *Proof*(*Completeness*): By Theorem 7.3.1.

▶ *Proof*(*Soundness*): We do the same construction as for AST but the definition of $\mathrm{Junk}(t_v)$ changes according to innermost evaluation. Here, we define $\pi \in \mathrm{Junk}(t_v)$ if there is no ADP $\ell_1 \overset{m}{\hookrightarrow} \{{}^{p_1}r_1, \ldots, {}^{p_k}r_k\} \in \mathcal{A}$ with annotations and no substitution $\sigma$ such that $\sharp_{\{\varepsilon\}}(t_v|_\pi) \overset{\mathtt{i}}{\to}^*_{\mathrm{np}(\mathcal{A})} \ell_1^\sharp\sigma$ and $\ell_1^\sharp\sigma \in \mathtt{ANF}_{\mathcal{A}}$.

*Proof* … *QED*

Example 7.4.12. For $\mathcal{P}_{\mathsf{alg}}$, up to now the proof of $\mathtt{AST}(\overset{\mathtt{i}}{\to}_{\mathcal{P}_{\mathsf{alg}}})$ is identical to the proof of $\mathtt{AST}(\to_{\mathcal{P}_{\mathsf{alg}}})$, i.e., here the results of $\mathrm{Proc}^{\mathtt{i}}_{\mathtt{UT}}$ on the ADP problems resulting from the (innermost) dependency graph processor are the same as the results of $\mathrm{Proc}_{\mathtt{UT}}$.

Example 7.4.13. When considering $\mathcal{P}_{\mathsf{dup}}$ and its ADPs

$$\mathsf{g} \overset{\mathtt{t}}{\hookrightarrow} \{{}^{3/4}\mathsf{d}^\sharp(\mathsf{g}^\sharp), {}^{1/4}0\}\ (\sharp 7.12), \qquad \mathsf{d}(x) \overset{\mathtt{t}}{\hookrightarrow} \{{}^{1}\mathsf{c}(x,x)\}\ (\sharp 7.14)$$

from Example 7.2.6 again, $\mathrm{Proc}^{\mathtt{i}}_{\mathtt{UT}}$ replaces ($\sharp$7.12) by $\mathsf{g} \overset{\mathtt{t}}{\hookrightarrow} \{{}^{3/4}\mathsf{d}(\mathsf{g}^\sharp), {}^{1/4}0\}$ ($\sharp 7.12'$).

### 7.4.3 Usable Rules Processor

Next, we consider the *usable rules processor*. Here, we will see a major difference between the ADP frameworks for $\mathtt{AST}(\hookrightarrow_{\mathcal{A}})$ and $\mathtt{AST}(\overset{\mathtt{i}}{\hookrightarrow}_{\mathcal{A}})$, since it will turn out that this processor is only sound for iAST, but not for AST. Therefore, this time we consider innermost evaluation first.

**Usable Rules Processor for iAST** In an innermost rewrite step, all variables of the used rule are instantiated with normal forms. The *usable rules processor* detects rules that cannot be used below annotations in right-hand sides of ADPs when their variables are instantiated with normal forms. For these rules we can set their flag to f, indicating that the annotated subterms on their right-hand sides may still lead to a non-$\mathtt{AST}(\overset{\mathtt{i}}{\hookrightarrow}_{\mathcal{A}})$ sequence, but the context of these annotations is irrelevant.

Theorem 7.4.14 – Usable Rules Processor for iAST.

For every $f \in \Sigma^\sharp$ and ADP problem $\mathcal{A}$, let $\mathrm{Rules}_\mathcal{A}(f) = \{\ell \xhookrightarrow{\mathtt{t}} \mu \in \mathcal{A} \mid \mathrm{root}(\ell) = f\}$. Moreover, for every $t \in \mathcal{T}^\sharp$, the *usable rules* $\mathcal{U}_\mathcal{A}(t)$ of $t$ w.r.t. $\mathcal{A}$ are recursively defined as follows:

$$\mathcal{U}_\mathcal{A}(t) = \varnothing, \qquad \text{if } t \in \mathcal{V} \text{ or } \mathcal{A} = \varnothing$$

$$\mathcal{U}_\mathcal{A}(f(t_1,\dots,t_n)) = \mathrm{Rules}_\mathcal{A}(f) \cup \bigcup_{j=1}^{n} \mathcal{U}_{\mathcal{A}'}(t_j) \cup \bigcup_{\substack{\ell \xhookrightarrow{\mathtt{t}} \mu \in \mathrm{Rules}_\mathcal{A}(f),\\ r \in \mathrm{Supp}(\mu)}} \mathcal{U}_{\mathcal{A}'}(\flat(r)), \quad \text{otherwise,}$$

where $\mathcal{A}' = \mathcal{A} \setminus \mathrm{Rules}_\mathcal{A}(f)$.

The *usable rules* of $\mathcal{A}$ are $\mathcal{U}(\mathcal{A}) = \bigcup_{\ell \xrightarrow{m} \mu \in \mathcal{A}, r \in \mathrm{Supp}(\mu), t \trianglelefteq^\sharp r} \mathcal{U}_\mathcal{A}(t^\sharp)$. Then, the processor $\mathrm{Proc}^{\mathtt{i}}_{\mathtt{UR}}(\mathcal{A}) = \{\mathcal{U}(\mathcal{A}) \cup \{\ell \xhookrightarrow{\mathtt{f}} \mu \mid \ell \xhookrightarrow{m} \mu \in \mathcal{A} \setminus \mathcal{U}(\mathcal{A})\}\}$ is sound and complete for iAST, i.e., we turn the flag $m$ of all non-usable rules to f.

*Proof.* We prove both directions individually. Let $\overline{\mathcal{A}} = \mathcal{U}(\mathcal{A}) \cup \{\ell \xhookrightarrow{\mathtt{f}} \mu \mid \ell \xhookrightarrow{m} \mu \in \mathcal{A} \setminus \mathcal{U}(\mathcal{A})\}$.

▶ *Proof(Completeness)*: By Theorem 7.3.1.

▶ *Proof(Soundness)*: Assume that $\mathtt{AST}(\xhookrightarrow{\mathtt{i}}_\mathcal{A})$ does not hold. Then by Lemma 7.3.3 there exists an innermost $\mathcal{A}$-CT $\mathfrak{T} = (N, E)$ that converges with probability $< 1$ that starts with $t$ and $\flat(t) = s\theta \in \mathtt{ANF}_\mathcal{A}$ for a substitution $\theta$ and an ADP $s \xhookrightarrow{m} \mu \in \mathcal{A}$, and $\mathrm{pos}_{\mathcal{D}^\sharp}(t) = \{\varepsilon\}$, and whose root performs its rewrite step at the position $\varepsilon$. We show that ADPs $\ell \xhookrightarrow{\mathtt{t}} \mu \in \mathcal{A}$ that are not usable (i.e., $\ell \xhookrightarrow{\mathtt{t}} \mu \notin \mathcal{U}(\mathcal{A})$) are never used strictly below an annotated position in $\mathfrak{T}$. Since the root performs its step at $\varepsilon$, every annotated subterm of a term $t_v$ with $v \neq \mathtt{r}$ results from an annotated subterm $r|_\chi$ of a right-hand side of an ADP that was used for a step of the form **at** or **af** at an earlier node $z$ with some substitution $\sigma$. As this step is innermost, $\sigma$ instantiates all variables of $r$ by normal forms. Due to innermost evaluation, this annotation is removed by every later step above it, and a step with an ADP with the flag f strictly below it removes it as well. So as long as the annotation is present, its subterm results from $\flat(r|_\chi)\sigma$ by innermost steps strictly below its root, which only use ADPs with the flag t. As in the non-probabilistic setting [GTSF06], by the definition of usable rules, all these steps use ADPs from $\mathcal{U}_\mathcal{A}(\sharp_{\{\varepsilon\}}(r|_\chi)) \subseteq \mathcal{U}(\mathcal{A})$. So every step with a non-usable ADP with the flag t in $\mathfrak{T}$ takes place at a position that is not strictly below any annotation, and thus, changing the flag of this ADP to f does not remove any annotations. Hence, we can also view $\mathfrak{T}$ as an innermost $\overline{\mathcal{A}}$-CT (note that $\mathtt{ANF}_\mathcal{A} = \mathtt{ANF}_{\overline{\mathcal{A}}}$) that converges with probability $< 1$ and thus $\mathtt{AST}(\xhookrightarrow{\mathtt{i}}_{\overline{\mathcal{A}}})$ does not hold.

*Proof* *QED*

Note that the innermost evaluation strategy is not affected by our usable rules processor, since it does not remove rules but only changes their flag to f. This is different from the non-probabilistic DP framework, where the usable rules processor from Theorem 7.1.13 reduces the number of rules. This may result in new redexes that are allowed for innermost evaluation. So in [GTS04] one has to extend DP problems by an additional component to achieve completeness of this processor (see Footnote 2), whereas the usable rules processor in our new ADP framework is complete.

Once the flag of an ADP has been turned to f and the ADP contains no annotations, it has become irrelevant in the sense of Theorem 7.3.5. Hence, we may afterwards replace all such ADPs by trivially terminating rules with the processor $\mathrm{Proc}^{\mathtt{i}}_{\mathtt{irr}}$, which

preserves the set of normal forms.

Example 7.4.15. The ADP problem $\{(\sharp 7.12'), (\sharp 7.14)\}$ with

$$\mathsf{g} \xhookrightarrow{t} \{^{3/4}\mathsf{d}(\mathsf{g}^\sharp),\ ^{1/4}\mathsf{0}\} \qquad (\sharp 7.12') \qquad\qquad \mathsf{d}(x) \xhookrightarrow{t} \{^{1}\mathsf{c}(x,x)\} \qquad (\sharp 7.14)$$

which we obtained for the $\mathtt{AST}(\xrightarrow{i}_{\mathcal{P}_{\mathsf{dup}}})$ proof of $\mathcal{P}_{\mathsf{dup}}$ in Example 7.4.13 has no subterms below annotations. So neither rule is usable and we set their flags to $\mathsf{f}$ which leads to

$$\mathsf{g} \xhookrightarrow{f} \{^{3/4}\mathsf{d}(\mathsf{g}^\sharp),\ ^{1/4}\mathsf{0}\} \qquad (\sharp 7.12'') \qquad\qquad \mathsf{d}(x) \xhookrightarrow{f} \{^{1}\mathsf{c}(x,x)\} \qquad (\sharp 7.14')$$

Example 7.4.16. Consider the ADP problem $\{(\sharp 7.5'), (\sharp 7.6'), \flat(\sharp 7.7)\text{–}\flat(\sharp 7.11)\}$ from Example 7.4.10 with

$$\mathsf{loop1}(y) \xhookrightarrow{t} \{^{1/2}\mathsf{loop1}^\sharp(\mathsf{double}(y)),\ ^{1/2}\mathsf{loop2}(\mathsf{double}(y))\} \qquad (\sharp 7.5')$$
$$\mathsf{loop1}(y) \xhookrightarrow{t} \{^{1/3}\mathsf{loop1}^\sharp(\mathsf{triple}(y)),\ ^{2/3}\mathsf{loop2}(\mathsf{triple}(y))\} \qquad (\sharp 7.6')$$

$$\mathsf{loop2}(\mathsf{s}(y)) \xhookrightarrow{t} \{^{1}\mathsf{loop2}(y)\} \quad (\flat(\sharp 7.7)) \qquad \mathsf{triple}(\mathsf{s}(y)) \xhookrightarrow{t} \{^{1}\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{triple}(y))))\} \quad (\flat(\sharp 7.10))$$
$$\mathsf{double}(\mathsf{s}(y)) \xhookrightarrow{t} \{^{1}\mathsf{s}(\mathsf{s}(\mathsf{double}(y)))\} \quad (\flat(\sharp 7.8)) \qquad \mathsf{triple}(\mathsf{0}) \xhookrightarrow{t} \{^{1}\mathsf{0}\} \quad (\flat(\sharp 7.11))$$
$$\mathsf{double}(\mathsf{0}) \xhookrightarrow{t} \{^{1}\mathsf{0}\} \quad (\flat(\sharp 7.9))$$

that resulted from applying the innermost dependency graph processor and the innermost usable terms processor to $\mathcal{DP}(\mathcal{P}_{\mathsf{alg}})$. When proving $\mathtt{AST}(\xrightarrow{i}_{\mathcal{P}_{\mathsf{alg}}})$ of $\mathcal{P}_{\mathsf{alg}}$, only the $\mathsf{double}$- and $\mathsf{triple}$-ADPs $\flat(\sharp 7.8)$–$\flat(\sharp 7.11)$ are usable, since the defined symbols $\mathsf{double}$ and $\mathsf{triple}$ occur below an annotated symbol $\mathsf{loop1}^\sharp$. So we can set the flag of all other ADPs in this problem to $\mathsf{f}$. This yields

$$\mathsf{loop1}(y) \xhookrightarrow{f} \{^{1/2}\mathsf{loop1}^\sharp(\mathsf{double}(y)),\ ^{1/2}\mathsf{loop2}(\mathsf{double}(y))\} \qquad (\sharp 7.5'')$$
$$\mathsf{loop1}(y) \xhookrightarrow{f} \{^{1/3}\mathsf{loop1}^\sharp(\mathsf{triple}(y)),\ ^{2/3}\mathsf{loop2}(\mathsf{triple}(y))\} \qquad (\sharp 7.6'')$$

$$\mathsf{loop2}(\mathsf{s}(y)) \xhookrightarrow{f} \{^{1}\mathsf{loop2}(y)\} \quad (\flat(\sharp 7.7)') \qquad \mathsf{triple}(\mathsf{s}(y)) \xhookrightarrow{t} \{^{1}\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{triple}(y))))\} \quad (\flat(\sharp 7.10))$$
$$\mathsf{double}(\mathsf{s}(y)) \xhookrightarrow{t} \{^{1}\mathsf{s}(\mathsf{s}(\mathsf{double}(y)))\} \quad (\flat(\sharp 7.8)) \qquad \mathsf{triple}(\mathsf{0}) \xhookrightarrow{t} \{^{1}\mathsf{0}\} \quad (\flat(\sharp 7.11))$$
$$\mathsf{double}(\mathsf{0}) \xhookrightarrow{t} \{^{1}\mathsf{0}\} \quad (\flat(\sharp 7.9))$$

All other ADP problems resulting from the innermost dependency graph and the innermost usable terms processor in this example have no usable rules, hence we can turn the flag of all ADPs in these ADP problems to $\mathsf{f}$.

**Usable Rules Processor for $\mathtt{AST}$** For full rewriting, the usable rules processor is unsound. This is already the case for non-probabilistic rewriting, but in the classical DP framework there nevertheless exist processors for full rewriting based on usable rules which rely on taking the $C_{\mathcal{E}}$-rules $\mathsf{h}(x,y) \to x$ and $\mathsf{h}(x,y) \to y$ for a fresh function symbol $\mathsf{h}$ into account, see, e.g., [GTSF06, GTS04, HM05, Urb04]. However, the following counterexample shows that this is not possible for $\mathtt{AST}$.

Counterexample 7.4.17. Consider the ADP problem with the two ADPs

$$\mathsf{a} \xhookrightarrow{t} \{^{5/8}\mathsf{c}(\mathsf{a},\mathsf{a}),\ ^{3/8}\mathsf{0}\} \qquad\qquad \mathsf{f}(\mathsf{c}(x_1,x_2)) \xhookrightarrow{t} \{^{1}\mathsf{c}(\mathsf{f}^\sharp(x_1),\mathsf{f}^\sharp(x_2))\}$$

Although the first ADP has no annotations, the ADP problem is not $\mathtt{AST}$:

$$^{1}\mathsf{f}^\sharp(\underline{\mathsf{a}}) \longrightarrow {}^{5/8}\underline{\mathsf{f}^\sharp}(\mathsf{c}(\mathsf{a},\mathsf{a})) \longrightarrow {}^{5/8}\mathsf{c}(\mathsf{f}^\sharp(\mathsf{a}),\mathsf{f}^\sharp(\mathsf{a})) \to \cdots \qquad {}^{3/8}\mathsf{f}^\sharp(\mathsf{0})$$

This represents a positively drifting random walk, where the number of $\mathsf{f}^\sharp(\mathsf{a})$ subterms increases by one with probability $5/8$ or decreases by one with probability $3/8$.

The ADP problem has no usable rules and thus, the usable rules processor would turn the flag of all ADPs to $\mathsf{f}$. However, if we now start with $\mathsf{f}^\sharp(\mathsf{a})$ (or $\mathsf{f}^\sharp(\mathsf{a}^\sharp)$), rewriting $\mathsf{a}$ (or $\mathsf{a}^\sharp$) would remove the annotation of $\mathsf{f}^\sharp$ above, since the flag indicates whether the ADP can be used to rewrite arguments of an annotated symbol, i.e., $\underline{\mathsf{f}^\sharp}(\mathsf{a}) \hookrightarrow \{^{5/8}\mathsf{f}(\mathsf{c}(\mathsf{a},\mathsf{a})), {}^{3/8}\mathsf{f}(\mathsf{0})\}$. Hence, all CTs are finite. This also holds when adding the $C_\mathcal{E}$-ADPs $\mathsf{h}(x,y) \overset{\mathsf{t}}{\hookrightarrow} \{^1x\}$ and $\mathsf{h}(x,y) \overset{\mathsf{t}}{\hookrightarrow} \{^1y\}$.

Thus, even integrating the $C_\mathcal{E}$-rules to represent non-determinism would not allow a usable rules processor for $\mathtt{AST}$. Moreover, the corresponding proofs in the non-probabilistic setting rely on the minimality property, which does not hold in the probabilistic setting, see Counterexample 5.3.1.

### 7.4.4 Probability Removal Processor

In proofs with the ADP framework, one may obtain ADP problems $\mathcal{A}$ with a non-probabilistic structure, i.e., where every ADP has the form $\ell \overset{m}{\hookrightarrow} \{^1r\}$. Then the *probability removal processor* allows us to switch to ordinary (non-probabilistic) DPs.[6]

> Theorem 7.4.18 – Probability Removal Processor for $\mathtt{AST}$ & $\mathtt{iAST}$.
>
> Let $\mathcal{A}$ be an ADP problem where every ADP in $\mathcal{A}$ has the form $\ell \overset{m}{\hookrightarrow} \{^1r\}$. Let $\mathrm{dp}(\mathcal{A}) = \{\ell^\sharp \to t^\sharp \mid \ell \overset{m}{\hookrightarrow} \{^1r\} \in \mathcal{A}, t \trianglelefteq_\sharp r\}$. Then $\mathcal{A}$ is $(\mathtt{i})\mathtt{AST}$ if the non-probabilistic DP problem $(\mathrm{dp}(\mathcal{A}), \mathrm{np}(\mathcal{A}))$ is (innermost) terminating.[6] So the processor $\mathrm{Proc}_{\mathtt{PR}}(\mathcal{A}) = \mathrm{Proc}^{\mathtt{i}}_{\mathtt{PR}}(\mathcal{A}) = \varnothing$ is sound and complete for $\mathtt{AST}$ and $\mathtt{iAST}$ if $(\mathrm{dp}(\mathcal{A}), \mathrm{np}(\mathcal{A}))$ is $(\mathtt{i})\mathtt{SN}$. For $\mathtt{AST}$, we even have "iff".

Note that Theorem 7.4.18 requires the absence of *all* infinite (innermost) chains, not only of minimal ones (cf. Remark 7.2.21). So when proving that $(\mathrm{dp}(\mathcal{A}), \mathrm{np}(\mathcal{A}))$ is $(\mathtt{i})\mathtt{SN}$ with the non-probabilistic DP framework, one must not use processors that rely on the minimality of chains.

*Proof.* Let $\mathcal{A}$ be an ADP problem such that every ADP in $\mathcal{A}$ has the form $\ell \overset{m}{\hookrightarrow} \{^1r\}$. Note that every $\mathcal{A}$-CT is a single (not necessarily finite) path. For such a CT $\mathfrak{T}$ that is only a single path, we have only two possibilities for $|\mathfrak{T}|$. If the path is finite, then $|\mathfrak{T}| = 1$, since we have a single leaf in this tree with probability 1. Otherwise, we have an infinite path, which means that there is no leaf at all and hence $|\mathfrak{T}| = 0$. Completeness of the processor holds trivially, since it always returns the empty set. In the following, we first prove the "only if" direction for $\mathtt{AST}$ (and the corresponding statement for $\mathtt{iAST}$ from the footnote of Theorem 7.4.18), and afterwards the "if" direction, i.e., the soundness of the processor.

*Proof* (

▶ *Proof* ($\Longrightarrow$, "*Only If*", $\mathtt{AST}$): We first consider $\mathtt{AST}$. Assume that $(\mathrm{dp}(\mathcal{A}), \mathrm{np}(\mathcal{A}))$ is not terminating. Then there exists an infinite $(\mathrm{dp}(\mathcal{A}), \mathrm{np}(\mathcal{A}))$-chain

$$t_0^\sharp \xrightarrow{\varepsilon}_{\mathrm{dp}(\mathcal{A})} \circ \to^*_{\mathrm{np}(\mathcal{A})} t_1^\sharp \xrightarrow{\varepsilon}_{\mathrm{dp}(\mathcal{A})} \circ \to^*_{\mathrm{np}(\mathcal{A})} t_2^\sharp \xrightarrow{\varepsilon}_{\mathrm{dp}(\mathcal{A})} \circ \to^*_{\mathrm{np}(\mathcal{A})} \cdots$$

[6] For $\mathtt{AST}(\overset{\mathtt{i}}{\hookrightarrow}_{\mathcal{A}})$, we obtain an "iff" for the probability removal processor when additionally handing over the set of left-hand sides to the dependency pair framework. The reason is that similar to the usable rules processor, the left-hand sides of all rules are needed to determine whether a rewrite step is innermost.

such that for all $i \in \mathbb{N}$ we have $t_i^\sharp = \ell_i^\sharp \sigma_i$ for some $\ell_i^\sharp \to r_i^\sharp \in \mathrm{dp}(\mathcal{A})$ and some substitution $\sigma_i$. From this infinite $(\mathrm{dp}(\mathcal{A}), \mathrm{np}(\mathcal{A}))$-chain, we will now construct an infinite $\mathcal{A}$-CT $\mathfrak{T} = (N, E)$. As explained above, we then know that this infinite $\mathcal{A}$-CT must be an infinite path, and thus $|\mathfrak{T}| = 0$. This means that $\mathcal{A}$ is not $\mathtt{AST}$, and thus, a processor with $\mathrm{Proc}_{\mathtt{PR}}(\mathcal{A}) = \varnothing$ would be unsound.




We start our CT with $t_0^\sharp$. In the non-probabilistic rewrite sequence, we have $t_0^\sharp \xrightarrow{\varepsilon}_{\mathrm{dp}(\mathcal{A})} \circ \to^*_{\mathrm{np}(\mathcal{A})} t_1^\sharp$, so there exists a natural number $k \geq 1$ such that

$$t_0^\sharp = \ell_0^\sharp\sigma_0 \to_{\mathrm{dp}(\mathcal{A})} r_0^\sharp\sigma_0 = s_1^\sharp \to_{\mathrm{np}(\mathcal{A})} s_2^\sharp \to_{\mathrm{np}(\mathcal{A})} \ldots \to_{\mathrm{np}(\mathcal{A})} s_k^\sharp = t_1^\sharp = \ell_1^\sharp\sigma_1.$$

Performing the same rewrite steps with $\mathcal{A}$ yields terms $a_1, \ldots, a_k$ such that $s_i \trianglelefteq_\sharp a_i$ for all $1 \leq i \leq k$. Here, one needs that all ADPs that yield the rules in $\mathrm{np}(\mathcal{A})$ have the flag $\mathtt{t}$ and thus, the annotations above the redex are not removed. For all $1 \leq i \leq k$, we now construct $a_i$ inductively.

In the induction base ($i = 1$), let $\ell_0 \xhookrightarrow{m} \{{}^1 r_0'\} \in \mathcal{A}$ be the ADP that was used to create the dependency pair $\ell_0^\sharp \to r_0^\sharp$ in $\mathrm{dp}(\mathcal{A})$. This means that we have $r_0 \trianglelefteq_\sharp r_0'$. Since we have $t_0^\sharp = \ell_0^\sharp\sigma_0$, we can also rewrite $t_0^\sharp$ with the ADP $\ell_0 \xhookrightarrow{m} \{{}^1 r_0'\} \in \mathcal{A}$ and the substitution $\sigma_0$ at the root position. We obtain $a_1 = r_0'\sigma_0$ and thus we have $s_1 = r_0\sigma_0 \trianglelefteq_\sharp r_0'\sigma_0 = a_1$.

In the induction step, we assume that $s_i \trianglelefteq_\sharp a_i$ for some $1 \leq i < k$. Let $\pi$ be the annotated position of $a_i$ where $s_i = \flat(a_i)|_\pi$. In our non-probabilistic rewrite sequence we have $s_i^\sharp \to_{\mathrm{np}(\mathcal{A})} s_{i+1}^\sharp$ using a rule $\ell' \to \flat(r') \in \mathrm{np}(\mathcal{A})$ and substitution $\delta_i$ at a position $\tau \neq \varepsilon$ such that $s_i^\sharp|_\tau = \ell'\delta_i$ and $s_{i+1}^\sharp = s_i^\sharp[\flat(r')\delta_i]_\tau$. We can mirror this rewrite step with the ADP $\ell' \xhookrightarrow{\mathtt{t}} \{{}^1 r'\} \in \mathcal{A}$, since by construction we have $s_i \trianglelefteq_\sharp a_i$ and $s_i = \flat(a_i)|_\pi$. We obtain $a_i \hookrightarrow_{\mathcal{A}} a_{i+1} = a_i[\sharp_\Phi(r'\delta_i)]_{\pi.\tau}$ with $\Phi = \mathrm{pos}_{\mathcal{D}^\sharp}(r') \cup \Psi_1$ (step of the form **at**) or $\Phi = \Psi_1$ (step of the form **nt**) by rewriting the subterm of $a_i$ at position $\pi.\tau$, which implies $s_{i+1} = s_i[\flat(r')\delta_i]_\tau \trianglelefteq_\sharp a_i[\sharp_\Phi(r'\delta_i)]_{\pi.\tau} = a_{i+1}$.

At the end of this induction, we obtain $a_k$. Next, we can mirror the step $t_1^\sharp \xrightarrow{\varepsilon}_{\mathrm{dp}(\mathcal{A})} \circ \to^*_{\mathrm{np}(\mathcal{A})} t_2^\sharp$ from our non-probabilistic rewrite sequence with the same construction, etc. This results in an infinite $\mathcal{A}$-CT. To see that this is indeed an $\mathcal{A}$-CT, note that all the local properties are satisfied since every edge represents a rewrite step with $\hookrightarrow_{\mathcal{A}}$. The liveness property is also satisfied since, in an infinite $(\mathrm{dp}(\mathcal{A}), \mathrm{np}(\mathcal{A}))$-chain, we use an infinite number of steps with $\to_{\mathrm{dp}(\mathcal{A})}$ so that our resulting CT has an infinite number of Γ-nodes.

▶ *Proof* ($\Longrightarrow$, "*Only If*", $\mathtt{iAST}$): As stated in the footnote of Theorem 7.4.18, for $\mathtt{AST}(\xhookrightarrow{\mathtt{i}}_{\mathcal{A}})$ this direction holds if one additionally hands over the left-hand sides of $\mathcal{A}$ to determine innermost evaluation. Apart from that, we just have to consider innermost evaluation, while the construction remains the same. More precisely, we start with an infinite innermost $(\mathrm{dp}(\mathcal{A}), \mathrm{np}_{\mathrm{i}}(\mathcal{A}))$-chain

$$t_0^\sharp \xrightarrow{\mathtt{i}\,\varepsilon}_{\mathrm{dp}(\mathcal{A})} \circ \xrightarrow{\mathtt{i}}{}^*_{\mathrm{np}_{\mathrm{i}}(\mathcal{A})} t_1^\sharp \xrightarrow{\mathtt{i}\,\varepsilon}_{\mathrm{dp}(\mathcal{A})} \circ \xrightarrow{\mathtt{i}}{}^*_{\mathrm{np}_{\mathrm{i}}(\mathcal{A})} t_2^\sharp \xrightarrow{\mathtt{i}\,\varepsilon}_{\mathrm{dp}(\mathcal{A})} \circ \xrightarrow{\mathtt{i}}{}^*_{\mathrm{np}_{\mathrm{i}}(\mathcal{A})} \cdots,$$

where innermost evaluation is determined by the left-hand sides of all ADPs in $\mathcal{A}$: $\xrightarrow{\mathtt{i}}_{\mathrm{np}_{\mathrm{i}}(\mathcal{A})}$ denotes rewriting with the rules of $\mathrm{np}(\mathcal{A})$ restricted to redexes $\ell\delta$ with $\ell\delta \in \mathtt{ANF}_{\mathcal{A}}$, and $\xrightarrow{\mathtt{i}\,\varepsilon}_{\mathrm{dp}(\mathcal{A})}$ denotes rewriting with $\mathrm{dp}(\mathcal{A})$ at the root restricted to redexes

$\ell^\sharp\delta$ with $\ell\delta \in \mathtt{ANF}_\mathcal{A}$. and construct an innermost $\mathcal{A}$-CT. But since all the steps in the chain are innermost and since innermost evaluation w.r.t. $\mathrm{np}_\mathsf{i}(\mathcal{A})$ is determined by $\mathtt{ANF}_\mathcal{A}$, i.e., by the left-hand sides of all ADPs in $\mathcal{A}$, the steps in the CT are innermost as well.

▶ *Proof*($\Longleftarrow$, *Soundness*, AST & iAST): Assume that $\mathcal{A}$ is not AST, i.e., that the processor $\mathrm{Proc}_{\mathtt{PR}}(\mathcal{A}) = \varnothing$ is unsound. By Lemma 7.3.3, there exists an $\mathcal{A}$-CT $\mathfrak{T} = (N, E)$ that converges with probability $< 1$ and starts with a term $b$ such that $\mathrm{pos}_{\mathcal{D}^\sharp}(b) = \{\pi_0\}$ and whose root performs its step at the position $\pi_0$. As explained above, this tree must be an infinite path, and by the liveness property, it contains an infinite number of nodes in $\Gamma$.

From $\mathfrak{T}$, we will now construct an infinite $(\mathrm{dp}(\mathcal{A}), \mathrm{np}(\mathcal{A}))$-chain, which shows that $(\mathrm{dp}(\mathcal{A}), \mathrm{np}(\mathcal{A}))$ is not terminating. We have $\flat(b|_{\pi_0}) = \ell_0\sigma_0$ for the ADP $\ell_0 \xrightarrow{m} \{{}^1r_0'\} \in \mathcal{A}$ used at the root and some substitution $\sigma_0$ without annotations. We start our infinite chain with the term $t_0^\sharp = \ell_0^\sharp\sigma_0 = b|_{\pi_0}$. For the term $a_1$ of the successor node in $\mathfrak{T}$, we have $a_1 = b[\sharp_{\mathrm{pos}_{\mathcal{D}^\sharp}(r_0')\cup\Psi_1}(r_0'\sigma_0)]_{\pi_0} = b[r_0'\sigma_0]_{\pi_0}$.

There must be a term $r_0 \trianglelefteq_\sharp r_0'$ (i.e., $r_0^\sharp\sigma_0 \trianglelefteq r_0'\sigma_0$) such that if we replace $a_1 = b[r_0'\sigma_0]_{\pi_0}$ with $r_0^\sharp\sigma_0$ and obtain the same $\mathcal{A}$-CT except when we would rewrite terms that do not exist anymore (i.e., we ignore these rewrite steps), then we still end up in an infinite number of nodes in $\Gamma$ (otherwise, $\mathfrak{T}$ would not have an infinite number of nodes in $\Gamma$). Let $\pi$ be the annotated position of $a_1$ where $r_0\sigma_0 = \flat(a_1)|_\pi$. Rewrite steps strictly above $\pi$ may move this subterm to another position, and then $\pi$ denotes its new position from then on. We can rewrite the term $t_0^\sharp$ with the dependency pair $\ell_0^\sharp \to r_0^\sharp \in \mathrm{dp}(\mathcal{A})$, using the substitution $\sigma_0$ since $\flat(t_0) = \ell_0\sigma_0$. Hence, we obtain $t_0^\sharp \to_{\mathrm{dp}(\mathcal{A})} r_0^\sharp\sigma_0 = s_1^\sharp$. Next, we mirror the rewrite steps from the $\mathcal{A}$-CT that are performed strictly below the root of $r_0^\sharp\sigma_0$ with $\mathrm{np}(\mathcal{A})$ until we would rewrite at the root of $r_0^\sharp\sigma_0$. With this construction, we ensure that each term $s_i^\sharp$ in our $(\mathrm{dp}(\mathcal{A}), \mathrm{np}(\mathcal{A}))$-chain satisfies $s_i \trianglelefteq_\sharp a_i$ and $s_i = \flat(a_i)|_\pi$. A rewrite step at position $\pi.\tau$ in $a_i$ with $\mathcal{A}$ is then mirrored with $\mathrm{np}(\mathcal{A})$ in $s_i^\sharp$ at position $\tau$. All these steps indeed use ADPs with flag t, whose rules are contained in $\mathrm{np}(\mathcal{A})$, as otherwise, we would remove the annotation at position $\pi$. Note that we only use a finite number of $\mathrm{np}(\mathcal{A})$ steps until we rewrite at the root, since the next node in $\Gamma$ must perform its step at the annotated position $\pi$.

So eventually, we obtain a term $t_1^\sharp = s_k^\sharp$ with $s_k = \flat(a_k)|_\pi$, and we rewrite at position $\pi$ in the $\mathcal{A}$-CT. We mirror the step $a_k \hookrightarrow_\mathcal{A} a_{k+1}$ with $\mathrm{dp}(\mathcal{A})$ and then use the same construction again until we reach term $t_2^\sharp$, etc. This construction creates a sequence $t_0^\sharp, t_1^\sharp, \ldots$ of terms such that

$$t_0^\sharp \;\to_{\mathrm{dp}(\mathcal{A})}\; r_0^\sharp\sigma_0 \;\to^*_{\mathrm{np}(\mathcal{A})}\; t_1^\sharp \;\to_{\mathrm{dp}(\mathcal{A})}\; r_1^\sharp\sigma_1 \;\to^*_{\mathrm{np}(\mathcal{A})}\; \ldots.$$

Therefore, $(\mathrm{dp}(\mathcal{A}), \mathrm{np}(\mathcal{A}))$ is not terminating.

For iAST, the same construction can be used: if $\mathfrak{T}$ is an innermost $\mathcal{A}$-CT, then every redex that is rewritten in $\mathfrak{T}$ is in $\mathtt{ANF}_\mathcal{A}$, and every term in $\mathtt{ANF}_\mathcal{A}$ is also in argument normal form w.r.t. $\mathrm{np}(\mathcal{A})$, since the rules of $\mathrm{np}(\mathcal{A})$ result from ADPs of $\mathcal{A}$ by removing their annotations. Hence, all mirrored rewrite steps are innermost as well, and we obtain an infinite innermost $(\mathrm{dp}(\mathcal{A}), \mathrm{np}(\mathcal{A}))$-chain, i.e., $(\mathrm{dp}(\mathcal{A}), \mathrm{np}(\mathcal{A}))$ is not iSN. *QED*

We illustrate the probability removal processor using the following example.

Example 7.4.19. To illustrate the probability removal processor, consider the PTRS $\mathcal{P}_{\mathsf{ag}}$ that combines the probabilistic geometric rule from Example 3.4.3 with the (non-probabilistic) Ackermann function:

$$\mathsf{geo}(x) \to \{{}^{1/2}\mathsf{geo}(\mathsf{s}(x)),\ {}^{1/2}x\} \quad (7.33) \qquad \mathsf{ack}(\mathsf{s}(x),0) \to \{{}^{1}\mathsf{ack}(x,\mathsf{s}(0))\} \quad (7.35)$$

$$\mathsf{ack}(0,y) \to \{{}^{1}\mathsf{s}(y)\} \quad (7.34) \qquad \mathsf{ack}(\mathsf{s}(x),\mathsf{s}(y)) \to \{{}^{1}\mathsf{ack}(x,\mathsf{ack}(\mathsf{s}(x),y))\} \quad (7.36)$$

We prove $\mathtt{AST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}_{\mathsf{ag}}})$ and hence start with the canonical ADP problem $\mathcal{DP}(\mathcal{P}_{\mathsf{ag}})$, where all defined symbols in the right-hand sides are annotated:

$$\mathsf{geo}(x) \xhookrightarrow{\mathtt{t}} \{{}^{1/2}\mathsf{geo}^\sharp(\mathsf{s}(x)),\ {}^{1/2}x\} \quad (\sharp 7.33) \qquad \mathsf{ack}(\mathsf{s}(x),0) \xhookrightarrow{\mathtt{t}} \{{}^{1}\mathsf{ack}^\sharp(x,\mathsf{s}(0))\} \quad (\sharp 7.35)$$

$$\mathsf{ack}(0,y) \xhookrightarrow{\mathtt{t}} \{{}^{1}\mathsf{s}(y)\} \quad (\sharp 7.34) \qquad \mathsf{ack}(\mathsf{s}(x),\mathsf{s}(y)) \xhookrightarrow{\mathtt{t}} \{{}^{1}\mathsf{ack}^\sharp(x,\mathsf{ack}^\sharp(\mathsf{s}(x),y))\} \quad (\sharp 7.36)$$

The innermost dependency graph (Definition 7.4.5) of $\mathcal{DP}(\mathcal{P}_{\mathsf{ag}})$ is shown in Figure 7.14. Its only non-trivial SCCs are $\{(\sharp 7.33)\}$ and $\{(\sharp 7.35),(\sharp 7.36)\}$, whereas $(\sharp 7.34)$ carries no annotation and thus lies on no cycle. Let us focus on the SCC $\{(\sharp 7.35),(\sharp 7.36)\}$. The dependency graph processor keeps the annotations of these two ADPs and removes all other annotations, yielding the ADP problem $\mathcal{A} = \{(\sharp 7.35),(\sharp 7.36),\flat(\sharp 7.33),\flat(\sharp 7.34)\}$.

Next, we apply the usable rules processor (Theorem 7.4.14). The only defined symbol reachable from the annotated subterms of $\mathcal{A}$ is $\mathsf{ack}$, so the usable rules $\mathcal{U}(\mathcal{A})$ are exactly the three $\mathsf{ack}$-ADPs, while $\flat(\sharp 7.33)$ is *not* usable. Hence we may turn its flag to $\mathsf{f}$, resulting in $\mathsf{geo}(x) \xhookrightarrow{\mathsf{f}} \{{}^{1/2}\mathsf{geo}(\mathsf{s}(x)),\ {}^{1/2}x\}$. Now this ADP has the flag $\mathsf{f}$ and no annotations, i.e., it has become irrelevant. By Theorem 7.3.5, for $\mathtt{iAST}$ we may replace it by $\mathsf{geo}(x) \xhookrightarrow{\mathsf{f}} \{{}^{1}\mathsf{stop}\}$ for a fresh constant $\mathsf{stop}$, thereby removing the only non-trivial probabilities from the problem.

At this point, every ADP in $\mathcal{A}$ has the form $\ell \xhookrightarrow{m} \{{}^{1}r\}$, so the probability removal processor (Theorem 7.4.18) applies. Collecting the annotated subterms of the right-hand sides yields the dependency pairs

$$\begin{aligned}\mathsf{ack}^\sharp(\mathsf{s}(x),0) &\to \mathsf{ack}^\sharp(x,\mathsf{s}(0))\\ \mathsf{ack}^\sharp(\mathsf{s}(x),\mathsf{s}(y)) &\to \mathsf{ack}^\sharp(x,\mathsf{ack}(\mathsf{s}(x),y))\\ \mathsf{ack}^\sharp(\mathsf{s}(x),\mathsf{s}(y)) &\to \mathsf{ack}^\sharp(\mathsf{s}(x),y)\end{aligned}$$

together with the underlying rules $\mathrm{np}(\mathcal{A}) = \{\mathsf{ack}(0,y) \to \mathsf{s}(y),\ \mathsf{ack}(\mathsf{s}(x),0) \to \mathsf{ack}(x,\mathsf{s}(0)),\ \mathsf{ack}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{ack}(x,\mathsf{ack}(\mathsf{s}(x),y))\}$ (note that the $\mathsf{geo}$-ADP has the flag $\mathsf{f}$ and is therefore not contained in $\mathrm{np}(\mathcal{A})$). This is precisely the standard (innermost) DP problem of the Ackermann function, which is $\mathtt{iSN}$ and can be solved by the non-probabilistic DP framework, e.g., via a reduction pair that compares the arguments of $\mathsf{ack}$ lexicographically. Therefore, $\mathrm{Proc}^{\mathtt{i}}_{\mathrm{PR}}(\mathcal{A}) = \varnothing$, i.e., the ADP problem $\mathcal{A}$ is $\mathtt{iAST}$. The ADP problem corresponding to the remaining SCC $\{(\sharp 7.33)\}$ can be solved by the probabilistic reduction pair processor (Theorem 7.5.1). Hence, we can conclude $\mathtt{AST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}_{\mathsf{ag}}})$.

### 7.4.5 Switching From Full to Innermost AST

In the non-probabilistic DP framework, there is a processor to switch from full to innermost evaluation if the DP problem satisfies certain conditions [GTS04, Thm. 32], e.g., if the rules and DPs are non-overlapping. This is useful as the DP framework for innermost termination is more powerful than the one for full termination and in this way, one can switch to the innermost case for certain sub-problems, even if the whole TRS does not belong to any class where innermost termination implies termination [Gra95]. As we showed in Theorem 4.2.2, to ensure that $\mathtt{AST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}})$ implies $\mathtt{AST}(\to_{\mathcal{P}})$ for PTRSs, in addition to non-overlappingness, we required linearity of the rules. However, in the

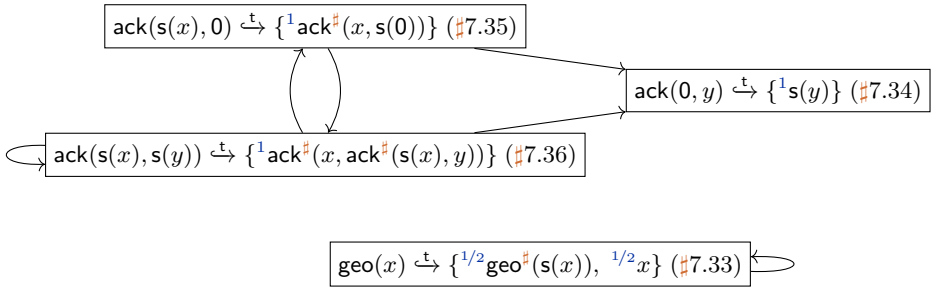



Figure 7.14: Innermost $\mathcal{DP}(\mathcal{P}_{\mathsf{ag}})$-dependency graph.

non-probabilistic setting, the soundness of this processor relies again on the minimality property which does not hold in the probabilistic setting (Counterexample 5.3.1). Indeed, the following counterexample which corresponds to [Thi07, Ex. 3.15] shows that a similar processor in the ADP framework would be unsound, even if we require that all ADPs are non-overlapping and linear. Non-overlappingness of ADP problems is defined as for TRSs, as both kinds of rules contain only a single left-hand side.

Counterexample 7.4.20. The ADP problem containing $\mathsf{f}(x) \overset{\mathsf{t}}{\hookrightarrow} \{^1\mathsf{f}^\sharp(\mathsf{a})\}$ and $\mathsf{a} \overset{\mathsf{t}}{\hookrightarrow} \{^1\mathsf{a}\}$ is not $\mathtt{AST}$ as we can rewrite $\mathsf{f}^\sharp(\mathsf{a})$ to itself with the $\mathsf{f}$-ADP. However, $\mathtt{AST}(\overset{\mathsf{i}}{\hookrightarrow}_{\mathcal{A}})$ holds since in innermost evaluations, we have to rewrite the inner $\mathsf{a}$, which does not terminate but does not use any annotations, i.e., any **at** or **af** steps. Note that the ADPs are non-overlapping and linear. Thus, a processor to switch from full to innermost $\mathtt{AST}$ cannot be applied on the level of ADP problems.

Hence, to prove $\mathtt{AST}$ of a PTRS $\mathcal{P}$ that belongs to the classes of Theorem 4.2.2 where $\mathtt{AST}(\overset{\mathsf{i}}{\to}_{\mathcal{P}})$ implies $\mathtt{AST}(\to_{\mathcal{P}})$, one should apply the ADP framework for $\mathtt{iAST}$, because its processors are more powerful. But otherwise, one has to use the ADP framework for $\mathtt{AST}$, which requires that $\mathcal{P}$ is $\mathtt{ND}$ (Theorem 7.2.19).

### 7.4.6 Switching Back to PTRSs

Throughout this chapter, the ADP framework refines a PTRS into ADP problems in order to apply modular termination techniques. However, sometimes it is beneficial to take the opposite direction and switch *back* from an ADP problem to a PTRS. The reason is that on the level of PTRSs we may apply termination techniques that are not (or not yet) available inside the ADP framework. The most prominent advantage of such a transformation is that it gives us access to the decision procedure for the subclass RNT of right-ground, non-overlapping, and tail-recursive PTRSs from Chapter 6. This is particularly useful as a *closing* processor at the leaves of a proof tree, e.g., after irrelevant ADPs have been replaced by trivially terminating ones (Theorem 7.3.5) or after transformational processors (Section 7.6) have been applied, the underlying PTRS of a remaining ADP problem may well fall into the decidable subclass even if the original PTRS did not.

Before switching back, it is often beneficial for $\mathtt{iAST}$ to first apply the processor for removing irrelevant ADPs (Theorem 7.3.5). Replacing ADPs with flag $\mathsf{f}$ and without annotations by ADPs with the right-hand side $\{^1\mathsf{stop}\}$ simplifies the PTRS we switch back to and may be exactly what is needed to bring it into a decidable subclass such as RNT.

During the switch back to PTRSs, we associate to every ADP its *underlying rule*, which is obtained by removing all annotations and flags.

**Definition 7.4.21** – Underlying PTRS.

*Let $\mathcal{A}$ be an ADP problem. Its underlying PTRS $\mathcal{DP}^{-1}(\mathcal{A})$ is defined as*

$$\mathcal{DP}^{-1}(\mathcal{A}) = \{\ell \to \{{}^{p_1}\flat(r_1), \dots, {}^{p_k}\flat(r_k)\} \mid \ell \xhookrightarrow{m} \{{}^{p_1}r_1, \dots, {}^{p_k}r_k\} \in \mathcal{A}\}.$$

Note that $\mathcal{DP}^{-1}(\mathcal{DP}(\mathcal{P})) = \mathcal{P}$ for every PTRS $\mathcal{P}$, i.e., the underlying PTRS of a canonical ADP problem is the original PTRS, which justifies the notation $\mathcal{DP}^{-1}(\,\cdot\,)$ as the inverse of the canonical ADP transformation $\mathcal{DP}(\,\cdot\,)$. The crucial observation for soundness is that rewriting with an ADP problem $\mathcal{A}$ only manipulates annotations (depending on the flags), but performs the very same rewrite steps as the underlying PTRS $\mathcal{DP}^{-1}(\mathcal{A})$. Hence, dropping all annotations turns every (innermost) $\mathcal{A}$-CT into an (innermost) $\mathcal{DP}^{-1}(\mathcal{A})$-RT with the same tree structure and the same probabilities. Consequently, if $\mathcal{DP}^{-1}(\mathcal{A})$ is (i)AST, then so is $\mathcal{A}$. This yields the following processor, which works for both AST and iAST.

**Theorem 7.4.22** – Switching Back Processor for AST & iAST.

Let $\mathcal{A}$ be an ADP problem. Then $\mathcal{A}$ is (i)AST if its underlying PTRS $\mathcal{DP}^{-1}(\mathcal{A})$ is (i)AST. So the processor $\mathrm{Proc}_{\mathtt{back}}(\mathcal{A}) = \mathrm{Proc}^{\mathtt{i}}_{\mathtt{back}}(\mathcal{A}) = \varnothing$ is sound and complete for AST and iAST if $\mathcal{DP}^{-1}(\mathcal{A})$ is (i)AST.

*Proof.* ▶ *Completeness*: Whenever the processor is applied, it returns $\mathrm{Proc}_{\mathtt{back}}(\mathcal{A}) = \varnothing$, so completeness holds trivially. The same holds for $\mathrm{Proc}^{\mathtt{i}}_{\mathtt{back}}$.

▶ *Soundness*: Soundness is non-trivial only in the case where the processor returns $\varnothing$, i.e., where $\mathcal{DP}^{-1}(\mathcal{A})$ is AST. So assume that $\mathcal{DP}^{-1}(\mathcal{A})$ is AST. We show that then $\mathcal{A}$ is AST (the argument for iAST is identical and indicated in parentheses).
Let $\mathfrak{T} = (N, E)$ be an arbitrary (innermost) $\mathcal{A}$-CT. We construct an (innermost) $\mathcal{DP}^{-1}(\mathcal{A})$-RT $\mathfrak{T}'$ with the same tree structure by relabeling every node $v \in N$ with label ${}^{p_v}t_v$ by ${}^{p_v}\flat(t_v)$. Consider an inner node $v$ with $t_v \hookrightarrow_{\mathcal{A}} \{{}^{p_{w_1}/p_v}t_{w_1}, \dots, {}^{p_{w_k}/p_v}t_{w_k}\}$ using the ADP $\ell \xhookrightarrow{m} \{{}^{p_1}r_1, \dots, {}^{p_k}r_k\} \in \mathcal{A}$ at a position $\pi$ with a VRF $(\varphi_j)_j$ and a substitution $\sigma$ such that $\flat(t_v|_\pi) = \ell\sigma$. By Definition 7.2.11, for each $1 \le j \le k$ the successor $t_{w_j}$ is obtained from $t_v$ by replacing the redex at $\pi$ with a variant of $r_j\sigma$ that differs from $r_j\sigma$ only in its annotations, and possibly by removing annotations strictly above $\pi$ (if $m = \mathtt{f}$). Since all of these operations only add or remove annotations but never change the underlying term, we obtain $\flat(t_{w_j}) = \flat(t_v)[\flat(r_j)\sigma]_\pi$. Therefore,

$$\flat(t_v) \to_{\mathcal{DP}^{-1}(\mathcal{A})} \{{}^{p_{w_1}/p_v}\flat(t_{w_1}), \dots, {}^{p_{w_k}/p_v}\flat(t_{w_k})\}$$

is a valid rewrite step of $\mathcal{DP}^{-1}(\mathcal{A})$ using the rule $\ell \to \{{}^{p_1}\flat(r_1), \dots, {}^{p_k}\flat(r_k)\} \in \mathcal{DP}^{-1}(\mathcal{A})$ at position $\pi$ with substitution $\sigma$. (For an innermost $\mathcal{A}$-CT we additionally have $\flat(t_v|_\pi) = \ell\sigma \in \mathtt{ANF}_{\mathcal{A}}$, so this is an innermost step of $\mathcal{DP}^{-1}(\mathcal{A})$.) Hence $\mathfrak{T}'$ is a valid (innermost) $\mathcal{DP}^{-1}(\mathcal{A})$-RT starting with $\flat(t_\mathfrak{t})$, and since the probabilities of all nodes are preserved, we have $|\mathfrak{T}'| = |\mathfrak{T}|$.
Now, if $\mathcal{DP}^{-1}(\mathcal{A})$ is AST, then every $\to_{\mathcal{DP}^{-1}(\mathcal{A})}$-RT converges with probability 1, so $|\mathfrak{T}'| = 1$ and thus $|\mathfrak{T}| = 1$. As $\mathfrak{T}$ was an arbitrary (innermost) $\mathcal{A}$-CT, this proves $\mathtt{AST}(\hookrightarrow_{\mathcal{A}})$ ($\mathtt{AST}(\xhookrightarrow{\mathtt{i}}_{\mathcal{A}})$), i.e., $\mathrm{Proc}_{\mathtt{back}}$ is sound for AST (and iAST). *QED*

After switching back to the underlying PTRS $\mathcal{DP}^{-1}(\mathcal{A})$, any technique for proving

(i)AST of PTRSs can be used to solve the remaining proof obligation. The key advantage is that $\mathcal{DP}^{-1}(\mathcal{A})$ may fall into the decidable subclass RNT.

Example 7.4.23 (Switching Back to PTRSs). Consider the PTRS $\mathcal{P}_{\mathsf{back}}$ with the rules

$$\mathsf{h}(x) \to \{^{1}\mathsf{c}(\mathsf{f}(x), \mathsf{a})\} \tag{7.37}$$

$$\mathsf{f}(x) \to \{^{1/2}\mathsf{g}, {}^{1/2}\mathsf{c}(\mathsf{f}(\mathsf{a}), \mathsf{f}(\mathsf{b}))\} \tag{7.38}$$

where (7.38) is the rule of the PTRS from Example 6.3.2 and Example 6.3.7. Due to the variable $x$ in the right-hand side of (7.37), the PTRS $\mathcal{P}_{\mathsf{back}}$ is not right-ground, i.e., we have $\mathcal{P}_{\mathsf{back}} \notin$ RNT and our decision procedure is not applicable directly.

To prove $\mathtt{AST}(\xrightarrow{\mathsf{i}}_{\mathcal{P}_{\mathsf{back}}})$, we start with the canonical ADP problem $\mathcal{DP}(\mathcal{P}_{\mathsf{back}})$ consisting of the two ADPs

$$\mathsf{h}(x) \hookrightarrow^{\mathsf{t}} \{^{1}\mathsf{c}(\mathsf{f}^{\sharp}(x), \mathsf{a})\} \tag{♯7.37}$$

$$\mathsf{f}(x) \hookrightarrow^{\mathsf{t}} \{^{1/2}\mathsf{g}, {}^{1/2}\mathsf{c}(\mathsf{f}^{\sharp}(\mathsf{a}), \mathsf{f}^{\sharp}(\mathsf{b}))\} \tag{♯7.38}$$

Since $\mathsf{h}$ does not occur in any right-hand side, the only non-trivial SCC of the innermost dependency graph is {(♯7.38)}. Hence, the innermost dependency graph processor (Theorem 7.4.6) removes the annotations of (♯7.37), yielding the ADP problem $\mathcal{A} = \{(\sharp 7.38), \flat(\sharp 7.37)\}$. Moreover, no defined symbol occurs below the annotations of $\mathcal{A}$ (the arguments of the annotated symbols are the constructor constants $\mathsf{a}$ and $\mathsf{b}$), so no ADP is usable, and the usable rules processor (Theorem 7.4.14) turns the flags of both ADPs to $\mathsf{f}$. Now $\flat(\sharp 7.37)$ has the flag $\mathsf{f}$ and no annotations, i.e., it has become irrelevant, and by Theorem 7.3.5 we may replace it by $\mathsf{h}(x) \hookrightarrow^{\mathsf{f}} \{^{1}\mathsf{stop}\}$ for a fresh constant $\mathsf{stop}$. Let $\mathcal{A}'$ denote the resulting ADP problem.

Now we can apply the switching back processor. The underlying PTRS $\mathcal{DP}^{-1}(\mathcal{A}') = \{(7.38), \mathsf{h}(x) \to \{^{1}\mathsf{stop}\}\}$ is right-ground, non-overlapping, and tail-recursive, i.e., we have $\mathcal{DP}^{-1}(\mathcal{A}') \in$ RNT, although the original PTRS $\mathcal{P}_{\mathsf{back}}$ was not in RNT. Thus, we can decide AST of $\mathcal{DP}^{-1}(\mathcal{A}')$. Running the decision procedure of Algorithm 6.3.6, as in Example 6.3.7 proves $\mathtt{AST}(\to_{\mathcal{DP}^{-1}(\mathcal{A}')})$ and thus also $\mathtt{AST}(\xrightarrow{\mathsf{i}}_{\mathcal{DP}^{-1}(\mathcal{A}')})$. So $\mathcal{A}'$ is iAST by Theorem 7.4.22, and as all applied processors are sound, we obtain $\mathtt{AST}(\xrightarrow{\mathsf{i}}_{\mathcal{P}_{\mathsf{back}}})$.

SECTION 7.5

## Processors Based on Algebras

In this section, we introduce processors based on algebras that can be searched for via SMT solvers or SAT solvers. In total, we present two different processors.

1. The *reduction pair processor* generalizes the direct application of algebras to ADP problems.
2. The *subterm criterion* tries to compare arguments w.r.t. the subterm relation in order to prove iAST.

### 7.5.1 Reduction Pair Processor

Next, we consider the reduction pair processor, whose adaptation is the same for $\mathtt{AST}(\hookrightarrow_{\mathcal{A}})$ and $\mathtt{AST}(\hookrightarrow^{\mathsf{i}}_{\mathcal{A}})$. As in the direct application of algebras in Section 3.4.5, in the probabilistic setting, we cannot use arbitrary term orders, but as in [ADLY20], we have

to consider algebras that allow the definition of an *expected value*, called *barycentric* $\Sigma^\sharp$*-algebras* (see Definition 3.4.30). Compared to [ADLY20], we have to require even more structure, i.e., we have to restrict ourselves to algebras whose carrier forms an $\mathbb{R}_{\geq 0}$-semimodule (see Definition 3.4.26), so that we have access to an addition $\oplus$ and a scalar multiplication $\odot$ with the probabilities. As in Section 3.4.5, the expected value of a distribution $\mu = \{{}^{p_1}t_1, \dots, {}^{p_k}t_k\}$ of carrier elements is $\mathbb{E}(\mu) = \bigoplus_{i=1}^{k} p_i \odot t_i$, and the order $\succ$ has to be compatible with the semimodule structure (see Definition 3.4.28).

For example, polynomial interpretations are barycentric $\Sigma^\sharp$-algebras that are collapsible (see Definition 3.4.31), provided that Pol maps each function symbol to a *multilinear* polynomial (see Definition 3.4.32), i.e., a polynomial whose monomials have the form $c \cdot x_1^{e_1} \cdot \ldots \cdot x_n^{e_n}$ with $c \in \mathbb{N}$ and $e_1, \dots, e_n \in \{0, 1\}$, to ensure concavity.

In the following, let $(\mathfrak{I}, \succ)$ be a collapsible barycentric $\Sigma^\sharp$-algebra. The constraints of the reduction pair processor of Theorem 7.5.1 are based on the conditions of a ranking function for AST as in [MMKK18], which is closely related to ranking supermartingales [CS13]. Remember that if one proves AST by considering the rules $\ell \to \mu$ of a PTRS directly, then one needs a *monotonic* algebra and requires a weak decrease when comparing $\mathfrak{I}(\ell)$ to the expected value $\mathbb{E}(\mathfrak{I}(\mu))$ of the right-hand side, where we extend $\mathfrak{I}$ to multi-distributions by defining $\mathfrak{I}(\{{}^{p_1}t_1, \dots, {}^{p_k}t_k\}) = \{{}^{p_1}\mathfrak{I}(t_1), \dots, {}^{p_k}\mathfrak{I}(t_k)\}$. Moreover, at least one $\mathfrak{I}(r)$ with $r \in \mathrm{Supp}(\mu)$ must be strictly smaller than $\mathfrak{I}(\ell)$.

For ADPs, we do not use the interpretation $\mathfrak{I}(r)$ of a term $r$ directly, but we take the occurring annotations into account. Since only the annotated subterms may lead to non-AST sequences, we only consider the interpretations of these subterms. Hence, we compare the $\sharp$*-sums*[7] of the terms $r$ in ADPs, i.e., $\mathfrak{I}^\sharp_{sum}(r) = \bigoplus_{t \trianglelefteq_\sharp r} \mathfrak{I}(t^\sharp)$ sums up the values of $r$'s annotated subterms w.r.t. the semimodule addition $\oplus$. Here, the sum contains one summand for every annotated position (so equal subterms at several annotated positions are summed several times), and $\mathfrak{I}^\sharp_{sum}(r) = \mathbf{0}$ if $r$ contains no annotations. So for example, we have $\mathfrak{I}^\sharp_{sum}(\mathsf{d}^\sharp(\mathsf{g}^\sharp)) = \mathfrak{I}(\mathsf{d}^\sharp(\mathsf{g})) \oplus \mathfrak{I}(\mathsf{g}^\sharp)$.

If in each rewrite step $s \hookrightarrow_{\mathcal{A}} \mu$ of a chain tree $\mathfrak{T}$, the $\sharp$*-sum* is weakly decreasing in expectation ($\mathfrak{I}^\sharp_{sum}(s) \succcurlyeq \mathbb{E}(\mathfrak{I}^\sharp_{sum}(\mu))$), and additionally, at least one $\mathfrak{I}^\sharp_{sum}(r)$ with $r \in \mathrm{Supp}(\mu)$ is smaller than $\mathfrak{I}^\sharp_{sum}(s)$ by at least some fixed $\epsilon > 0$, then we can conclude that $\mathfrak{T}$ converges with probability 1. Again, $\mathfrak{I}^\sharp_{sum}$ is extended to multi-distributions by defining $\mathfrak{I}^\sharp_{sum}(\{{}^{p_1}t_1, \dots, {}^{p_k}t_k\}) = \{{}^{p_1}\mathfrak{I}^\sharp_{sum}(t_1), \dots, {}^{p_k}\mathfrak{I}^\sharp_{sum}(t_k)\}$. So for a rewrite step of the form $\underline{\mathsf{g}^\sharp} \hookrightarrow_{\mathcal{A}} \{{}^{3/4}\mathsf{d}^\sharp(\mathsf{g}^\sharp), {}^{1/4}0\}$ one would for example obtain the constraints $\mathfrak{I}^\sharp_{sum}(\mathsf{g}^\sharp) = \mathfrak{I}(\mathsf{g}^\sharp) \succcurlyeq \mathbb{E}(\{{}^{3/4}\mathfrak{I}^\sharp_{sum}(\mathsf{d}^\sharp(\mathsf{g}^\sharp)), {}^{1/4}\mathfrak{I}^\sharp_{sum}(0)\}) = {}^{3/4}\!/_4 \odot \mathfrak{I}^\sharp_{sum}(\mathsf{d}^\sharp(\mathsf{g}^\sharp)) \oplus {}^{1/4}\!/_4 \odot \mathfrak{I}^\sharp_{sum}(0) = {}^{3/4}\!/_4 \odot (\mathfrak{I}(\mathsf{d}^\sharp(\mathsf{g})) \oplus \mathfrak{I}(\mathsf{g}^\sharp)) \oplus {}^{1/4}\!/_4 \odot \mathbf{0} = {}^{3/4}\!/_4 \odot (\mathfrak{I}(\mathsf{d}^\sharp(\mathsf{g})) \oplus \mathfrak{I}(\mathsf{g}^\sharp))$ and $\mathfrak{I}^\sharp_{sum}(\mathsf{g}^\sharp) = \mathfrak{I}(\mathsf{g}^\sharp) \succ_\epsilon \mathfrak{I}^\sharp_{sum}(0) = \mathbf{0}$.

Instead of imposing (usually infinitely many) constraints for all possible rewrite steps $s \hookrightarrow_{\mathcal{A}} \mu$, we only want to impose finitely many constraints which correspond to the finitely many ADPs in the ADP problem $\mathcal{A}$. If these inequality constraints hold for the ADPs, then they also hold for arbitrary rewrite steps if $(\mathfrak{I}, \succ)$ is *weakly monotonic*. So as in the non-probabilistic reduction pair processor of Theorem 7.1.15, due to the usage of $\sharp$-sums, weak instead of strict monotonicity is sufficient. However, as in [ADLY20], the requirement of concavity of all functions $f_\mathfrak{I}$ is needed to ensure that we also have

[7] This is the reason why we require $\mathbb{R}_{\geq 0}$-semimodules instead of simply requiring the existence of a barycentric operation $\mathbb{E}$ as in [ADLY20].

weak monotonicity w.r.t. expected values. So $\mathfrak{I}(t) \succcurlyeq \mathbb{E}(\mathfrak{I}(\mu))$ implies

$$\begin{aligned}\mathfrak{I}(f(\dots,t,\dots)) &= f_{\mathfrak{I}}(\dots,\mathfrak{I}(t),\dots)\\ &\quad\downarrow \quad \text{(by weak monotonicity)}\\ &\succcurlyeq f_{\mathfrak{I}}(\dots,\mathbb{E}(\mathfrak{I}(\mu)),\dots)\\ &\quad\downarrow \quad \text{(by concavity)}\\ &\succcurlyeq \mathbb{E}(f_{\mathfrak{I}}(\dots,\mathfrak{I}(\mu),\dots)).\end{aligned}$$

Similarly to Theorem 7.1.15, we do not need to find an algebra that orients all ADPs strictly. We impose the following constraints for all ADPs (which adapt the constraints (1)–(3) from Theorem 7.1.15 to the probabilistic setting): First, (1) we require a weak decrease when comparing the annotated left-hand side with the expected value of the $\sharp$-*sums* in the right-hand sides. The processor then removes the annotations from those ADPs where (2a) there is at least one right-hand side $r$ whose $\sharp$-*sum* is strictly decreasing by some fixed $\epsilon > 0$. If $m = \mathtt{t}$, then in addition, (2b) the corresponding non-annotated right-hand side $\flat(r)$ must be at least weakly decreasing to ensure that nested annotations behave monotonically.[8] Finally, (3) for every rule with the flag $\mathtt{t}$ (which can therefore be used for steps below annotations), the expected value must be weakly decreasing when removing all annotations. Moreover, we require the embedding of the algebra to be $\epsilon$-separated on interpretations of terms: otherwise, weakly decreasing steps could decrease the embedded values by arbitrarily small positive amounts, which would break the probability bounds in our proof.[9]

Theorem 7.5.1 – Reduction Pair Processor for AST & iAST.

Let $\mathcal{A}$ be an ADP problem, let $(\mathfrak{I},\succ)$ be a weakly monotonic, collapsible barycentric $\Sigma^\sharp$-algebra, and let $\epsilon > 0$ such that an embedding emb that witnesses the collapsibility of $(\mathfrak{I},\succ)$ is $\epsilon$-*separated* on interpretations of terms, i.e., for all terms $t, t'$, the values $\mathrm{emb}(\mathfrak{I}_0^\sharp(t))$ and $\mathrm{emb}(\mathfrak{I}_0^\sharp(t'))$ are either equal or differ by at least $\epsilon$, where $\mathfrak{I}_0^\sharp$ behaves like $\mathfrak{I}_{sum}^\sharp$ but maps every variable to $\mathbf{0}$. Let $\mathcal{A} = \mathcal{A}_{\succcurlyeq} \uplus \mathcal{A}_{\succ}$ with $\mathcal{A}_{\succ} \neq \varnothing$ where:

| | | | |
|---|---|---|---|
| (1) | | $\forall \ell \xrightarrow{m} \mu \in \mathcal{A}.$ | $\mathfrak{I}_{sum}^\sharp(\ell^\sharp) \succcurlyeq \mathbb{E}(\mathfrak{I}_{sum}^\sharp(\mu))$ |
| (2) | (a) | $\forall \ell \xrightarrow{m} \mu \in \mathcal{A}_{\succ} \exists r \in \mathrm{Supp}(\mu).$ | $\mathfrak{I}_{sum}^\sharp(\ell^\sharp) \succ_\epsilon \mathfrak{I}_{sum}^\sharp(r)$ |
| | (b) | If $m = \mathtt{t}$, then we additionally have | $\mathfrak{I}(\ell) \succcurlyeq \mathfrak{I}(\flat(r))$ |
| (3) | | $\forall \ell \xrightarrow{\mathtt{t}} \mu \in \mathcal{A}.$ | $\mathfrak{I}(\ell) \succcurlyeq \mathbb{E}(\mathfrak{I}(\flat(\mu)))$. |

Then $\mathrm{Proc}_{\mathtt{RP}}(\mathcal{A}) = \mathrm{Proc}^{\mathtt{i}}_{\mathtt{RP}}(\mathcal{A}) = \{\mathcal{A}_{\succcurlyeq} \cup \flat(\mathcal{A}_{\succ})\}$ is sound and complete for AST and iAST.

Before proving the soundness and completeness of the reduction pair processor, we prove an additional lemma that contains all technicalities. Here, we additionally assume that (+) every infinite path in $\mathfrak{T}$ has an infinite number of $\Gamma_{\succ}$ nodes. However, this is not a restriction but can be ensured during the proof of the actual reduction pair processor.

[8] For example, it ensures that $\mathfrak{I}(g^\sharp) \succ_\epsilon \mathfrak{I}(h^\sharp)$ also implies that the $\sharp$-*sum* of $f^\sharp(g^\sharp)$ is greater than the $\sharp$-sum of $f^\sharp(h^\sharp)$, i.e., $\mathfrak{I}(g^\sharp) \succ_\epsilon \mathfrak{I}(h^\sharp)$ must imply that $\mathfrak{I}_{sum}^\sharp(f^\sharp(g^\sharp)) = \mathfrak{I}(f^\sharp(g)) \oplus \mathfrak{I}(g^\sharp) \succ_\epsilon \mathfrak{I}(f^\sharp(h)) \oplus \mathfrak{I}(h^\sharp) = \mathfrak{I}_{sum}^\sharp(f^\sharp(h^\sharp))$, which is ensured by $\mathfrak{I}(g) \succcurlyeq \mathfrak{I}(h)$.

[9] In fact, the proof relies on the following property of weakly decreasing steps, which holds pointwise for the values of the terms: If $\mathfrak{I}^\sharp(s) \succcurlyeq \mathbb{E}(\{p_1 \mathfrak{I}^\sharp(t_1), \dots, p_k \mathfrak{I}^\sharp(t_k)\})$, then either $\mathrm{emb}(\mathfrak{I}^\sharp(s)) = \mathrm{emb}(\mathfrak{I}^\sharp(t_i))$ for all $1 \le i \le k$, or we have at least one $1 \le j \le k$ with $\mathrm{emb}(\mathfrak{I}^\sharp(s)) \ge \mathrm{emb}(\mathfrak{I}^\sharp(t_j)) + \epsilon$.

Lemma 7.5.2 – Proving AST of Chain Trees with Algebras.

*Let $\mathcal{A}$ be an ADP problem, let $(\mathfrak{I}, \succ)$ be a weakly monotonic, collapsible barycentric $\Sigma^\sharp$-algebra, and let $\epsilon > 0$ such that an embedding* emb *that witnesses the collapsibility of $(\mathfrak{I}, \succ)$ is $\epsilon$-separated on interpretations of terms, i.e., for all terms $t, t'$, the values* $\mathrm{emb}(\mathfrak{I}_0^\sharp(t))$ *and* $\mathrm{emb}(\mathfrak{I}_0^\sharp(t'))$ *are either equal or differ by at least $\epsilon$, where $\mathfrak{I}_0^\sharp$ behaves like $\mathfrak{I}_{sum}^\sharp$ but maps every variable to $\mathbf{0}$. Let $\mathcal{A} = \mathcal{A}_{\succcurlyeq} \uplus \mathcal{A}_{\succ}$ with $\mathcal{A}_{\succ} \neq \varnothing$ where*

| | | | |
|---|---|---|---|
| (1) | | $\forall \ell \xhookrightarrow{m} \mu \in \mathcal{A}$. | $\mathfrak{I}_{sum}^\sharp(\ell^\sharp) \succcurlyeq \mathbb{E}(\mathfrak{I}_{sum}^\sharp(\mu))$ |
| (2) | (a) | $\forall \ell \xhookrightarrow{m} \mu \in \mathcal{A}_{\succ} \exists r \in \mathrm{Supp}(\mu)$. | $\mathfrak{I}_{sum}^\sharp(\ell^\sharp) \succ_\epsilon \mathfrak{I}_{sum}^\sharp(r)$ |
| | (b) | *If* $m = \mathtt{t}$, *then we additionally have* | $\mathfrak{I}(\ell) \succcurlyeq \mathfrak{I}(\flat(r))$ |
| (3) | | $\forall \ell \xhookrightarrow{\mathtt{t}} \mu \in \mathcal{A}$. | $\mathfrak{I}(\ell) \succcurlyeq \mathbb{E}(\mathfrak{I}(\flat(\mu)))$. |

*Let $\mathfrak{T} = (N, E)$ be an $\mathcal{A}$-CT. We can partition $\Gamma = \Gamma_{\succcurlyeq} \uplus \Gamma_{\succ}$ according to $\mathcal{A} = \mathcal{A}_{\succcurlyeq} \uplus \mathcal{A}_{\succ}$. If*

(+) *every infinite path in $\mathfrak{T}$ has an infinite number of $\Gamma_{\succ}$ nodes,*

*then $|\mathfrak{T}| = 1$.*

*Proof.* This proof uses the proof idea for AST from [MMKK18] and lifts the proof of Theorem 3.4.38 for PTRSs to the ADP problems. For an $\mathcal{A}$-CT $\mathfrak{T}$ that satisfies (+), the core steps of the proof are the following:

1. We extend the conditions (1), (2), and (3) to rewrite steps instead of just rules.
2. We create a CT $\mathfrak{T}^{\leq U}$ for any $U \in \mathbb{N}$.
3. We prove that $|\mathfrak{T}^{\leq U}| \geq p_{min}^{U+1}$ for any $U \in \mathbb{N}$.
4. We prove that $|\mathfrak{T}^{\leq U}| = 1$ for any $U \in \mathbb{N}$.
5. Finally, we prove that $|\mathfrak{T}| = 1$.

Here, let $p_{min} > 0$ be the minimal probability that occurs in the ADPs from $\mathcal{A}$. As $\mathcal{A}$ consists of only finitely many ADPs and all occurring multi-distributions are finite as well, this minimum is well defined. We have $p_{min}^{U+1} = \underbrace{p_{min} \cdot \ldots \cdot p_{min}}_{U+1 \text{ times}}$.

**Step 1** Extend the conditions to rewrite steps instead of just rules

We show that the conditions (1), (2), and (3) extend to rewrite steps instead of just rules. Let $\mathcal{P}$ be a PTRS with $\ell \to \{{}^{p_1}r_1, \ldots, {}^{p_k}r_k\} \in \mathcal{P}$.

(a) If we have $s \to \{{}^{p_1}t_1, \ldots, {}^{p_k}t_k\}$ using the rule $\ell \to \{{}^{p_1}r_1, \ldots, {}^{p_k}r_k\}$ where $\mathfrak{I}(\ell) \succcurlyeq \mathfrak{I}(r_j)$ for some $1 \leq j \leq k$, then we have $\mathfrak{I}(s) \succcurlyeq \mathfrak{I}(t_j)$.

(b) If we have $a \hookrightarrow_{\mathcal{A}} \{{}^{p_1}b_1, \ldots, {}^{p_k}b_k\}$ using $\ell \xhookrightarrow{m} \{{}^{p_1}r_1, \ldots, {}^{p_k}r_k\} \in \mathcal{A}_{\succ}$ at a position $\pi \in \mathrm{pos}_{\mathcal{D}^\sharp}(a)$, then $\mathfrak{I}_{sum}^\sharp(a) \succ_\epsilon \mathfrak{I}_{sum}^\sharp(b_j)$ for some $1 \leq j \leq k$.

(c) If we have $s \to \{{}^{p_1}t_1, \ldots, {}^{p_k}t_k\}$ using the rule $\ell \to \{{}^{p_1}r_1, \ldots, {}^{p_k}r_k\}$ where $\mathfrak{I}(\ell) \succcurlyeq \bigoplus_{1 \leq j \leq k} p_j \odot \mathfrak{I}(r_j)$, then $\mathfrak{I}(s) \succcurlyeq \bigoplus_{1 \leq j \leq k} p_j \odot \mathfrak{I}(t_j)$.

(d) If we have $a \hookrightarrow_{\mathcal{A}} \{{}^{p_1}b_1, \ldots, {}^{p_k}b_k\}$ using $\ell \xhookrightarrow{m} \{{}^{p_1}r_1, \ldots, {}^{p_k}r_k\} \in \mathcal{A}$, then $\mathfrak{I}_{sum}^\sharp(a) \succcurlyeq \bigoplus_{1 \leq j \leq k} p_j \odot \mathfrak{I}_{sum}^\sharp(b_j)$.

(a) In this case, there exist a rule $\ell \to \{{}^{p_1}r_1, \dots, {}^{p_k}r_k\}$ with $\mathfrak{I}(\ell) \succcurlyeq \mathfrak{I}(r_j)$ for some $1 \le j \le k$, a substitution $\sigma$, and a position $\pi$ of $s$ such that $s|_\pi = \ell\sigma$ and $t_h = s[r_h\sigma]_\pi$ for all $1 \le h \le k$.

We perform structural induction on $\pi$. So in the induction base, let $\pi = \varepsilon$. Hence, we have $s = \ell\sigma \to_{\mathcal{P}} \{{}^{p_1}r_1\sigma, \dots, {}^{p_k}r_k\sigma\}$. By assumption, we have $\mathfrak{I}(\ell) \succcurlyeq \mathfrak{I}(r_j)$ for some $1 \le j \le k$. As these inequations hold for all instantiations of the occurring variables, for $t_j = r_j\sigma$ we have

$$\mathfrak{I}(s) = \mathfrak{I}(\ell\sigma) \succcurlyeq \mathfrak{I}(r_j\sigma) = \mathfrak{I}(t_j).$$

In the induction step, we have $\pi = i.\pi'$, $s = f(s_1, \dots, s_i, \dots, s_n)$, $f \in \Sigma^\sharp$, $s_i \to_{\mathcal{P}} \{{}^{p_1}t_{i,1}, \dots, {}^{p_k}t_{i,k}\}$, and $t_j = f(s_1, \dots, t_{i,j}, \dots, s_n)$ with $t_{i,j} = s_i[r_j\sigma]_{\pi'}$ for all $1 \le j \le k$. Then by the induction hypothesis we have $\mathfrak{I}(s_i) \succcurlyeq \mathfrak{I}(t_{i,j})$. For $t_j = f(s_1, \dots, t_{i,j}, \dots, s_n)$ we obtain

$$\begin{aligned}
\mathfrak{I}(s) =& \mathfrak{I}(f(s_1, \dots, s_i, \dots, s_n)) \\
=& f_{\mathfrak{I}}(\mathfrak{I}(s_1), \dots, \mathfrak{I}(s_i), \dots, \mathfrak{I}(s_n)) \\
& \quad \downarrow \quad \text{(by weak monotonicity of } f_{\mathfrak{I}} \text{ and } \mathfrak{I}(s_i) \succcurlyeq \mathfrak{I}(t_{i,j})) \\
\succcurlyeq& f_{\mathfrak{I}}(\mathfrak{I}(s_1), \dots, \mathfrak{I}(t_{i,j}), \dots, \mathfrak{I}(s_n)) \\
=& \mathfrak{I}(f(s_1, \dots, t_{i,j}, \dots, s_n)) \\
=& \mathfrak{I}(t_j).
\end{aligned}$$

(b) In this case, there exist an ADP $\ell \xhookrightarrow{m} \{{}^{p_1}r_1, \dots, {}^{p_k}r_k\} \in \mathcal{A}_\succ$, a VRF $(\varphi_j)_{1 \le j \le k}$, a substitution $\sigma$, and position $\pi \in \mathrm{pos}_{\mathcal{D}^\sharp}(a)$ with $\flat(a|_\pi) = \ell\sigma$ and $b_j \overset{\flat}{=} a[r_j\sigma]_\pi$. Assume that $m = \mathtt{t}$. Let $I_1 = \{\tau \in \mathrm{pos}_{\mathcal{D}^\sharp}(a) \mid \tau < \pi\}$ be the set of positions of all annotations strictly above $\pi$, $I_2 = \{\tau \in \mathrm{pos}_{\mathcal{D}^\sharp}(a) \mid \pi < \tau \wedge \neg\exists\gamma \in \mathrm{pos}_{\mathcal{V}}(\ell).\ \pi.\gamma \le \tau\}$ be the set of positions of all annotations strictly below $\pi$ inside the non-variable part of the left-hand side $\ell$ (but not at the root of $\ell$), $I_3 = \{\tau \in \mathrm{pos}_{\mathcal{D}^\sharp}(a) \mid \exists\gamma \in \mathrm{pos}_{\mathcal{V}}(\ell).\ \pi.\gamma \le \tau\}$ be the set of positions of all remaining annotations at or below a variable position of the used redex $\ell\sigma$ (i.e., inside the substitution), and let $I_4 = \{\tau \in \mathrm{pos}_{\mathcal{D}^\sharp}(a) \mid \tau \perp \pi\}$ be the set of positions of all annotations orthogonal to $\pi$. Furthermore, for each $\tau \in I_1$ let $\kappa_\tau$ be the position such that $\tau.\kappa_\tau = \pi$, and for each $\tau \in I_3$ let $\gamma_\tau$ and $\rho_\tau$ be the positions such that $\gamma_\tau \in \mathrm{pos}_{\mathcal{V}}(\ell)$ and $\pi.\gamma_\tau.\rho_\tau = \tau$ (where $\rho_\tau = \varepsilon$ is possible). By Requirement (2), there exists a $1 \le j \le k$ with $\mathfrak{I}(\ell^\sharp) = \mathfrak{I}^\sharp_{sum}(\ell^\sharp) \succ_\epsilon \mathfrak{I}^\sharp_{sum}(r_j) = \bigoplus_{t \trianglelefteq^\sharp r_j} \mathfrak{I}(t^\sharp)$ and, additionally, $\mathfrak{I}(\ell) \succcurlyeq \mathfrak{I}(\flat(r_j))$ since $m = \mathtt{t}$. As these inequations hold for all instantiations of the occurring variables, we have

$$\begin{aligned}
\mathfrak{I}^\sharp_{sum}(a) \;=\; & \bigoplus_{t \trianglelefteq^\sharp a} \mathfrak{I}(t^\sharp) \\
= \mathfrak{I}(\sharp_{\{\varepsilon\}}(a|_\pi)) \oplus & \bigoplus_{\tau \in I_1} \mathfrak{I}(\sharp_{\{\varepsilon\}}(a|_\tau)) \oplus \bigoplus_{\tau \in I_2} \mathfrak{I}(\sharp_{\{\varepsilon\}}(a|_\tau)) \\
\oplus & \bigoplus_{\tau \in I_3} \mathfrak{I}(\sharp_{\{\varepsilon\}}(a|_\tau)) \oplus \bigoplus_{\tau \in I_4} \mathfrak{I}(\sharp_{\{\varepsilon\}}(a|_\tau))
\end{aligned}$$

$$
\begin{aligned}
&\quad\downarrow \quad \text{(removed } I_2 \text{ by monotonicity of } \oplus\text{)}\\
&\succcurlyeq \mathfrak{I}(\sharp_{\{\varepsilon\}}(a|_\pi)) \oplus \bigoplus_{\tau\in I_1} \mathfrak{I}(\sharp_{\{\varepsilon\}}(a|_\tau))\\
&\qquad \oplus \bigoplus_{\tau\in I_3} \mathfrak{I}(\sharp_{\{\varepsilon\}}(a|_\tau)) \oplus \bigoplus_{\tau\in I_4} \mathfrak{I}(\sharp_{\{\varepsilon\}}(a|_\tau))\\
&\quad\downarrow \quad \text{(as } \sharp_{\{\varepsilon\}}(a|_\pi) = \sharp_{\{\varepsilon\}}(\ell)\sigma\text{)}\\
&= \mathfrak{I}(\sharp_{\{\varepsilon\}}(\ell)\sigma) \oplus \bigoplus_{\tau\in I_1} \mathfrak{I}(\sharp_{\{\varepsilon\}}(a|_\tau))\\
&\qquad \oplus \bigoplus_{\tau\in I_3} \mathfrak{I}(\sharp_{\{\varepsilon\}}(a|_\tau)) \oplus \bigoplus_{\tau\in I_4} \mathfrak{I}(\sharp_{\{\varepsilon\}}(a|_\tau))\\
&\quad\downarrow \quad \text{(as } \mathfrak{I}(\sharp_{\{\varepsilon\}}(\ell)) \succ_\epsilon \bigoplus_{t\trianglelefteq_\sharp r_j} \mathfrak{I}(\sharp_{\{\varepsilon\}}(t))\text{, hence } \mathfrak{I}(\sharp_{\{\varepsilon\}}(\ell)\sigma) \succ_\epsilon \bigoplus_{t\trianglelefteq_\sharp r_j} \mathfrak{I}(\sharp_{\{\varepsilon\}}(t)\sigma)\text{)}\\
&\succ_\epsilon \bigoplus_{t\trianglelefteq_\sharp r_j} \mathfrak{I}(\sharp_{\{\varepsilon\}}(t)\sigma) \oplus \bigoplus_{\tau\in I_1} \mathfrak{I}(\sharp_{\{\varepsilon\}}(a|_\tau))\\
&\qquad \oplus \bigoplus_{\tau\in I_3} \mathfrak{I}(\sharp_{\{\varepsilon\}}(a|_\tau)) \oplus \bigoplus_{\tau\in I_4} \mathfrak{I}(\sharp_{\{\varepsilon\}}(a|_\tau))\\
&\quad\downarrow \quad \text{(by } \mathfrak{I}(\ell) \succcurlyeq \mathfrak{I}(\flat(r_j)) \text{ and (a))}\\
&\succcurlyeq \bigoplus_{t\trianglelefteq_\sharp r_j\sigma} \mathfrak{I}(\sharp_{\{\varepsilon\}}(t)) \oplus \bigoplus_{\tau\in I_1} \mathfrak{I}(\sharp_{\{\varepsilon\}}(a|_\tau[r_j\sigma]_{\kappa_\tau}))\\
&\qquad \oplus \bigoplus_{\tau\in I_3} \mathfrak{I}(\sharp_{\{\varepsilon\}}(a|_\tau)) \oplus \bigoplus_{\tau\in I_4} \mathfrak{I}(\sharp_{\{\varepsilon\}}(a|_\tau))\\
&\quad\downarrow \quad \text{(moving } \tau = \pi.\gamma_\tau.\rho_\tau \in I_3 \text{ via the VRF)}\\
&\succcurlyeq \bigoplus_{t\trianglelefteq_\sharp r_j\sigma} \mathfrak{I}(\sharp_{\{\varepsilon\}}(t)) \oplus \bigoplus_{\tau\in I_1} \mathfrak{I}(\sharp_{\{\varepsilon\}}(a|_\tau[r_j\sigma]_{\kappa_\tau}))\\
&\qquad \oplus \bigoplus_{\substack{\tau\in I_3\\ \varphi_j(\gamma_\tau)\neq\bot}} \mathfrak{I}(\sharp_{\{\varepsilon\}}(b_j|_{\pi.\varphi_j(\gamma_\tau).\rho_\tau})) \oplus \bigoplus_{\tau\in I_4} \mathfrak{I}(\sharp_{\{\varepsilon\}}(a|_\tau))\\
&\quad\downarrow \quad \text{(}\succcurlyeq\text{, as VRFs may be non-injective)}\\
&\succcurlyeq \bigoplus_{t\trianglelefteq_\sharp b_j} \mathfrak{I}(t^\sharp) \;=\; \mathfrak{I}^\sharp_{sum}(b_j).
\end{aligned}
$$

Since the single $\succ_\epsilon$-step is combined with weak steps only, the whole chain yields $\mathfrak{I}^\sharp_{sum}(a) \succ_\epsilon \mathfrak{I}^\sharp_{sum}(b_j)$, as $\succ_\epsilon$ absorbs $\succcurlyeq$ (see the remark below Definition 3.4.31). In the case $m = \mathsf{f}$, the $I_1$-summands are removed as well, and the inequation remains correct.

(c) In this case, there exist a rule $\ell \to \{{}^{p_1}r_1, \ldots, {}^{p_k}r_k\}$ with $\mathfrak{I}(\ell) \succcurlyeq \bigoplus_{1\le j\le k} p_j \odot \mathfrak{I}(r_j)$, a substitution $\sigma$, and a position $\pi$ of $s$ such that $s|_\pi = \ell\sigma$, and $t_j = s[r_j\sigma]_\pi$ for all $1 \le j \le k$.

We perform structural induction on $\pi$. So in the induction base $\pi = \varepsilon$ we have $s = \ell\sigma \to_{\mathcal{P}} \{{}^{p_1}r_1\sigma, \ldots, {}^{p_k}r_k\sigma\}$. As $\mathfrak{I}(\ell) \succcurlyeq \bigoplus_{1\le j\le k} p_j \odot \mathfrak{I}(r_j)$ holds for all

instantiations of the occurring variables, for $t_j = r_j\sigma$ we obtain

$$\mathfrak{I}(s) \;=\; \mathfrak{I}(\ell\sigma) \;\succcurlyeq\; \bigoplus_{1\le j\le k} p_j \odot \mathfrak{I}(r_j\sigma) \;=\; \bigoplus_{1\le j\le k} p_j \odot \mathfrak{I}(t_j).$$

In the induction step, we have $\pi = i.\pi'$, $s = f(s_1,\dots,s_i,\dots,s_n)$, $s_i \to_{\mathcal{P}} \{{}^{p_1}t_{i,1},\dots,{}^{p_k}t_{i,k}\}$, and $t_j = f(s_1,\dots,t_{i,j},\dots,s_n)$ with $t_{i,j} = s_i[r_j\sigma]_{\pi'}$ for all $1 \le j \le k$. Then by the induction hypothesis we have $\mathfrak{I}(s_i) \succcurlyeq \bigoplus_{1\le j\le k} p_j \odot \mathfrak{I}(t_{i,j})$. Thus, we have

$$\begin{aligned}
\mathfrak{I}(s) &= \mathfrak{I}(f(s_1,\dots,s_i,\dots,s_n))\\
&= f_{\mathfrak{J}}(\mathfrak{I}(s_1),\dots,\mathfrak{I}(s_i),\dots,\mathfrak{I}(s_n))\\
&\qquad\downarrow \quad \text{(by weak monotonicity of } f_{\mathfrak{J}} \text{ and } \mathfrak{I}(s_i) \succcurlyeq \textstyle\bigoplus_{1\le j\le k} p_j \odot \mathfrak{I}(t_{i,j})\text{)}\\
&\succcurlyeq f_{\mathfrak{J}}(\mathfrak{I}(s_1),\dots,\textstyle\bigoplus_{1\le j\le k} p_j \odot \mathfrak{I}(t_{i,j}),\dots,\mathfrak{I}(s_n))\\
&\qquad\downarrow \quad \text{(as } f_{\mathfrak{J}} \text{ is concave)}\\
&\succcurlyeq \textstyle\bigoplus_{1\le j\le k} p_j \odot f_{\mathfrak{J}}(\mathfrak{I}(s_1),\dots,\mathfrak{I}(t_{i,j}),\dots,\mathfrak{I}(s_n))\\
&= \textstyle\bigoplus_{1\le j\le k} p_j \odot \mathfrak{I}(f(s_1,\dots,t_{i,j},\dots,s_n))\\
&= \textstyle\bigoplus_{1\le j\le k} p_j \odot \mathfrak{I}(t_j).
\end{aligned}$$

(d) In this case, there exist an ADP $\ell \xhookrightarrow{m} \{{}^{p_1}r_1,\dots,{}^{p_k}r_k\} \in \mathcal{A}$, a substitution $\sigma$, and position $\pi$ with $\flat(a|_\pi) = \ell\sigma$ and $b_j \overset{\flat}{=} a[r_j\sigma]_\pi$. First, assume that $m = \mathtt{t}$ and $\pi \in \mathrm{pos}_{\mathcal{D}^\sharp}(a)$. Let $I_1 = \{\tau \in \mathrm{pos}_{\mathcal{D}^\sharp}(a) \mid \tau < \pi\}$ be the set of positions of all annotations strictly above $\pi$, $I_2 = \{\tau \in \mathrm{pos}_{\mathcal{D}^\sharp}(a) \mid \pi < \tau \wedge \neg\exists\gamma \in \mathrm{pos}_{\mathcal{V}}(\ell).\ \pi.\gamma \le \tau\}$ be the set of positions of all annotations strictly below $\pi$ inside the non-variable part of the left-hand side $\ell$ (but not at the root of $\ell$), $I_3 = \{\tau \in \mathrm{pos}_{\mathcal{D}^\sharp}(a) \mid \exists\gamma \in \mathrm{pos}_{\mathcal{V}}(\ell).\ \pi.\gamma \le \tau\}$ be the set of positions of all remaining annotations at or below a variable position of the used redex $\ell\sigma$ (i.e., inside the substitution), and let $I_4 = \{\tau \in \mathrm{pos}_{\mathcal{D}^\sharp}(a) \mid \tau \perp \pi\}$ be the set of positions of all annotations orthogonal to $\pi$. Furthermore, for each $\tau \in I_1$ let $\kappa_\tau$ be the position such that $\tau.\kappa_\tau = \pi$, and for each $\tau \in I_3$ let $\gamma_\tau$ and $\rho_\tau$ be the positions such that $\gamma_\tau \in \mathrm{pos}_{\mathcal{V}}(\ell)$ and $\pi.\gamma_\tau.\rho_\tau = \tau$ (where $\rho_\tau = \varepsilon$ is possible). By Requirement (1), we have $\mathfrak{I}(\sharp_{\{\varepsilon\}}(\ell)) = \mathfrak{I}^\sharp_{sum}(\sharp_{\{\varepsilon\}}(\ell)) \succcurlyeq \bigoplus_{1\le j\le k} p_j \odot \bigoplus_{t \trianglelefteq_\sharp r_j} \mathfrak{I}(\sharp_{\{\varepsilon\}}(t))$ and by (3) we have $\mathfrak{I}(\ell) \succcurlyeq \bigoplus_{1\le j\le k} p_j \odot \mathfrak{I}(\flat(r_j))$. As these inequations hold for all instantiations of the occurring variables, we have

$$\mathfrak{I}^\sharp_{sum}(a) \;=\; \bigoplus_{t \trianglelefteq_\sharp a} \mathfrak{I}(t^\sharp)$$

$$= \mathfrak{I}(\sharp_{\{\varepsilon\}}(a|_\pi)) \oplus \bigoplus_{\tau\in I_1} \mathfrak{I}(\sharp_{\{\varepsilon\}}(a|_\tau)) \oplus \bigoplus_{\tau\in I_2} \mathfrak{I}(\sharp_{\{\varepsilon\}}(a|_\tau))$$

$$\oplus \bigoplus_{\tau\in I_3} \mathfrak{I}(\sharp_{\{\varepsilon\}}(a|_\tau)) \oplus \bigoplus_{\tau\in I_4} \mathfrak{I}(\sharp_{\{\varepsilon\}}(a|_\tau))$$

$\downarrow$ (removed $I_2$ by monotonicity of $\oplus$)

$$\succcurlyeq \mathfrak{I}(\sharp_{\{\varepsilon\}}(a|_\pi)) \oplus \bigoplus_{\tau\in I_1} \mathfrak{I}(\sharp_{\{\varepsilon\}}(a|_\tau)) \oplus \bigoplus_{\tau\in I_3} \mathfrak{I}(\sharp_{\{\varepsilon\}}(a|_\tau)) \oplus \bigoplus_{\tau\in I_4} \mathfrak{I}(\sharp_{\{\varepsilon\}}(a|_\tau))$$

$\downarrow$ (as $\sharp_{\{\varepsilon\}}(a|_\pi) = \sharp_{\{\varepsilon\}}(\ell)\sigma$)

$$= \mathfrak{I}(\sharp_{\{\varepsilon\}}(\ell)\sigma) \oplus \bigoplus_{\tau\in I_1} \mathfrak{I}(\sharp_{\{\varepsilon\}}(a|_\tau)) \oplus \bigoplus_{\tau\in I_3} \mathfrak{I}(\sharp_{\{\varepsilon\}}(a|_\tau)) \oplus \bigoplus_{\tau\in I_4} \mathfrak{I}(\sharp_{\{\varepsilon\}}(a|_\tau))$$

$\downarrow$ $\Big($ by $\mathfrak{I}(\sharp_{\{\varepsilon\}}(\ell)) \succcurlyeq \bigoplus_{1\le j\le k} p_j \odot \bigoplus_{t\trianglelefteq_\sharp r_j} \mathfrak{I}(\sharp_{\{\varepsilon\}}(t))$,

hence $\mathfrak{I}(\sharp_{\{\varepsilon\}}(\ell)\sigma) \succcurlyeq \bigoplus_{1\le j\le k} p_j \odot \bigoplus_{t\trianglelefteq_\sharp r_j\sigma} \mathfrak{I}(\sharp_{\{\varepsilon\}}(t))\Big)$

$$\succcurlyeq \bigoplus_{1\le j\le k} p_j \odot \bigoplus_{t\trianglelefteq_\sharp r_j\sigma} \mathfrak{I}(\sharp_{\{\varepsilon\}}(t)) \oplus \bigoplus_{\tau\in I_1} \mathfrak{I}(\sharp_{\{\varepsilon\}}(a|_\tau)) \oplus \bigoplus_{\tau\in I_3} \mathfrak{I}(\sharp_{\{\varepsilon\}}(a|_\tau)) \oplus \bigoplus_{\tau\in I_4} \mathfrak{I}(\sharp_{\{\varepsilon\}}(a|_\tau))$$

$\downarrow$ $\Big($ by $\mathfrak{I}(\ell) \succcurlyeq \bigoplus_{1\le j\le k} p_j \odot \mathfrak{I}(\flat(r_j))$ and (c) $\Big)$

$$\succcurlyeq \bigoplus_{1\le j\le k} p_j \odot \bigoplus_{t\trianglelefteq_\sharp r_j\sigma} \mathfrak{I}(\sharp_{\{\varepsilon\}}(t)) \oplus \bigoplus_{\tau\in I_1} \bigoplus_{1\le j\le k} p_j \odot \mathfrak{I}(\sharp_{\{\varepsilon\}}(a|_\tau[r_j\sigma]_{\kappa_\tau})) \oplus \bigoplus_{\tau\in I_3} \mathfrak{I}(\sharp_{\{\varepsilon\}}(a|_\tau)) \oplus \bigoplus_{\tau\in I_4} \mathfrak{I}(\sharp_{\{\varepsilon\}}(a|_\tau))$$

$$= \bigoplus_{1\le j\le k} p_j \odot \bigoplus_{t\trianglelefteq_\sharp r_j\sigma} \mathfrak{I}(\sharp_{\{\varepsilon\}}(t)) \oplus \bigoplus_{1\le j\le k} \bigoplus_{\tau\in I_1} p_j \odot \mathfrak{I}(\sharp_{\{\varepsilon\}}(a|_\tau[r_j\sigma]_{\kappa_\tau})) \oplus \bigoplus_{\tau\in I_3} \mathfrak{I}(\sharp_{\{\varepsilon\}}(a|_\tau)) \oplus \bigoplus_{\tau\in I_4} \mathfrak{I}(\sharp_{\{\varepsilon\}}(a|_\tau))$$

$$= \bigoplus_{1\le j\le k} p_j \odot \bigoplus_{t\trianglelefteq_\sharp r_j\sigma} \mathfrak{I}(\sharp_{\{\varepsilon\}}(t)) \oplus \bigoplus_{1\le j\le k} p_j \odot \bigoplus_{\tau\in I_1} \mathfrak{I}(\sharp_{\{\varepsilon\}}(a|_\tau[r_j\sigma]_{\kappa_\tau})) \oplus \bigoplus_{\tau\in I_3} \mathfrak{I}(\sharp_{\{\varepsilon\}}(a|_\tau)) \oplus \bigoplus_{\tau\in I_4} \mathfrak{I}(\sharp_{\{\varepsilon\}}(a|_\tau))$$

$$= \bigoplus_{1\le j\le k} p_j \odot \bigoplus_{t\trianglelefteq_\sharp r_j\sigma} \mathfrak{I}(\sharp_{\{\varepsilon\}}(t)) \oplus \bigoplus_{1\le j\le k} p_j \odot \bigoplus_{\tau\in I_1} \mathfrak{I}(\sharp_{\{\varepsilon\}}(a|_\tau[r_j\sigma]_{\kappa_\tau})) \oplus \bigoplus_{1\le j\le k} p_j \odot \bigoplus_{\tau\in I_3} \mathfrak{I}(\sharp_{\{\varepsilon\}}(a|_\tau)) \oplus \bigoplus_{1\le j\le k} p_j \odot \bigoplus_{\tau\in I_4} \mathfrak{I}(\sharp_{\{\varepsilon\}}(a|_\tau))$$

$$= \bigoplus_{1\le j\le k} p_j \odot \Big( \bigoplus_{t \trianglelefteq_\sharp r_j\sigma} \mathfrak{I}(\sharp_{\{\varepsilon\}}(t)) \oplus \bigoplus_{\tau\in I_1} \mathfrak{I}(\sharp_{\{\varepsilon\}}(a|_\tau[r_j\sigma]_{\kappa_\tau})) \oplus \bigoplus_{\tau\in I_3} \mathfrak{I}(\sharp_{\{\varepsilon\}}(a|_\tau)) \oplus \bigoplus_{\tau\in I_4} \mathfrak{I}(\sharp_{\{\varepsilon\}}(a|_\tau)) \Big)$$

$$\downarrow \quad \text{(moving } \tau = \pi.\gamma_\tau.\rho_\tau \in I_3 \text{ via the VRF)}$$

$$\succcurlyeq \bigoplus_{1\le j\le k} p_j \odot \Big( \bigoplus_{t \trianglelefteq_\sharp r_j\sigma} \mathfrak{I}(\sharp_{\{\varepsilon\}}(t)) \oplus \bigoplus_{\tau\in I_1} \mathfrak{I}(\sharp_{\{\varepsilon\}}(a|_\tau[r_j\sigma]_{\kappa_\tau})) \oplus \bigoplus_{\substack{\tau\in I_3 \\ \varphi_j(\gamma_\tau)\neq\bot}} \mathfrak{I}(\sharp_{\{\varepsilon\}}(b_j|_{\pi.\varphi_j(\gamma_\tau).\rho_\tau})) \oplus \bigoplus_{\tau\in I_4} \mathfrak{I}(\sharp_{\{\varepsilon\}}(a|_\tau)) \Big)$$

$$\downarrow \quad (\succcurlyeq, \text{ as VRFs may be non-injective})$$

$$\succcurlyeq \bigoplus_{1\le j\le k} p_j \odot \bigoplus_{t \trianglelefteq_\sharp b_j} \mathfrak{I}(t^\sharp) = \bigoplus_{1\le j\le k} p_j \odot \mathfrak{I}^\sharp_{sum}(b_j).$$

In the case $\pi \notin \mathrm{pos}_{\mathcal{D}^\sharp}(a)$, we need to remove $\mathfrak{I}(\sharp_{\{\varepsilon\}}(\ell)\sigma)$ as this annotated subterm does not exist in $a$, and therefore also $\bigoplus_{t \trianglelefteq_\sharp r_j\sigma} \mathfrak{I}(t^\sharp)$ in the end, leading to the same result. In the case $m = \mathsf{f}$, the $I_1$-summands are removed as well, leading to the same result without them.

**Step 2** Create a CT $\mathfrak{T}^{\le U}$ for any $U \in \mathbb{N}$

Let $\mathfrak{T} = (N, E)$ be an $\mathcal{A}$-CT that satisfies (+). Let $\mathrm{emb} : A \to \mathbb{R}_{\ge 0}$ be the embedding fixed for the collapsibility of $\mathfrak{I}$. We replace emb by $1/\epsilon \cdot \mathrm{emb}$, which is again monotone and concave; then $a \succ_\epsilon b$ implies $\mathrm{emb}(a) \ge \mathrm{emb}(b) + 1$, and by the $\epsilon$-separation, for all terms $t, t'$ the values $\mathrm{emb}(\mathfrak{I}^\sharp_0(t))$ and $\mathrm{emb}(\mathfrak{I}^\sharp_0(t'))$ are either equal or differ by at least 1. We define the *tree value* of any node $v \in N$ in our CT by

$$\mathcal{V}al_{\mathfrak{T}} : N \to \mathbb{R}_{\ge 0}, \qquad v \mapsto \begin{cases} 0, & \text{if } v \in \mathrm{Leaf}, \\ \mathrm{emb}(\mathfrak{I}^\sharp_0(t_v)) + 1, & \text{otherwise}, \end{cases}$$

where $\mathfrak{I}^\sharp_0$ behaves exactly as $\mathfrak{I}^\sharp_{sum}$ but in addition, it maps every variable to $\mathbf{0}$. Remember that $t_v$ denotes the term in the labeling of the node $v$. Since $\mathfrak{I}^\sharp_{sum}(t) \succ \mathfrak{I}^\sharp_{sum}(s)$ and $\mathfrak{I}^\sharp_{sum}(t) \succ_\epsilon \mathfrak{I}^\sharp_{sum}(s)$ hold for all instantiations of the occurring variables, setting every variable to $\mathbf{0}$ shows that they imply $\mathfrak{I}^\sharp_0(t) \succ \mathfrak{I}^\sharp_0(s)$ and $\mathfrak{I}^\sharp_0(t) \succ_\epsilon \mathfrak{I}^\sharp_0(s)$, respectively. Furthermore, note that we have $\mathcal{V}al_{\mathfrak{T}}(v) = 0$ if and only if $v$ is a leaf in $\mathfrak{T}$, and that any two tree values are either equal or differ by at least 1. For any $U \in \mathbb{N}$, we create a modified tree $\mathfrak{T}^{\le U}$, where we cut everything below a node $v$ of the tree with $\mathcal{V}al_{\mathfrak{T}}(v) \ge U + 1$. Let $C = \mathrm{Leaf}^{\mathfrak{T}^{\le U}} \setminus \mathrm{Leaf}^{\mathfrak{T}}$ be the set of all new leaves in $\mathfrak{T}^{\le U}$ due to the cut. So for all $v \in C$ we have $\mathcal{V}al_{\mathfrak{T}}(v) \ge U + 1$.
Our goal is to prove that we have $|\mathfrak{T}| = 1$. First, we prove that $|\mathfrak{T}^{\le U}| = 1$ for any $U \in \mathbb{N}$. However, this does not yet prove that $|\mathfrak{T}| = 1$, which we will show afterwards.

**Step 3** Prove that $|\mathfrak{T}^{\leq U}| \geq p_{min}^{U+1}$ for any $U \in \mathbb{N}$

Note that in a rewrite step $a \hookrightarrow_{\mathcal{A}} \{^{p_1}b_1, \ldots, ^{p_k}b_k\}$ with Case **nt** or **nf** or in a rewrite step with an ADP from $\mathcal{A}_{\succcurlyeq}$, we cannot guarantee that there exists a $b_j$ that is strictly decreasing in value. It is also possible that all values stay the same. Thus, there does not have to be a single witness path of length $U+1$ in $\mathfrak{T}^{\leq U}$ that shows termination with a probability of at least $p_{min}^{U+1}$, as it has been the case in the proof of Theorem 3.4.38 for PTRSs. Instead, we have to use multiple witness paths of finite length to ensure that we have $|\mathfrak{T}^{\leq U}| \geq p_{min}^{U+1}$. Here, the $\epsilon$-gap of $\succ_\epsilon$ together with the $\epsilon$-separation of the embedding ensures that $p_{min}^{U+1} > 0$ is a lower bound on the termination probability. If one wants to adapt this theorem to more complex algebras (that are not collapsible), then one needs to find different ways to obtain a constant lower bound $> 0$ for every $U \in \mathbb{N}$. We first explain how to find such a set of witness paths in a finite sub $\mathcal{A}$-CT that starts with a node from $\Gamma_\succ$, and then we prove by induction that we have $|\mathfrak{T}^{\leq U}| \geq p_{min}^{U+1}$.

**Step 3.1** Find witnesses in subtrees of $\mathfrak{T}^{\leq U}$

In this part, we prove a first observation regarding the existence of certain witnesses that shows a guaranteed decrease of values. Let $\Gamma$ now refer to $\mathfrak{T}^{\leq U}$, i.e., $\Gamma$ only contains inner nodes of $\mathfrak{T}^{\leq U}$ and thus, $\Gamma \cap C = \varnothing$. Moreover, let $\Gamma_\succ$ and $\Gamma_{\succcurlyeq}$ correspond to the rewrite steps at nodes in $\Gamma$ with $\mathcal{A}_\succ$ and $\mathcal{A}_{\succcurlyeq}$, respectively. For every $v \in \Gamma_\succ$, let $\mathfrak{T}_v$ be the subtree that starts at $v$ and where we cut every edge after the second node from $\Gamma_\succ$. This is illustrated in Figure 7.15, where an empty node stands for a leaf that already existed in $\mathfrak{T}^{\leq U}$ (i.e., it is in $\mathrm{Leaf}^{\mathfrak{T}}$ or in $C$). Since $\mathfrak{T}$ satisfies (+) and is finitely branching, we know that $\mathfrak{T}_v$ must be finite. We want to prove that for such a tree $\mathfrak{T}_v$, we have a set of leaves (a set of witnesses) that show a certain decrease of values compared to the root value $\mathcal{V}al_{\mathfrak{T}}(v)$. To be precise, we want to prove that there exists a set $W^v \subseteq \mathrm{Leaf}^{\mathfrak{T}_v}$ of leaves in $\mathfrak{T}_v$ with the following two properties:

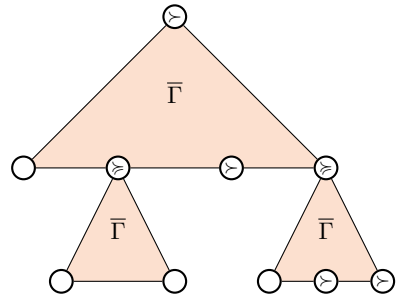



Figure 7.15: The subtree $\mathfrak{T}_v$ that starts at a node $v \in \Gamma_\succ$ and is cut after the second node from $\Gamma_\succ$.

(W-1) For all $w \in W^v$ we have $\overline{\mathcal{V}al}_{\mathfrak{T}}(w) \leq \overline{\mathcal{V}al}_{\mathfrak{T}}(v) - 1$.

(W-2) $\sum_{w \in W^v} p_w^{\mathfrak{T}_v} \cdot p_{min}^{\overline{\mathcal{V}al}_{\mathfrak{T}}(w)} \geq p_{min}^{\overline{\mathcal{V}al}_{\mathfrak{T}}(v)}$.

Here, we use an *adjusted tree value* function $\overline{\mathcal{V}al}_{\mathfrak{T}} : N^{\mathfrak{T}^{\leq U}} \to \mathbb{R}_{\geq 0}$, such that for every node $v \in N^{\mathfrak{T}^{\leq U}}$ we have

$$\overline{\mathcal{V}al}_{\mathfrak{T}}(v) = \begin{cases} 0, & \text{if } v \text{ is a leaf in } \mathfrak{T}^{\leq U}, \\ \mathcal{V}al_{\mathfrak{T}}(v), & \text{otherwise.} \end{cases}$$

This is the same tree value function as before, except for the nodes in $C$. For all $v \in C$ we have $\mathcal{V}al_{\mathfrak{T}}(v) \geq U+1$ and $\overline{\mathcal{V}al}_{\mathfrak{T}}(v) = 0$. Here, $p_{min}^x$ for $x \in \mathbb{R}_{\geq 0}$ denotes the standard real power of $p_{min}$, which is non-increasing in $x$, as $0 < p_{min} \leq 1$. The

first property (W-1) states that all of our witnesses have a strictly smaller value than the root. Furthermore, we have to be careful that the probabilities for our witnesses are not too low. The second property (W-2) states that the sum of all probabilities for the witnesses is still big enough. The additional $p_{min}^{\overline{\mathcal{V}al}_{\mathfrak{T}}(w)}$ is used to allow smaller probabilities for our witnesses if the value decrease is high enough.
In order to show the existence of such a set $W^v$, we prove by induction on $i$ that for all $1 \leq i \leq H$, where $H$ is the height of $\mathfrak{T}_v$, there exists a set of nodes $W_i^v$ such that we have

(W-1!) For all $w \in W_i^v$ we have $\overline{\mathcal{V}al}_{\mathfrak{T}}(w) \leq \overline{\mathcal{V}al}_{\mathfrak{T}}(v) - 1$.

(W-2!) $\sum_{w \in W_i^v} p_w^{\mathfrak{T}_v} \cdot p_{min}^{\overline{\mathcal{V}al}_{\mathfrak{T}}(w)} \geq p_{min}^{\overline{\mathcal{V}al}_{\mathfrak{T}}(v)}$.

(W-3!) Every $w \in W_i^v$ is a leaf or an inner node at the $i$-th depth of $\mathfrak{T}_v$.

Then in the end if $i = H$ is the height of the tree $\mathfrak{T}_v$, we get a set $W^v = W_H^v$ such that every node in $W_H^v$ is a leaf in $\mathfrak{T}_v$ (i.e., $W_H^v \subseteq \mathrm{Leaf}^{\mathfrak{T}_v}$) and both (W-1) and (W-2) are satisfied.
In the induction base, we consider depth $i = 1$ and look at the rewrite step at the root. The first edge represents a rewrite step with $\mathcal{A}_\succ$ at a position $\pi \in \mathrm{pos}_{\mathcal{D}^\sharp}(t_v)$. Let $vE = \{w_1, \ldots, w_k\}$ be the set of the successors of $v$. We have $t_v \hookrightarrow_{\mathcal{A}} \{{}^{p_1}t_{w_1}, \ldots, {}^{p_k}t_{w_k}\}$ using a rule from $\mathcal{A}_\succ$.

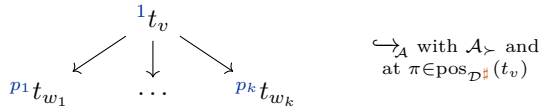



Figure 7.16: The nodes considered in the induction base.

Due to (b) there is a $1 \leq j \leq k$ with $\mathfrak{I}^\sharp_{sum}(t_v) \succ_\epsilon \mathfrak{I}^\sharp_{sum}(t_{w_j})$. By the gap of $\succ_\epsilon$ (rescaled to 1, as explained above) this means $\mathrm{emb}(\mathfrak{I}^\sharp_0(t_v)) \geq \mathrm{emb}(\mathfrak{I}^\sharp_0(t_{w_j})) + 1$, and hence $\mathcal{V}al_{\mathfrak{T}}(v) \geq \mathcal{V}al_{\mathfrak{T}}(w_j) + 1$, which also implies that $\overline{\mathcal{V}al}_{\mathfrak{T}}(v) \geq \overline{\mathcal{V}al}_{\mathfrak{T}}(w_j) + 1$. Since $0 < p_{min} \leq 1$, we therefore have $p_{min}^{\overline{\mathcal{V}al}_{\mathfrak{T}}(v)} \leq p_{min}^{\overline{\mathcal{V}al}_{\mathfrak{T}}(w_j)+1}$. Thus we can set $W_1^v = \{w_j\}$ and have (W-1!) satisfied, since $\overline{\mathcal{V}al}_{\mathfrak{T}}(v) \geq \overline{\mathcal{V}al}_{\mathfrak{T}}(w_j) + 1$ so $\overline{\mathcal{V}al}_{\mathfrak{T}}(w_j) \leq \overline{\mathcal{V}al}_{\mathfrak{T}}(v) - 1$. Property (W-3!) is clearly satisfied and (W-2!) holds as well since we have

$$
\begin{aligned}
&\sum_{w \in W_1^v} p_w^{\mathfrak{T}_v} \cdot p_{min}^{\overline{\mathcal{V}al}_{\mathfrak{T}}(w)} \\
&= p_j \cdot p_{min}^{\overline{\mathcal{V}al}_{\mathfrak{T}}(w_j)} \\
&\quad \downarrow \quad \left(\begin{smallmatrix} p_j \geq p_{min} \,\wedge \\ \overline{\mathcal{V}al}_{\mathfrak{T}}(w_j) \leq \overline{\mathcal{V}al}_{\mathfrak{T}}(v) - 1 \end{smallmatrix}\right) \\
&\geq p_{min} \cdot p_{min}^{\overline{\mathcal{V}al}_{\mathfrak{T}}(v)-1} \\
&= p_{min}^{\overline{\mathcal{V}al}_{\mathfrak{T}}(v)}.
\end{aligned}
$$

In the induction step, we consider depth $i > 1$. Due to the induction hypothesis, there is a set $W_{i-1}^v$ that satisfies (W-1!), (W-2!), and (W-3!). For every node $w \in W_{i-1}^v$ that is not a leaf of $\mathfrak{T}_v$, we proceed as follows: Let $wE = \{w_1, \ldots, w_k\}$ be the set of its successors. We rewrite $t_w$ either with $\hookrightarrow_{\mathcal{A}}$ and an ADP from $\mathcal{A}_{\succcurlyeq}$ or with $\hookrightarrow_{\mathcal{A}}$ and a position $\pi \notin \mathrm{pos}_{\mathcal{D}^\sharp}(t_w)$, which rewrites $t_w$ to a multi-distribution $\{{}^{p_1}t_{w_1}, \ldots, {}^{p_k}t_{w_k}\}$.

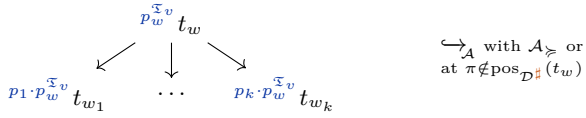



Figure 7.17: The nodes considered in the induction step.

Due to (d), we have $\mathfrak{I}^\sharp_{sum}(t_w) \succcurlyeq \bigoplus_{1\le j\le k} p_j \odot \mathfrak{I}^\sharp_{sum}(t_{w_j})$. Since emb is monotone and concave we get

$$\begin{aligned} \mathrm{emb}(\mathfrak{I}^\sharp_0(t_w)) &\ge \mathrm{emb}\Big(\bigoplus_{1\le j\le k} p_j \odot \mathfrak{I}^\sharp_0(t_{w_j})\Big) \\ &\ge \sum_{1\le j\le k} p_j \cdot \mathrm{emb}(\mathfrak{I}^\sharp_0(t_{w_j})) \\ \implies \qquad \overline{\mathcal{V}al}_{\mathfrak{T}}(w) &\ge \sum_{1\le j\le k} p_j \cdot \overline{\mathcal{V}al}_{\mathfrak{T}}(w_j). \end{aligned}$$

Hence, we either have $\overline{\mathcal{V}al}_{\mathfrak{T}}(w) = \overline{\mathcal{V}al}_{\mathfrak{T}}(w_j)$ for all $1 \le j \le k$ or there exists at least one $1 \le j \le k$ with $\overline{\mathcal{V}al}_{\mathfrak{T}}(w) > \overline{\mathcal{V}al}_{\mathfrak{T}}(w_j)$.
We partition the set $W^v_{i-1}$ into the disjoint subsets $W^{v(1)}_{i-1}$, $W^{v(2)}_{i-1}$, and $W^{v(3)}_{i-1}$, where

- $w \in W^{v(1)}_{i-1} :\Leftrightarrow w \in W^v_{i-1}$ and $w$ is a leaf of $\mathfrak{T}_v$.
- $w \in W^{v(2)}_{i-1} :\Leftrightarrow w \in W^v_{i-1}$ and $\overline{\mathcal{V}al}_{\mathfrak{T}}(w) = \overline{\mathcal{V}al}_{\mathfrak{T}}(w_j)$ for all $1 \le j \le k$.
- $w \in W^{v(3)}_{i-1} :\Leftrightarrow w \in W^v_{i-1}$ and there exists a $1 \le j \le k$ with $\overline{\mathcal{V}al}_{\mathfrak{T}}(w) > \overline{\mathcal{V}al}_{\mathfrak{T}}(w_j)$. We denote this node $w_j$ by $w^+$.

In the first case, $w$ is already contained in $W^v_{i-1}$. In the second case, we get $\overline{\mathcal{V}al}_{\mathfrak{T}}(w) = \overline{\mathcal{V}al}_{\mathfrak{T}}(w_j)$ for all $1 \le j \le k$. And in the third case, we have $\overline{\mathcal{V}al}_{\mathfrak{T}}(w) > \overline{\mathcal{V}al}_{\mathfrak{T}}(w^+)$, hence $\overline{\mathcal{V}al}_{\mathfrak{T}}(w^+) \le \overline{\mathcal{V}al}_{\mathfrak{T}}(w) - 1$, as any two tree values are either equal or differ by at least 1. (This is where the $\epsilon$-separation of emb on interpretations of terms is needed: The node $w^+$ comes from a *weakly* decreasing step, so its decrease is not backed by an $\epsilon$-gap, and without separation it could be arbitrarily small.) So for all of these nodes, we can be sure that the value does not increase. We can now define $W^v_i$ as:

$$\begin{aligned} W^v_i \quad &= \quad W^{v(1)}_{i-1} \\ &\cup \quad \textstyle\bigcup_{w \in W^{v(2)}_{i-1}} wE \\ &\cup \quad \{w^+ \mid w \in W^{v(3)}_{i-1}\}. \end{aligned}$$

Intuitively, this means that every leaf remains inside the set of witnesses ($W^{v(1)}_{i-1}$) and for every inner node $w$ we have two cases. If there exists a successor $w^+$ with a strictly smaller value, then we replace the node $w$ by this successor $w^+$ in our set of witnesses ($\{w^+ \mid w \in W^{v(3)}_{i-1}\}$). Otherwise, all the successors $w_1, \ldots, w_k$ of the node $w$ have the same value as the node $w$ itself, so that we have to replace $w$ with all of its successors in our set of witnesses as there is no single node with a guaranteed value decrease ($\bigcup_{w \in W^{v(2)}_{i-1}} wE$).
It remains to show that our properties (W-1!), (W-2!), and (W-3!) are still satisfied for $W^v_i$. In order to see that (W-1!) is satisfied, note that we have $\overline{\mathcal{V}al}_{\mathfrak{T}}(w) \le \overline{\mathcal{V}al}_{\mathfrak{T}}(v) - 1$, for all $w \in W^v_{i-1}$ by our induction hypothesis. Thus, for all $w' \in W^{v(1)}_{i-1} \subseteq W^v_{i-1}$,

we also obtain $\overline{\mathcal{V}al}_{\mathfrak{T}}(w') \leq \overline{\mathcal{V}al}_{\mathfrak{T}}(v) - 1$. For all $w' \in \bigcup_{w \in W_{i-1}^{v(2)}} wE$, we have $\overline{\mathcal{V}al}_{\mathfrak{T}}(w') = \overline{\mathcal{V}al}_{\mathfrak{T}}(w)$ for some $w \in W_{i-1}^{v}$ and thus $\overline{\mathcal{V}al}_{\mathfrak{T}}(w') = \overline{\mathcal{V}al}_{\mathfrak{T}}(w) \leq \overline{\mathcal{V}al}_{\mathfrak{T}}(v) - 1$. Finally, for all $w' \in \{w^+ \mid w \in W_{i-1}^{v(3)}\}$, we have $\overline{\mathcal{V}al}_{\mathfrak{T}}(w') \leq \overline{\mathcal{V}al}_{\mathfrak{T}}(w) - 1$ for some $w \in W_{i-1}^{v}$ and thus $\overline{\mathcal{V}al}_{\mathfrak{T}}(w') \leq \overline{\mathcal{V}al}_{\mathfrak{T}}(w) - 1 \leq \overline{\mathcal{V}al}_{\mathfrak{T}}(v) - 2 \leq \overline{\mathcal{V}al}_{\mathfrak{T}}(v) - 1$.
Property (W-3!) holds as well, since every node $w \in W_{i-1}^{v}$ that is not a leaf of $\mathfrak{T}_v$ is at depth $i-1$ of the tree $\mathfrak{T}_v$ by our induction hypothesis. We exchange each such node with one or all of its successors. The leaves in $W_{i-1}^{v}$ are at a depth of at most $i-1$ by induction hypothesis and remain in $W_i^v$. Hence, all the nodes of $W_i^v$ are at a depth of at most $i$, and the nodes that are not leaves are at a depth of precisely $i$.
Finally, we regard (W-2!). For $\bigcup_{w \in W_{i-1}^{v(2)}} wE$ we have:

$$
\begin{aligned}
&\sum_{w' \in \bigcup_{w \in W_{i-1}^{v(2)}} wE} p_{w'}^{\mathfrak{T}_v} \cdot p_{min}^{\overline{\mathcal{V}al}_{\mathfrak{T}}(w')} \\
&= \sum_{w \in W_{i-1}^{v(2)}} \sum_{w' \in wE} p_{w'}^{\mathfrak{T}_v} \cdot p_{min}^{\overline{\mathcal{V}al}_{\mathfrak{T}}(w')} \\
&= \sum_{\substack{w \in W_{i-1}^{v(2)} \\ wE = \{w_1, \ldots, w_k\}}} \sum_{1 \leq j \leq k} p_{w_j}^{\mathfrak{T}_v} \cdot p_{min}^{\overline{\mathcal{V}al}_{\mathfrak{T}}(w_j)} \\
&\quad \downarrow \quad (\text{as } \overline{\mathcal{V}al}_{\mathfrak{T}}(w_j) = \overline{\mathcal{V}al}_{\mathfrak{T}}(w) \text{ for all } 1 \leq j \leq k) \\
&= \sum_{\substack{w \in W_{i-1}^{v(2)} \\ wE = \{w_1, \ldots, w_k\}}} \sum_{1 \leq j \leq k} p_{w_j}^{\mathfrak{T}_v} \cdot p_{min}^{\overline{\mathcal{V}al}_{\mathfrak{T}}(w)} \\
&\quad \downarrow \quad (\text{as } p_{w_j}^{\mathfrak{T}_v} = p_j \cdot p_w^{\mathfrak{T}_v} \text{ for all } 1 \leq j \leq k) \\
&= \sum_{\substack{w \in W_{i-1}^{v(2)} \\ wE = \{w_1, \ldots, w_k\}}} \sum_{1 \leq j \leq k} p_j \cdot p_w^{\mathfrak{T}_v} \cdot p_{min}^{\overline{\mathcal{V}al}_{\mathfrak{T}}(w)} \\
&= \sum_{\substack{w \in W_{i-1}^{v(2)} \\ wE = \{w_1, \ldots, w_k\}}} p_w^{\mathfrak{T}_v} \cdot p_{min}^{\overline{\mathcal{V}al}_{\mathfrak{T}}(w)} \cdot \sum_{1 \leq j \leq k} p_j \\
&\quad \downarrow \quad (\text{as } \sum_{1 \leq j \leq k} p_j = 1) \\
&= \sum_{w \in W_{i-1}^{v(2)}} p_w^{\mathfrak{T}_v} \cdot p_{min}^{\overline{\mathcal{V}al}_{\mathfrak{T}}(w)} \cdot 1 \\
&= \sum_{w \in W_{i-1}^{v(2)}} p_w^{\mathfrak{T}_v} \cdot p_{min}^{\overline{\mathcal{V}al}_{\mathfrak{T}}(w)}.
\end{aligned}
$$

For $\{w^+ \mid w \in W_{i-1}^{v(3)}\}$ we have:

$$
\begin{aligned}
&\sum_{w' \in \{w^+ \mid w \in W_{i-1}^{v(3)}\}} p_{w'}^{\mathfrak{T}_v} \cdot p_{min}^{\overline{\mathcal{V}al}_{\mathfrak{T}}(w')} \\
=\ &\sum_{w \in W_{i-1}^{v(3)}} p_{w^+}^{\mathfrak{T}_v} \cdot p_{min}^{\overline{\mathcal{V}al}_{\mathfrak{T}}(w^+)} \\
&\quad \downarrow \quad (\text{as } p_{w^+}^{\mathfrak{T}_v} \geq p_{min} \cdot p_w^{\mathfrak{T}_v}) \\
\geq\ &\sum_{w \in W_{i-1}^{v(3)}} p_{min} \cdot p_w^{\mathfrak{T}_v} \cdot p_{min}^{\overline{\mathcal{V}al}_{\mathfrak{T}}(w^+)} \\
&\quad \downarrow \quad (\text{as } \overline{\mathcal{V}al}_{\mathfrak{T}}(w^+) \leq \overline{\mathcal{V}al}_{\mathfrak{T}}(w) - 1) \\
\geq\ &\sum_{w \in W_{i-1}^{v(3)}} p_{min} \cdot p_w^{\mathfrak{T}_v} \cdot p_{min}^{\overline{\mathcal{V}al}_{\mathfrak{T}}(w)-1} \\
=\ &\sum_{w \in W_{i-1}^{v(3)}} p_w^{\mathfrak{T}_v} \cdot p_{min}^{\overline{\mathcal{V}al}_{\mathfrak{T}}(w)}.
\end{aligned}
$$

To summarize, we have

$$
\begin{aligned}
&\sum_{w' \in W_i^v} p_{w'}^{\mathfrak{T}_v} \cdot p_{min}^{\overline{\mathcal{V}al}_{\mathfrak{T}}(w')} \\
=\ &\sum_{w' \in W_{i-1}^{v(1)}} p_{w'}^{\mathfrak{T}_v} \cdot p_{min}^{\overline{\mathcal{V}al}_{\mathfrak{T}}(w')} \\
&\quad + \sum_{w' \in \bigcup_{w \in W_{i-1}^{v(2)}} wE} p_{w'}^{\mathfrak{T}_v} \cdot p_{min}^{\overline{\mathcal{V}al}_{\mathfrak{T}}(w')} \\
&\qquad + \sum_{w' \in \{w^+ \mid w \in W_{i-1}^{v(3)}\}} p_{w'}^{\mathfrak{T}_v} \cdot p_{min}^{\overline{\mathcal{V}al}_{\mathfrak{T}}(w')} \\
\geq\ &\sum_{w \in W_{i-1}^{v(1)}} p_w^{\mathfrak{T}_v} \cdot p_{min}^{\overline{\mathcal{V}al}_{\mathfrak{T}}(w)} \\
&\quad + \sum_{w \in W_{i-1}^{v(2)}} p_w^{\mathfrak{T}_v} \cdot p_{min}^{\overline{\mathcal{V}al}_{\mathfrak{T}}(w)} \\
&\qquad + \sum_{w \in W_{i-1}^{v(3)}} p_w^{\mathfrak{T}_v} \cdot p_{min}^{\overline{\mathcal{V}al}_{\mathfrak{T}}(w)} \\
\overset{(IH)}{\geq}\ &p_{min}^{\overline{\mathcal{V}al}_{\mathfrak{T}}(v)}.
\end{aligned}
$$

**Step 3.2** Prove that $|\mathfrak{T}^{\leq U}| \geq p_{min}^{U+1}$ for any $U \in \mathbb{N}$

We now prove that $|\mathfrak{T}^{\leq U}| \geq p_{min}^{U+1}$ holds for any $U \in \mathbb{N}$. Let $Z_k$ denote the set of nodes in $\mathfrak{T}^{\leq U}$ from $\Gamma_\succ$ that have precisely $k-1$ nodes from $\Gamma_\succ$ above them (so they are themselves the $k$-th node from $\Gamma_\succ$). Here, $\Gamma$, $\Gamma_\succcurlyeq$, and $\Gamma_\succ$ again refer to $\mathfrak{T}^{\leq U}$,

i.e., we again have $\Gamma \cap C = \varnothing$. Let $\mathrm{Leaf}_k$ denote the set of all leaves in $\mathfrak{T}^{\leq U}$ that are reachable by a path that uses fewer than $k$ nodes from $\Gamma_{\succ}$. We show by induction that for all $1 \leq k \leq U+1$, we have

$$\sum_{\substack{v \in Z_k \cup \mathrm{Leaf}_k \\ 0 \leq \overline{\mathcal{V}al}_{\mathfrak{T}}(v) < U+2-k}} p_v^{\mathfrak{T}^{\leq U}} \cdot p_{min}^{\overline{\mathcal{V}al}_{\mathfrak{T}}(v)} \geq p_{min}^{U+1}.$$

Then, for $k = U+1$ we finally have:

$$\begin{aligned}
p_{min}^{U+1} &\leq \sum_{\substack{v \in Z_{U+1} \cup \mathrm{Leaf}_{U+1} \\ 0 \leq \overline{\mathcal{V}al}_{\mathfrak{T}}(v) < U+2-(U+1)}} p_v^{\mathfrak{T}^{\leq U}} \cdot p_{min}^{\overline{\mathcal{V}al}_{\mathfrak{T}}(v)} \\
&= \sum_{\substack{v \in Z_{U+1} \cup \mathrm{Leaf}_{U+1} \\ \overline{\mathcal{V}al}_{\mathfrak{T}}(v) < 1}} p_v^{\mathfrak{T}^{\leq U}} \cdot p_{min}^{\overline{\mathcal{V}al}_{\mathfrak{T}}(v)} \\
&\quad \downarrow \quad \text{(as nonzero tree values are at least 1)} \\
&= \sum_{\substack{v \in Z_{U+1} \cup \mathrm{Leaf}_{U+1} \\ \overline{\mathcal{V}al}_{\mathfrak{T}}(v) = 0}} p_v^{\mathfrak{T}^{\leq U}} \cdot p_{min}^{\overline{\mathcal{V}al}_{\mathfrak{T}}(v)} \\
&= \sum_{\substack{v \in Z_{U+1} \cup \mathrm{Leaf}_{U+1} \\ \overline{\mathcal{V}al}_{\mathfrak{T}}(v) = 0}} p_v^{\mathfrak{T}^{\leq U}} \cdot 1 \\
&= \sum_{\substack{v \in Z_{U+1} \cup \mathrm{Leaf}_{U+1} \\ \overline{\mathcal{V}al}_{\mathfrak{T}}(v) = 0}} p_v^{\mathfrak{T}^{\leq U}} \\
&\quad \downarrow \quad \text{(as } \overline{\mathcal{V}al}_{\mathfrak{T}}(v) = 0 \text{ iff } v \text{ is a leaf of } \mathfrak{T}^{\leq U}\text{)} \\
&= \sum_{v \in \mathrm{Leaf}_{U+1}} p_v^{\mathfrak{T}^{\leq U}} \\
&\leq \sum_{v \in \mathrm{Leaf}^{\mathfrak{T}^{\leq U}}} p_v^{\mathfrak{T}^{\leq U}} \;=\; |\mathfrak{T}^{\leq U}|.
\end{aligned}$$

In the induction base, we have $k = 1$, and thus

$$\begin{aligned}
&\sum_{\substack{v \in Z_1 \cup \mathrm{Leaf}_1 \\ 0 \leq \overline{\mathcal{V}al}_{\mathfrak{T}}(v) < U+2-1}} p_v^{\mathfrak{T}^{\leq U}} \cdot p_{min}^{\overline{\mathcal{V}al}_{\mathfrak{T}}(v)} \\
= &\sum_{\substack{v \in Z_1 \cup \mathrm{Leaf}_1 \\ 0 \leq \overline{\mathcal{V}al}_{\mathfrak{T}}(v) < U+1}} p_v^{\mathfrak{T}^{\leq U}} \cdot p_{min}^{\overline{\mathcal{V}al}_{\mathfrak{T}}(v)} \\
&\quad \downarrow \quad \text{(since } \overline{\mathcal{V}al}_{\mathfrak{T}}(v) < U+1\text{, and hence } p_{min}^{\overline{\mathcal{V}al}_{\mathfrak{T}}(v)} \geq p_{min}^{U+1}\text{)} \\
\geq &\sum_{\substack{v \in Z_1 \cup \mathrm{Leaf}_1 \\ 0 \leq \overline{\mathcal{V}al}_{\mathfrak{T}}(v) < U+1}} p_v^{\mathfrak{T}^{\leq U}} \cdot p_{min}^{U+1}
\end{aligned}$$

$$
\begin{aligned}
&= p_{min}^{U+1} \cdot \sum_{\substack{v \in Z_1 \cup \mathrm{Leaf}_1 \\ 0 \le \overline{\mathcal{V}al}_{\mathfrak{T}}(v) < U+1}} p_v^{\mathfrak{T}^{\le U}} \\
&\quad \downarrow \quad \text{(inner nodes satisfy } \overline{\mathcal{V}al}_{\mathfrak{T}}(v) \le \mathcal{V}al_{\mathfrak{T}}(v) < U+1, \\
&\qquad \text{and leaves have } \overline{\mathcal{V}al}_{\mathfrak{T}}(v) = 0) \\
&= p_{min}^{U+1} \cdot \sum_{v \in Z_1 \cup \mathrm{Leaf}_1} p_v^{\mathfrak{T}^{\le U}} \\
&\quad \downarrow \quad \text{(since } \sum_{v \in Z_1 \cup \mathrm{Leaf}_1} p_v^{\mathfrak{T}^{\le U}} = 1) \\
&= p_{min}^{U+1} \cdot 1 \;=\; p_{min}^{U+1}.
\end{aligned}
$$

Here, we have $\sum_{v \in Z_1 \cup \mathrm{Leaf}_1} p_v^{\mathfrak{T}^{\le U}} = 1$, since $Z_1 \cup \mathrm{Leaf}_1$ are the leaves of the finite subtree where we cut everything below the first node of $\mathcal{A}_\succ$ (i.e., we cut directly after the nodes in $Z_1$). This tree is finite, because by (+) there is no infinite path without $\Gamma_\succ$ nodes. All finite CTs converge with probability 1.

In the induction step, we assume that the statement holds for some $1 \le k \le U$. Then

$$
\begin{aligned}
p_{min}^{U+1} &\overset{(IH)}{\le} \sum_{\substack{v \in Z_k \cup \mathrm{Leaf}_k \\ 0 \le \overline{\mathcal{V}al}_{\mathfrak{T}}(v) < U+2-k}} p_v^{\mathfrak{T}^{\le U}} \cdot p_{min}^{\overline{\mathcal{V}al}_{\mathfrak{T}}(v)} \\
&= \sum_{v \in \mathrm{Leaf}_k} p_v^{\mathfrak{T}^{\le U}} \cdot p_{min}^{\overline{\mathcal{V}al}_{\mathfrak{T}}(v)} + \sum_{\substack{v \in Z_k \\ 1 \le \overline{\mathcal{V}al}_{\mathfrak{T}}(v) < U+2-k}} p_v^{\mathfrak{T}^{\le U}} \cdot p_{min}^{\overline{\mathcal{V}al}_{\mathfrak{T}}(v)} \\
&\quad \downarrow \quad \text{(existence of the set } W^v \text{ by the previous Step 3.1 and (W-2))} \\
&\le \sum_{v \in \mathrm{Leaf}_k} p_v^{\mathfrak{T}^{\le U}} \cdot p_{min}^{\overline{\mathcal{V}al}_{\mathfrak{T}}(v)} + \sum_{\substack{v \in Z_k \\ 1 \le \overline{\mathcal{V}al}_{\mathfrak{T}}(v) < U+2-k}} p_v^{\mathfrak{T}^{\le U}} \cdot \sum_{w \in W^v} p_w^{\mathfrak{T}_v} \cdot p_{min}^{\overline{\mathcal{V}al}_{\mathfrak{T}}(w)} \\
&= \sum_{v \in \mathrm{Leaf}_k} p_v^{\mathfrak{T}^{\le U}} \cdot p_{min}^{\overline{\mathcal{V}al}_{\mathfrak{T}}(v)} + \sum_{\substack{v \in Z_k \\ 1 \le \overline{\mathcal{V}al}_{\mathfrak{T}}(v) < U+2-k}} \sum_{w \in W^v} p_v^{\mathfrak{T}^{\le U}} \cdot p_w^{\mathfrak{T}_v} \cdot p_{min}^{\overline{\mathcal{V}al}_{\mathfrak{T}}(w)} \\
&\quad \downarrow \quad \text{(as } p_v^{\mathfrak{T}^{\le U}} \cdot p_w^{\mathfrak{T}_v} = p_w^{\mathfrak{T}^{\le U}}\text{)} \\
&= \sum_{v \in \mathrm{Leaf}_k} p_v^{\mathfrak{T}^{\le U}} \cdot p_{min}^{\overline{\mathcal{V}al}_{\mathfrak{T}}(v)} + \sum_{\substack{v \in Z_k \\ 1 \le \overline{\mathcal{V}al}_{\mathfrak{T}}(v) < U+2-k}} \sum_{w \in W^v} p_w^{\mathfrak{T}^{\le U}} \cdot p_{min}^{\overline{\mathcal{V}al}_{\mathfrak{T}}(w)}.
\end{aligned}
$$

Note that the sets $W^v$ for the different nodes $v \in Z_k$ are pairwise disjoint, since the nodes of $Z_k$ are pairwise incomparable (none of them is above another one) and $W^v$ only contains nodes of $\mathfrak{T}_v$. Every node in $W^v$ is either contained in $\mathrm{Leaf}_{k+1}$ or contained in $Z_{k+1}$. The reason for that is that $W^v$ only contains leaves of $\mathfrak{T}_v$ and a leaf in $\mathfrak{T}_v$ is either also a leaf in $\mathfrak{T}^{\le U}$ so that it is contained in $\mathrm{Leaf}_{k+1}$, or contained in $\Gamma_\succ$ and thus in $Z_{k+1}$, since $v$ is contained in $Z_k$ and there is no other inner node from $\Gamma_\succ$ in $\mathfrak{T}_v$. Furthermore, we know that $\overline{\mathcal{V}al}_{\mathfrak{T}}(w) \le \overline{\mathcal{V}al}_{\mathfrak{T}}(v) - 1$ for all $w \in W^v$ by (W-1). Moreover, we have $\mathrm{Leaf}_k \subseteq \mathrm{Leaf}_{k+1}$. Thus, we get

$$
\sum_{v \in \mathrm{Leaf}_k} p_v^{\mathfrak{T}^{\le U}} \cdot p_{min}^{\overline{\mathcal{V}al}_{\mathfrak{T}}(v)} + \sum_{\substack{v \in Z_k \\ 1 \le \overline{\mathcal{V}al}_{\mathfrak{T}}(v) < U+2-k}} \sum_{w \in W^v} p_w^{\mathfrak{T}^{\le U}} \cdot p_{min}^{\overline{\mathcal{V}al}_{\mathfrak{T}}(w)}
$$

$$\leq \sum_{\substack{w \in Z_{k+1} \cup \mathrm{Leaf}_{k+1} \\ 0 \leq \overline{\mathcal{V}al}_{\mathfrak{T}}(w) < U+2-(k+1)}} p_w^{\mathfrak{T}^{\leq U}} \cdot p_{min}^{\overline{\mathcal{V}al}_{\mathfrak{T}}(w)}.$$

To summarize, we have shown that $|\mathfrak{T}^{\leq U}| \geq p_{min}^{U+1}$.

**Step 4** Prove that $|\mathfrak{T}^{\leq U}| = 1$ for any $U \in \mathbb{N}$

We have proven that $|\mathfrak{T}^{\leq U}| \geq p_{min}^{U+1}$ holds for all $\mathcal{A}$-CTs $\mathfrak{T}$ that satisfy (+). Hence, for any $U \in \mathbb{N}$, we have

$$p_U^\star = \inf_{\mathfrak{T} \text{ is an } \mathcal{A}\text{-CT satisfying (+)}} (|\mathfrak{T}^{\leq U}|) \geq p_{min}^{U+1} > 0. \tag{7.39}$$

We now prove by contradiction that this is enough to ensure $p_U^\star = 1$. So assume that $p_U^\star < 1$. Then we define $\epsilon' = \frac{p_U^\star \cdot (1-p_U^\star)}{2} > 0$. By definition of the infimum, $p_U^\star + \epsilon'$ is not a lower bound of $|\mathfrak{T}^{\leq U}|$ for all $\mathcal{A}$-CTs $\mathfrak{T}$ that satisfy (+). Hence, there must exist an $\mathcal{A}$-CT $\mathfrak{T}$ satisfying (+) such that

$$p_U^\star \leq |\mathfrak{T}^{\leq U}| < p_U^\star + \epsilon'. \tag{7.40}$$

For readability, let $Z = \mathrm{Leaf}^{\mathfrak{T}^{\leq U}}$ be the set of leaves of the tree $\mathfrak{T}^{\leq U}$ and let $\overline{Z} = N^{\mathfrak{T}^{\leq U}} \setminus \mathrm{Leaf}^{\mathfrak{T}^{\leq U}}$ be the set of inner nodes of the tree $\mathfrak{T}^{\leq U}$. Since $p_v^{\mathfrak{T}^{\leq U}} = p_v^{\mathfrak{T}}$ for all $v \in N^{\mathfrak{T}^{\leq U}}$, in the following we just write $p_v$. By the monotonicity of $|\cdot|$ w.r.t. the depth of the tree $\mathfrak{T}^{\leq U}$, there must exist a natural number $m^\star \in \mathbb{N}$ such that

$$\sum_{\substack{v \in Z \\ \mathrm{d}(v) \leq m^*}} p_v > \frac{p_U^\star}{2}. \tag{7.41}$$

Here, $\mathrm{d}(v)$ denotes the depth of node $v$ in $\mathfrak{T}^{\leq U}$. For every $v \in N^{\mathfrak{T}^{\leq U}}$ with $\mathrm{d}(v) = m^\star$ and $v \notin Z$, we define the sub $\mathcal{A}$-CT of $\mathfrak{T}^{\leq U}$ starting at node $v$ by $\mathfrak{T}^{\leq U}(v) = \mathfrak{T}^{\leq U}[v(E^{\mathfrak{T}^{\leq U}})^*]$. Then we have

$$|\mathfrak{T}^{\leq U}| = \sum_{\substack{v \in Z \\ \mathrm{d}(v) \leq m^*}} p_v + \sum_{\substack{v \in \overline{Z} \\ \mathrm{d}(v) = m^*}} p_v \cdot |\mathfrak{T}^{\leq U}(v)|. \tag{7.42}$$

Furthermore, we have

$$\sum_{\substack{v \in \overline{Z} \\ \mathrm{d}(v) = m^*}} p_v = 1 - \sum_{\substack{v \in Z \\ \mathrm{d}(v) \leq m^*}} p_v, \tag{7.43}$$

since $\sum_{\substack{v \in \overline{Z} \\ \mathrm{d}(v) = m^*}} p_v + \sum_{\substack{v \in Z \\ \mathrm{d}(v) \leq m^*}} p_v = 1$, as the nodes $v \in \overline{Z}$ with $\mathrm{d}(v) = m^*$ are the leaves of the finite, grounded subtree of $\mathfrak{T}^{\leq U}$ where we cut every edge after the nodes of depth $m^*$. We obtain

$$\begin{aligned} & p_U^\star + \epsilon' \\ & \quad \downarrow \quad \text{(by (7.40))} \\ & > |\mathfrak{T}^{\leq U}| \end{aligned}$$

$$
\begin{aligned}
&\downarrow \quad \text{(by (7.42) and (7.39))} \\
&= \sum_{\substack{v \in Z \\ \mathrm{d}(v) \leq m^*}} p_v + \sum_{\substack{v \in \overline{Z} \\ \mathrm{d}(v) = m^*}} p_v \cdot \underbrace{|\mathfrak{T}^{\leq U}(v)|}_{\geq p_U^\star} \\
&\geq \sum_{\substack{v \in Z \\ \mathrm{d}(v) \leq m^*}} p_v + \sum_{\substack{v \in \overline{Z} \\ \mathrm{d}(v) = m^*}} p_v \cdot p_U^\star \\
&= \sum_{\substack{v \in Z \\ \mathrm{d}(v) \leq m^*}} p_v + p_U^\star \cdot \sum_{\substack{v \in \overline{Z} \\ \mathrm{d}(v) = m^*}} p_v \\
&\quad \downarrow \quad \text{(by (7.43))} \\
&= \sum_{\substack{v \in Z \\ \mathrm{d}(v) \leq m^*}} p_v + p_U^\star \cdot (1 - \sum_{\substack{v \in Z \\ \mathrm{d}(v) \leq m^*}} p_v) \\
&= p_U^\star + \sum_{\substack{v \in Z \\ \mathrm{d}(v) \leq m^*}} p_v - p_U^\star \cdot \sum_{\substack{v \in Z \\ \mathrm{d}(v) \leq m^*}} p_v \\
&= p_U^\star + \sum_{\substack{v \in Z \\ \mathrm{d}(v) \leq m^*}} p_v \cdot (1 - p_U^\star) \\
&\quad \downarrow \quad \text{(by (7.41))} \\
&> p_U^\star + (1 - p_U^\star) \cdot \tfrac{p_U^\star}{2} \;=\; p_U^\star + \epsilon', \quad \lightning
\end{aligned}
$$

which is a contradiction. So $p_U^\star = 1$. In particular, this means that for every $U \in \mathbb{N}$ and every $\mathcal{A}$-CT $\mathfrak{T}$ satisfying (+), we have

$$|\mathfrak{T}^{\leq U}| = 1. \tag{7.44}$$

**Step 5** Finally, we prove that $|\mathfrak{T}| = 1$

We adjust the tree value function for this new tree $\mathfrak{T}^{\leq U}$ once again and define:

$$\widehat{\mathcal{V}al}_{\mathfrak{T}} : N^{\mathfrak{T}^{\leq U}} \to \mathbb{R}_{\geq 0}, \qquad v \mapsto \begin{cases} U+1, & \text{if } v \in C, \\ 0, & \text{if } v \in \mathrm{Leaf}^{\mathfrak{T}}, \\ \mathrm{emb}(\mathfrak{I}_0^\sharp(t_v)) + 1, & \text{otherwise.} \end{cases}$$

Now for a node $v \in N^{\mathfrak{T}^{\leq U}}$ we have $0 \leq \widehat{\mathcal{V}al}_{\mathfrak{T}}(v) \leq U+1$. Furthermore, we have $\mathcal{V}al_{\mathfrak{T}}(v) \geq \widehat{\mathcal{V}al}_{\mathfrak{T}}(v)$ for all $v \in N^{\mathfrak{T}^{\leq U}}$. Let $|\mathfrak{T}^{\leq U}|_{\widehat{\mathcal{V}al}_{\mathfrak{T}}} = \sum_{v \in \mathrm{Leaf}^{\mathfrak{T}^{\leq U}}} p_v \cdot \widehat{\mathcal{V}al}_{\mathfrak{T}}(v)$. This (possibly infinite) sum is well defined (i.e., it is a convergent series), because all addends are non-negative and the sum is bounded from above by $U+1$, since

$$\sum_{v \in \mathrm{Leaf}^{\mathfrak{T}^{\leq U}}} p_v \cdot \widehat{\mathcal{V}al}_{\mathfrak{T}}(v) \leq \sum_{v \in \mathrm{Leaf}^{\mathfrak{T}^{\leq U}}} p_v \cdot (U+1) = (U+1) \cdot \sum_{v \in \mathrm{Leaf}^{\mathfrak{T}^{\leq U}}} p_v \leq (U+1) \cdot 1 = U+1.$$

For the root $\mathfrak{r}$ of $\mathfrak{T}$ and $\mathfrak{T}^{\leq U}$, we have $\widehat{\mathcal{V}al}_{\mathfrak{T}}(\mathfrak{r}) \geq |\mathfrak{T}^{\leq U}|_{\widehat{\mathcal{V}al}_{\mathfrak{T}}}$ because due to (d), we know that the (expected) value is non-increasing: every inner node $w$ satisfies $\widehat{\mathcal{V}al}_{\mathfrak{T}}(w) \geq \sum_j p_j \cdot \widehat{\mathcal{V}al}_{\mathfrak{T}}(w_j)$, and the bound follows by induction over the depth and taking the limit.

Now we fix $U \in \mathbb{N}$ and a CT $\mathfrak{T}$ satisfying (+), and obtain the corresponding transformed tree $\mathfrak{T}^{\leq U}$. Note that by (7.44) we have $|\mathfrak{T}^{\leq U}| = 1 = q_U + q'_U$, where $q_U = \sum_{v \in \mathrm{Leaf}^{\mathfrak{T}^{\leq U}}, \widehat{\mathcal{V}al}_{\mathfrak{T}}(v) = 0} p_v$ and $q'_U = \sum_{v \in \mathrm{Leaf}^{\mathfrak{T}^{\leq U}}, \widehat{\mathcal{V}al}_{\mathfrak{T}}(v) > 0} p_v = \sum_{v \in C} p_v$. Now we can determine $|\mathfrak{T}^{\leq U}|_{\widehat{\mathcal{V}al}_{\mathfrak{T}}}$. The probabilities of zero-valued nodes (i.e., leaves in the original tree) add up to $q_U$, while the probabilities of new leaves due to a cut add up to

probability $q'_U = 1 - q_U$. So $|\mathfrak{T}^{\leq U}|_{\widehat{\mathcal{V}al}_\mathfrak{T}} = q_U \cdot 0 + (1 - q_U) \cdot (U + 1) = (1 - q_U) \cdot (U + 1)$. Thus,

$$\mathcal{V}al_\mathfrak{T}(\mathfrak{r}) \geq \widehat{\mathcal{V}al}_\mathfrak{T}(\mathfrak{r}) \geq |\mathfrak{T}^{\leq U}|_{\widehat{\mathcal{V}al}_\mathfrak{T}} \geq (1 - q_U) \cdot (U + 1),$$

which implies $q_U \geq 1 - \frac{\mathcal{V}al_\mathfrak{T}(\mathfrak{r})}{U+1}$. Note that $q_U$ is weakly monotonically increasing and bounded from above by 1 for $U \to \infty$. Hence, $q = \lim_{U\to\infty} q_U$ exists and $1 \geq q \geq \lim_{U\to\infty}(1 - \frac{\mathcal{V}al_\mathfrak{T}(\mathfrak{r}^\mathfrak{T})}{U+1}) = 1$, i.e., $q = 1$. Moreover, we have $q_U \leq |\mathfrak{T}|$ for every $U \in \mathbb{N}$, since every true leaf reached in $\mathfrak{T}^{\leq U}$ lies on a branch without any cut and is thus also reached in $\mathfrak{T}$. Hence, we obtain $|\mathfrak{T}| \geq \lim_{U\to\infty} q_U = q = 1$, i.e., $|\mathfrak{T}| = 1$. *QED*

Figure 7.18 visualizes the interpretation of terms in a chain tree. The expected $\sharp$-sum is weakly decreasing in each step from a node to its children. Moreover, the left successor $t_1$ has a strictly smaller $\sharp$-sum than $t$. The $\sharp$-sum of the right successor $t_2$ may even be larger than $t$, but the expected $\sharp$-sum needs to be weakly decreasing.

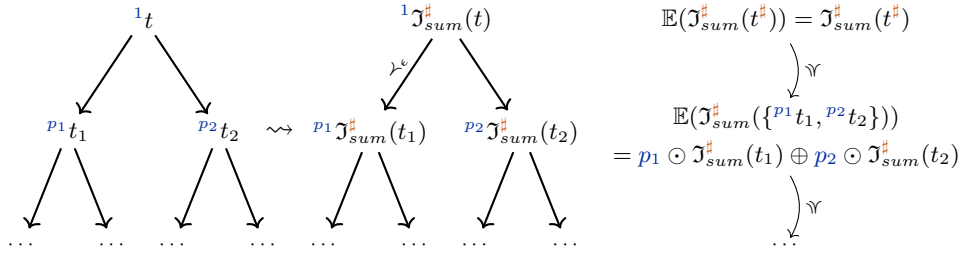



Figure 7.18: Each term $r$ in the chain tree (left) is mapped to the corresponding $\sharp$-sum $\mathfrak{I}^\sharp_{sum}(r)$ (middle). The expected $\sharp$-sum (right) is always weakly decreasing.

We can now prove the soundness (and completeness) of the actual reduction pair processor.

*Proof of Theorem 7.5.1.* We prove both directions individually. Let $\overline{\mathcal{A}} = \mathcal{A}_{\succcurlyeq} \cup \flat(\mathcal{A}_\succ)$. *Proof*

▶ *Proof(Completeness)*: By Theorem 7.3.1.

▶ *Proof(Soundness)*: Suppose that every (innermost) $\overline{\mathcal{A}}$-CT converges with probability 1. Assume for a contradiction that there exists an (innermost) $\mathcal{A}$-CT $\mathfrak{T} = (N, E)$ that converges with probability $< 1$. Let us partition $\Gamma = \Gamma_{\succcurlyeq} \uplus \Gamma_\succ$ according to the used ADPs for the rewrite steps. Since every $\overline{\mathcal{A}}$-CT converges with probability 1, so does every subtree of $\mathfrak{T}$ that only contains $\Gamma$-nodes from $\Gamma_{\succcurlyeq}$. By the $\Gamma$-Partition Lemma (Lemma 7.3.2), there must be an (innermost) $\mathcal{A}$-CT $\mathfrak{T}'$ with $|\mathfrak{T}'| < 1$ such that every infinite path has an infinite number of nodes from $\Gamma_\succ$. But this is a contradiction to Lemma 7.5.2. *QED*

Note that Requirement (2) only ensures a strict decrease for rewrite steps of type **at** or **af**. This is the reason why we require an infinite number of such steps within each infinite path of a chain tree, see Definition 7.2.12. Moreover, collapsibility implies that every strictly decreasing $\succ_\epsilon$-chain is finite: each $\succ_\epsilon$-step decreases the embedding $\mathrm{emb} \geq 0$ by at least $\epsilon$, so the length of the chain is bounded by $\mathrm{emb}(a_0)/\epsilon$ for its first element $a_0$. This is needed in order to show that the properties of the reduction pair processor are indeed sufficient for `AST`.

As for the direct application of algebras in Section 3.4.5, to search for a suitable barycentric $\Sigma^\sharp$-algebra fully automatically, we instantiate Theorem 7.5.1 with

polynomial interpretations. Just as in Corollary 3.4.39, every natural, weakly monotonic, and multilinear polynomial interpretation Pol is a weakly monotonic collapsible barycentric $\Sigma^\sharp$-algebra: its carrier $\mathbb{R}_{\geq 0}$ forms an $\mathbb{R}_{\geq 0}$-semimodule with the standard addition and scalar multiplication (see Examples 3.4.27 and 3.4.29), multilinearity ensures concavity of all $f_{\mathrm{Pol}}$, and collapsibility holds via the monotone, concave (in fact affine) identity embedding $\mathrm{emb} = id : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$, for which the $\epsilon$-decrease $\succ_\epsilon$ is the standard gap order $>_\epsilon$, lifted over instantiations from $\mathbb{R}_{\geq 0}$ (as for the corollaries in Section 3.4.5). Moreover, $\mathrm{emb} = id$ is $\epsilon$-separated on interpretations of terms for every $\epsilon \leq 1$. In [KG24, KG26], this processor was introduced for barycentric $\Sigma^\sharp$-algebras whose embedding maps into $\mathbb{N}$: there, integrality was intended to yield both the uniform gap and the $\epsilon$-separation for free, so no explicit $\epsilon$ is needed. However, as formulated there, a concave $\mathbb{N}$-valued embedding on a carrier like $\mathbb{R}_{\geq 0}$ is essentially constant, and the embedding condition was only required for single terms, while the proof uses it for $\sharp$-sums. Our real-valued formulation repairs this, and it covers the intended instances (e.g., natural polynomial interpretations, whose terms have natural values) with $\epsilon = 1$. Hence, the semimodule operations $\oplus$ and $\odot$ become the standard addition $+$ and multiplication $\cdot$, the order $\succcurlyeq$ becomes $\geq$ (read over instantiations from $\mathbb{R}_{\geq 0}$), and the expected value $\mathbb{E}$ becomes the standard expected value.

Corollary 7.5.3 – Reduction Pair Processor via Polynomial Interpretations.

*Let $\mathcal{A}$ be an ADP problem, let $\mathrm{Pol} : \Sigma^\sharp \to \mathbb{N}[\mathcal{V}]$ be a weakly monotonic, multilinear polynomial interpretation, let $\epsilon > 0$, and let $\mathrm{Pol}^\sharp_{sum}(r) = \sum_{t \trianglelefteq_\sharp r} \mathrm{Pol}(t^\sharp)$, and let $\mathcal{A} = \mathcal{A}_{\succcurlyeq} \uplus \mathcal{A}_{\succ}$ with $\mathcal{A}_{\succ} \neq \varnothing$ where:*

| | | | |
|---|---|---|---|
| (1) | | $\forall \ell \xhookrightarrow{m} \{{}^{p_1}r_1, \dots, {}^{p_k}r_k\} \in \mathcal{A}$. | $\mathrm{Pol}^\sharp_{sum}(\ell^\sharp) \geq \sum_{1 \leq j \leq k} p_j \cdot \mathrm{Pol}^\sharp_{sum}(r_j)$ |
| (2) | (a) | $\forall \ell \xhookrightarrow{m} \mu \in \mathcal{A}_{\succ} \exists r \in \mathrm{Supp}(\mu)$. | $\mathrm{Pol}^\sharp_{sum}(\ell^\sharp) >_\epsilon \mathrm{Pol}^\sharp_{sum}(r)$ |
| | (b) | *If $m = \mathtt{t}$, then we additionally have* | $\mathrm{Pol}(\ell) \geq \mathrm{Pol}(\flat(r))$ |
| (3) | | $\forall \ell \xhookrightarrow{\mathtt{t}} \{{}^{p_1}r_1, \dots, {}^{p_k}r_k\} \in \mathcal{A}$. | $\mathrm{Pol}(\ell) \geq \sum_{1 \leq j \leq k} p_j \cdot \mathrm{Pol}(\flat(r_j))$. |

*Then $\mathrm{Proc}_{\mathtt{RP}}(\mathcal{A}) = \mathrm{Proc}^{\mathtt{i}}_{\mathtt{RP}}(\mathcal{A}) = \{\mathcal{A}_{\succcurlyeq} \cup \flat(\mathcal{A}_{\succ})\}$ is sound and complete for $\mathtt{AST}$ and $\mathtt{iAST}$.*

*Proof.* This is an immediate consequence of Theorem 7.5.1. $(\mathrm{Pol}, >)$ is a weakly monotonic, collapsible barycentric $\Sigma^\sharp$-algebra whose embedding $\mathrm{emb} = id$ is $\epsilon'$-separated on interpretations of terms for $\epsilon' = \min(\epsilon, 1)$, since all terms have natural values. As $>_\epsilon$ implies $>_{\epsilon'}$, we can apply Theorem 7.5.1 with $\epsilon'$, where $\mathfrak{I}^\sharp_{sum}$ becomes $\mathrm{Pol}^\sharp_{sum}$. *QED*

The same holds for matrix interpretations as in [EWZ06].

We demonstrate the power of the reduction pair processor in the probabilistic setting using our running examples.

Example 7.5.4. To conclude $\mathtt{AST}(\xrightarrow{\mathtt{i}}_{\mathcal{P}_{\mathsf{dup}}})$ we have to remove all remaining annotations in the ADP problem $\{(\sharp 7.12''), (\sharp 7.14')\}$ with

$$\mathsf{g} \xhookrightarrow{\mathtt{f}} \{{}^{3/4}\mathsf{d}(\mathsf{g}^\sharp),\ {}^{1/4}\mathsf{0}\} \qquad (\sharp 7.12'') \qquad\qquad \mathsf{d}(x) \xhookrightarrow{\mathtt{f}} \{{}^{1}\mathsf{c}(x, x)\} \qquad (\sharp 7.14')$$

from Example 7.4.15. Here, we can use the reduction pair processor with the polynomial interpretation $(\mathfrak{I}, \succ) = (\mathrm{Pol}, >)$ that maps $\mathsf{g}^\sharp_{\mathrm{Pol}} = 1$, and $f_{\mathrm{Pol}} = 0$ for all other symbols $f$. Then $(\sharp 7.14')$ is weakly decreasing, and $(\sharp 7.12'')$ is strictly decreasing, since (1) $\mathrm{Pol}^\sharp_{sum}(\mathsf{g}^\sharp) = \mathrm{Pol}(\mathsf{g}^\sharp) = 1 \geq {}^3/_4 \cdot \mathrm{Pol}^\sharp_{sum}(\mathsf{d}(\mathsf{g}^\sharp)) + {}^1/_4 \cdot \mathrm{Pol}^\sharp_{sum}(\mathsf{0}) = {}^3/_4 \cdot \mathrm{Pol}(\mathsf{g}^\sharp) = {}^3/_4$

and (2a) $\mathrm{Pol}^\sharp_{sum}(\mathsf{g}^\sharp) = \mathrm{Pol}(\mathsf{g}^\sharp) = 1 > \mathrm{Pol}^\sharp_{sum}(0) = 0$, so Constraint (2a) holds with $\epsilon = 1$. Thus, the annotation of $\mathsf{g}^\sharp$ in ($\sharp$7.12″) is deleted. Then another application of the dependency graph processor yields the empty set of ADP problems.

Note that this polynomial interpretation would also satisfy the constraints for $\mathcal{DP}(\mathcal{P}_{\mathsf{dup}}) = \{(\sharp 7.12), (\sharp 7.14)\}$ from Example 7.2.6, i.e., it would allow us to remove the annotations from the canonical ADP directly. Hence, if we extended our approach for AST to GVRFs that can duplicate annotations, then the reduction pair processor would be unsound, as it would allow us to falsely "prove" AST of $\mathcal{DP}(\mathcal{P}_{\mathsf{dup}})$. The problem is that we compare terms with annotations via their $\sharp$-sum, but for duplicating ADPs like ($\sharp$7.14), $\mathrm{Pol}(\mathsf{d}(x)) \geq \mathrm{Pol}(\mathsf{c}(x,x))$ does not imply $\mathrm{Pol}^\sharp_{sum}(\mathsf{d}(\mathsf{g}^\sharp)) \geq \mathrm{Pol}^\sharp_{sum}(\mathsf{c}(\mathsf{g}^\sharp,\mathsf{g}^\sharp))$ since $\mathrm{Pol}^\sharp_{sum}(\mathsf{d}(\mathsf{g}^\sharp)) = \mathrm{Pol}(\mathsf{g}^\sharp)$ and $\mathrm{Pol}^\sharp_{sum}(\mathsf{c}(\mathsf{g}^\sharp,\mathsf{g}^\sharp)) = \mathrm{Pol}(\mathsf{g}^\sharp) + \mathrm{Pol}(\mathsf{g}^\sharp)$.

Example 7.5.5. To prove $\mathtt{AST}(\to_{\mathcal{P}_{\mathsf{alg}}})$, we also have to remove all annotations from all remaining sub-problems. For instance, for the sub-problem $\mathcal{A}_{(\sharp 7.5'),(\sharp 7.6')} = \{(\sharp 7.5'), (\sharp 7.6'), \flat(\sharp 7.7)\text{–}\flat(\sharp 7.11)\}$ from Example 7.4.10 with

$$\mathsf{loop1}(y) \overset{\mathsf{t}}{\hookrightarrow} \{^{1/2}\mathsf{loop1}^\sharp(\mathsf{double}(y)),\ ^{1/2}\mathsf{loop2}(\mathsf{double}(y))\} \qquad (\sharp 7.5')$$

$$\mathsf{loop1}(y) \overset{\mathsf{t}}{\hookrightarrow} \{^{1/3}\mathsf{loop1}^\sharp(\mathsf{triple}(y)),\ ^{2/3}\mathsf{loop2}(\mathsf{triple}(y))\} \qquad (\sharp 7.6')$$

$$\mathsf{loop2}(\mathsf{s}(y)) \overset{\mathsf{t}}{\hookrightarrow} \{^{1}\mathsf{loop2}(y)\} \quad (\flat(\sharp 7.7)) \qquad \mathsf{triple}(\mathsf{s}(y)) \overset{\mathsf{t}}{\hookrightarrow} \{^{1}\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{triple}(y))))\} \quad (\flat(\sharp 7.10))$$

$$\mathsf{double}(\mathsf{s}(y)) \overset{\mathsf{t}}{\hookrightarrow} \{^{1}\mathsf{s}(\mathsf{s}(\mathsf{double}(y)))\} \quad (\flat(\sharp 7.8)) \qquad \mathsf{triple}(0) \overset{\mathsf{t}}{\hookrightarrow} \{^{1}0\} \quad (\flat(\sharp 7.11))$$

$$\mathsf{double}(0) \overset{\mathsf{t}}{\hookrightarrow} \{^{1}0\} \quad (\flat(\sharp 7.9))$$

we can use the reduction pair processor with the polynomial interpretation $(\mathfrak{I}, \succ) = (\mathrm{Pol}, >)$ that maps $\mathsf{s}_{\mathrm{Pol}}(x) = x+1$, $\mathsf{double}_{\mathrm{Pol}}(x) = 2x$, $\mathsf{triple}_{\mathrm{Pol}}(x) = 3x$, $\mathsf{loop1}^\sharp_{\mathrm{Pol}}(x) = 1$, and $f_{\mathrm{Pol}} = 0$ for all other symbols $f$. Then ($\sharp$7.6′) is strictly decreasing, since (1) $\mathrm{Pol}^\sharp_{sum}(\mathsf{loop1}^\sharp(y)) = \mathrm{Pol}(\mathsf{loop1}^\sharp(y)) = 1 \geq 1/3 \cdot \mathrm{Pol}^\sharp_{sum}(\mathsf{loop1}^\sharp(\mathsf{triple}(y))) + 2/3 \cdot \mathrm{Pol}^\sharp_{sum}(\mathsf{loop2}(\mathsf{triple}(y))) = 1/3$ and (2a) $\mathrm{Pol}^\sharp_{sum}(\mathsf{loop1}^\sharp(y)) = \mathrm{Pol}(\mathsf{loop1}^\sharp(y)) = 1 > \mathrm{Pol}^\sharp_{sum}(\mathsf{loop2}(\mathsf{triple}(y))) = 0$ (so Constraint (2a) holds with $\epsilon = 1$). Similarly, ($\sharp$7.5′) is also strictly decreasing, and we can remove all annotations from this ADP problem. One can find similar interpretations to delete the remaining annotations also from the other remaining sub-problems. This proves $\mathtt{AST}(\hookrightarrow_{\mathcal{DP}(\mathcal{P}_{\mathsf{alg}})})$, and hence $\mathtt{AST}(\to_{\mathcal{P}_{\mathsf{alg}}})$.

If we only want to prove $\mathtt{AST}(\overset{\mathsf{i}}{\to}_{\mathcal{P}_{\mathsf{alg}}})$, then the application of the reduction pair processor becomes significantly easier. Except for $\flat(\sharp 7.8)\text{–}\flat(\sharp 7.11)$ in the ADP problem $\mathcal{A}_{(\sharp 7.5'),(\sharp 7.6')}$ above, all other ADPs have the flag f after the usable rules processor, see Example 7.4.16. Therefore, (2b) from Theorem 7.5.1 does not apply, and (3) only has to hold for $\flat(\sharp 7.8)\text{–}\flat(\sharp 7.11)$. Thus, we obtain significantly fewer constraints, which makes the search for interpretations easier for the SMT solver.

Example 7.5.6. We now prove $\mathtt{AST}(\overset{\mathsf{i}}{\to}_{\mathcal{P}_{\mathsf{div}}})$ for the PTRS $\mathcal{P}_{\mathsf{div}}$ from Counterexample 7.0.1 with the ADP framework, following the proof tree of Figure 7.9. Note that the rule (7.4) duplicates the variable $y$, i.e., $\mathcal{P}_{\mathsf{div}}$ is not ND, and hence, the chain criterion for full rewriting (Theorem 7.2.19) is not applicable. By the chain criterion for innermost rewriting (Theorem 7.2.16), it suffices to prove $\mathtt{AST}(\overset{\mathsf{i}}{\hookrightarrow}_{\mathcal{DP}(\mathcal{P}_{\mathsf{div}})})$ for the canonical ADP problem. The defined symbols of $\mathcal{P}_{\mathsf{div}}$ are minus and div, so by Definition 7.2.5 we annotate every subterm with root minus or div in the right-hand sides, which yields $\mathcal{DP}(\mathcal{P}_{\mathsf{div}}) = \{(\sharp 7.1), (\sharp 7.2), (\sharp 7.3), (\sharp 7.4)\}$:

$$\mathsf{minus}(x, 0) \overset{\mathsf{t}}{\hookrightarrow} \{^{1/2}\mathsf{minus}^\sharp(x, 0),\ ^{1/2}x\} \qquad (\sharp 7.1)$$

$$\mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)) \overset{\mathsf{t}}{\hookrightarrow} \{^{1/2}\mathsf{minus}^\sharp(\mathsf{s}(x), \mathsf{s}(y)),\ ^{1/2}\mathsf{minus}^\sharp(x, y)\} \qquad (\sharp 7.2)$$

$$\mathsf{div}(0,\mathsf{s}(y)) \overset{\mathsf{t}}{\hookrightarrow} \{{}^{1/2}\mathsf{div}^\sharp(0,\mathsf{s}(y)),\ {}^{1/2}0\} \tag{♯7.3}$$

$$\mathsf{div}(\mathsf{s}(x),\mathsf{s}(y)) \overset{\mathsf{t}}{\hookrightarrow} \{{}^{1/2}\mathsf{div}^\sharp(\mathsf{s}(x),\mathsf{s}(y)),\ {}^{1/2}\mathsf{s}(\mathsf{div}^\sharp(\mathsf{minus}^\sharp(x,y),\mathsf{s}(y)))\} \tag{♯7.4}$$

*Dependency Graph Processor*: The innermost $\mathcal{DP}(\mathcal{P}_{\mathsf{div}})$-dependency graph (Definition 7.4.5) is shown in Figure 7.19. Here, every ADP has an edge to itself. Furthermore, (♯7.2) and (♯7.4) have edges to both (♯7.1) and (♯7.2) via their annotated subterm $\mathsf{minus}^\sharp(x,y)$, and (♯7.4) has an edge to (♯7.3) via its annotated subterm $\mathsf{div}^\sharp(\mathsf{minus}^\sharp(x,y),\mathsf{s}(y))$, whose first argument $\mathsf{minus}(x,y)$ can be rewritten to $\mathsf{0}$. There are no further edges, and hence, the only cycles are the four self-loops, and the SCCs are the singletons $\{(♯7.1)\}$, $\{(♯7.2)\}$, $\{(♯7.3)\}$, and $\{(♯7.4)\}$. By Theorem 7.4.6, $\mathrm{Proc}^{\mathtt{i}}_{\mathtt{DG}}$ thus returns the four ADP problems

$$\mathcal{A}_i = \{i\} \cup \flat\big(\mathcal{DP}(\mathcal{P}_{\mathsf{div}}) \setminus \{i\}\big) \qquad \text{for } i \in \{(♯7.1),(♯7.2),(♯7.3),(♯7.4)\},$$

i.e., in each $\mathcal{A}_i$ only the $i$-th ADP keeps its annotations.

*Reduction Pair Processor*: We remove the remaining annotations from each $\mathcal{A}_i$ with Corollary 7.5.3, where in every case the single annotated ADP is the strictly decreasing one ($\mathcal{A}_\succ = \{i\}$) of the reduction pair processor. All four flattened rules are weakly decreasing in expectation under the base interpretation $\mathsf{0}_{\mathrm{Pol}} = 0$, $\mathsf{s}_{\mathrm{Pol}}(x) = x+1$, $\mathsf{minus}_{\mathrm{Pol}}(x,y) = x$, and $\mathsf{div}_{\mathrm{Pol}}(x,y) = x$, so Constraint (3) is always satisfied, and Constraint (2b) holds for the respective branches as well. For instance, for (♯7.4) we have $\mathrm{Pol}(\mathsf{div}(\mathsf{s}(x),\mathsf{s}(y))) = x+1 \geq {}^1\!/_2 \cdot (x+1) + {}^1\!/_2 \cdot \mathrm{Pol}(\mathsf{s}(\mathsf{div}(\mathsf{minus}(x,y),\mathsf{s}(y)))) = {}^1\!/_2 \cdot (x+1) + {}^1\!/_2 \cdot (x+1) = x+1$, using $\mathrm{Pol}(\mathsf{s}(\mathsf{div}(\mathsf{minus}(x,y),\mathsf{s}(y)))) = \mathrm{Pol}(\mathsf{minus}(x,y)) + 1 = x+1$. It remains to choose the interpretations of the annotated symbols.

For the two $\mathsf{minus}$-ADPs $\mathcal{A}_{(♯7.1)}$ and $\mathcal{A}_{(♯7.2)}$, we additionally set $\mathsf{minus}^\sharp_{\mathrm{Pol}}(x,y) = x+1$ (and $\mathsf{div}^\sharp_{\mathrm{Pol}}(x,y) = 0$). Then (♯7.1) is strictly decreasing in $\mathcal{A}_{(♯7.1)}$, since (1) $\mathrm{Pol}^\sharp_{sum}(\mathsf{minus}^\sharp(x,0)) = x+1 \geq {}^1\!/_2 \cdot \mathrm{Pol}^\sharp_{sum}(\mathsf{minus}^\sharp(x,0)) + {}^1\!/_2 \cdot \mathrm{Pol}^\sharp_{sum}(x) = \frac{x+1}{2}$ and (2a) $\mathrm{Pol}^\sharp_{sum}(\mathsf{minus}^\sharp(x,0)) = x+1 > 0 = \mathrm{Pol}^\sharp_{sum}(x)$. Similarly, (♯7.2) is strictly decreasing in $\mathcal{A}_{(♯7.2)}$, since (1) $\mathrm{Pol}^\sharp_{sum}(\mathsf{minus}^\sharp(\mathsf{s}(x),\mathsf{s}(y))) = x+2 \geq {}^1\!/_2 \cdot (x+2) + {}^1\!/_2 \cdot (x+1) = x + \frac{3}{2}$ and (2a) $\mathrm{Pol}^\sharp_{sum}(\mathsf{minus}^\sharp(\mathsf{s}(x),\mathsf{s}(y))) = x+2 > x+1 = \mathrm{Pol}^\sharp_{sum}(\mathsf{minus}^\sharp(x,y))$.

For the two $\mathsf{div}$-ADPs $\mathcal{A}_{(♯7.3)}$ and $\mathcal{A}_{(♯7.4)}$, we instead set $\mathsf{div}^\sharp_{\mathrm{Pol}}(x,y) = x+1$ and $\mathsf{minus}^\sharp_{\mathrm{Pol}}(x,y) = 0$. Then (♯7.3) is strictly decreasing in $\mathcal{A}_{(♯7.3)}$ as in the case of (♯7.1), since (1) $\mathrm{Pol}^\sharp_{sum}(\mathsf{div}^\sharp(0,\mathsf{s}(y))) = 1 \geq {}^1\!/_2 \cdot 1 + {}^1\!/_2 \cdot 0 = \frac{1}{2}$ and (2a) $\mathrm{Pol}^\sharp_{sum}(\mathsf{div}^\sharp(0,\mathsf{s}(y))) = 1 > 0 = \mathrm{Pol}^\sharp_{sum}(\mathsf{0})$. Finally, (♯7.4) is strictly decreasing w.r.t. $\mathcal{A}_{(♯7.4)}$. The $\sharp$-sum of its second right-hand side $\mathsf{s}(\mathsf{div}^\sharp(\mathsf{minus}^\sharp(x,y),\mathsf{s}(y)))$ has the two summands $\mathrm{Pol}(\mathsf{div}^\sharp(\mathsf{minus}(x,y),\mathsf{s}(y))) = \mathrm{Pol}(\mathsf{minus}(x,y)) + 1 = x+1$ and $\mathrm{Pol}(\mathsf{minus}^\sharp(x,y)) = 0$, so that $\mathrm{Pol}^\sharp_{sum}(\mathsf{s}(\mathsf{div}^\sharp(\mathsf{minus}^\sharp(x,y),\mathsf{s}(y)))) = x+1$. Hence (1) $\mathrm{Pol}^\sharp_{sum}(\mathsf{div}^\sharp(\mathsf{s}(x),\mathsf{s}(y))) = x+2 \geq {}^1\!/_2 \cdot (x+2) + {}^1\!/_2 \cdot (x+1) = x + \frac{3}{2}$ and (2a) $\mathrm{Pol}^\sharp_{sum}(\mathsf{div}^\sharp(\mathsf{s}(x),\mathsf{s}(y))) = x+2 > x+1 = \mathrm{Pol}^\sharp_{sum}(\mathsf{s}(\mathsf{div}^\sharp(\mathsf{minus}^\sharp(x,y),\mathsf{s}(y))))$. All strict decreases above have a uniform gap of at least 1, so Constraint (2a) holds with $\epsilon = 1$ in each case. Here, the strict decrease stems from the first argument of $\mathsf{div}^\sharp$ dropping from $\mathsf{s}(x)$ to $\mathsf{minus}(x,y)$, i.e., from $x+1$ to $x$.

In each of the four ADP problems, the reduction pair processor removes the annotations from the single annotated ADP, leaving only ADP problems without any annotations, which are trivially $\mathtt{iAST}$. Thus, by the soundness of the processors and the chain criterion, we obtain $\mathtt{AST}(\overset{\mathtt{i}}{\to}_{\mathcal{P}_{\mathsf{div}}})$.

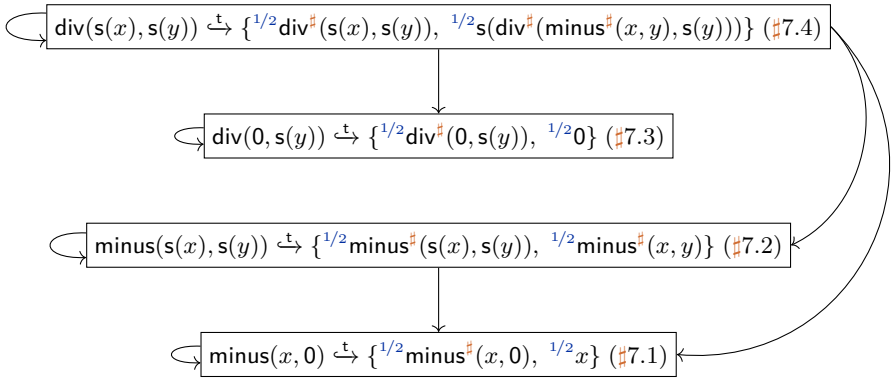



Figure 7.19: Innermost $\mathcal{DP}(\mathcal{P}_{\mathsf{div}})$-dependency graph.

Note that the reduction pair processor (together with the chain criterion) does not subsume the direct application of polynomial interpretations as described in Corollary 3.4.39. The reason is that the chain criterion for AST (Theorem 7.2.19) is only applicable to ND PTRSs, because a duplicating rewrite step would have to copy the annotations of the duplicated subterms, which our VRFs cannot express (see the discussion in Section 7.2.3).

### 7.5.2 Subterm Criterion

Next, we consider the *subterm criterion* of [HM04, HM07] which is a purely syntactical way of removing loops from the dependency graph. In the non-probabilistic setting, the subterm criterion searches for a *simple projection* for every annotated symbol $f^\sharp$, i.e., a projection proj which maps $f^\sharp$ to one of its argument positions $i$ and replaces every term $f^\sharp(\dots)$ by its $i$-th argument. If the projection of the right-hand side $\mathrm{proj}(r^\sharp)$ is a subterm of the projected left-hand side $\mathrm{proj}(\ell^\sharp)$ for every dependency pair $\ell^\sharp \to r^\sharp \in \mathcal{D}$, then we can remove those dependency pairs from $\mathcal{D}$ where $\mathrm{proj}(r^\sharp)$ is a proper subterm of $\mathrm{proj}(\ell^\sharp)$.[10]

Definition 7.5.7 – Simple Projection.

*A simple projection is a mapping* proj *that assigns to every $n$-ary annotated symbol $f^\sharp \in \mathcal{D}^\sharp$ with $n \geq 1$ an argument position $i \in \{1, \dots, n\}$. We extend* proj *to terms with annotated root symbol as follows: For $t_1, \dots, t_n \in \mathcal{T}$ with $n \geq 1$ and $\mathrm{proj}(f^\sharp) = i$, we define $\mathrm{proj}(f^\sharp(t_1, \dots, t_n)) = t_i$. If $f^\sharp$ has arity 0, then we define $\mathrm{proj}(f^\sharp) = f^\sharp$.*

**Subterm Criterion Processor for AST** In the non-probabilistic setting, if $\mathrm{proj}(\ell^\sharp) \trianglerighteq \mathrm{proj}(r^\sharp)$ holds for all $\ell^\sharp \to r^\sharp$ in a set of DPs $\mathcal{D}$, then one can remove all DPs from $\mathcal{D}$ where we have $\mathrm{proj}(\ell^\sharp) \triangleright \mathrm{proj}(r^\sharp)$. So the DP processor which transforms $(\mathcal{D}, \mathcal{R})$ into $(\mathcal{D} \setminus \mathcal{D}_\triangleright, \mathcal{R})$ is sound and complete for termination, where $\mathcal{D}_\triangleright = \{\ell^\sharp \to r^\sharp \in \mathcal{D} \mid \mathrm{proj}(\ell^\sharp) \triangleright \mathrm{proj}(r^\sharp)\}$. The advantage is that in contrast to the reduction pair processor, the criterion of this processor is purely syntactical (i.e., one does not have to search for interpretations or well-founded relations), and one can

[10] Projections are often denoted by $\pi$ in the literature, but we use $\pi$ for positions.

remove DPs without considering the rewrite rules $\mathcal{R}$ at all. However, the soundness of this processor for full rewriting relies on the minimality property, which does not hold in the probabilistic setting, see Counterexample 5.3.1. Thus, such a processor would be unsound for $\mathtt{AST}$ as shown by the following counterexample.

Counterexample 7.5.8. The ADP problem with $\mathsf{f}(\mathsf{s}(x)) \overset{\mathsf{t}}{\hookrightarrow} \{^1\mathsf{f}^\sharp(x)\}$ and $\mathsf{g} \overset{\mathsf{t}}{\hookrightarrow} \{^1\mathsf{s}(\mathsf{g})\}$ is not $\mathtt{AST}$ as we can rewrite $\mathsf{f}^\sharp(\mathsf{s}(\mathsf{g}))$ to itself by first applying the $\mathsf{f}$-ADP in an **at**-step, followed by an **nt**-step with the $\mathsf{g}$-ADP. However, consider the simple projection proj which maps each annotated symbol to its only argument. Then, the projected right-hand side $\mathrm{proj}(\mathsf{f}^\sharp(x)) = x$ of the $\mathsf{f}$-ADP is a proper subterm of the projected annotated left-hand side $\mathrm{proj}(\mathsf{f}^\sharp(\mathsf{s}(x))) = \mathsf{s}(x)$, while the $\mathsf{g}$-ADP does not contain any annotations, so we can ignore it. Hence, we would falsely assume that we can remove the annotation of the $\mathsf{f}$-ADP, leading to an ADP problem which is $\mathtt{AST}$ as it does not contain any annotations anymore. The problem here is the non-terminating $\mathsf{g}$-rule that can generate infinitely many $\mathsf{s}$-symbols.

**Subterm Criterion Processor for $\mathtt{iAST}$** For non-probabilistic innermost rewriting, the soundness of the subterm criterion does not rely on the minimality property. Thus, we now show how to adapt the subterm criterion to the probabilistic setting in order to prove $\mathtt{AST}(\overset{\mathsf{i}}{\hookrightarrow}_{\mathcal{A}})$. For innermost rewriting, Counterexample 7.5.8 is no longer a counterexample, because the ADP problem from Counterexample 7.5.8 satisfies $\mathtt{AST}(\overset{\mathsf{i}}{\hookrightarrow}_{\mathcal{A}})$. The reason is that terms without the symbol $\mathsf{g}$ only have finite reductions, and if the start term contains the symbol $\mathsf{g}$, then we have to rewrite it due to the innermost evaluation strategy and we cannot rewrite any $\mathsf{f}^\sharp$-symbols on positions above $\mathsf{g}$.

However, there is another problem when adapting the subterm criterion to the probabilistic setting. Since we consider ADPs instead of ordinary DPs, we may have terms with several annotations in the right-hand side. For the reduction pair processor we solved this problem by considering the $\sharp$-sum. However, for the subterm criterion it is not clear how to combine several annotated subterms. Therefore, we restrict ourselves to ADP problems where all terms in the right-hand sides contain at most one annotation. This leads to the following processor.

Theorem 7.5.9 – Subterm Criterion for $\mathtt{iAST}$.
Let $\mathcal{A}$ be an ADP problem where all terms in the right-hand sides contain at most one annotation. Let proj be a simple projection such that for all $\ell \overset{m}{\hookrightarrow} \mu \in \mathcal{A}$, all $r \in \mathrm{Supp}(\mu)$, and all $t \trianglelefteq_\sharp r$, we have $\mathrm{proj}(\ell^\sharp) \trianglerighteq \mathrm{proj}(t^\sharp)$. Moreover, let $\mathcal{A}_\triangleright \subseteq \mathcal{A}$ be the set of all ADPs $\ell \overset{m}{\hookrightarrow} \mu \in \mathcal{A}$ for which there exist an $r \in \mathrm{Supp}(\mu)$ and a term $t \trianglelefteq_\sharp r$ such that $\mathrm{proj}(\ell^\sharp) \triangleright \mathrm{proj}(t^\sharp)$. Then the processor $\mathrm{Proc}^{\mathsf{i}}_{\mathtt{SC}}(\mathcal{A}) = \{(\mathcal{A} \setminus \mathcal{A}_\triangleright) \cup \flat(\mathcal{A}_\triangleright)\}$ is sound and complete for $\mathtt{iAST}$.

*Proof.* We prove both directions individually.

▶ *Proof*(*Completeness*): By Theorem 7.3.1.

▶ *Proof*(*Soundness*): Consider the polynomial interpretation with $f_{\mathrm{Pol}}(x_1, \ldots, x_n) = x_1 + \ldots + x_n + 1$ for each $f \in \Sigma$, $f^\sharp_{\mathrm{Pol}}(x_1, \ldots, x_n) = x_{\mathrm{proj}(f^\sharp)}$ for each $f^\sharp \in \mathcal{D}^\sharp$ of arity $n \geq 1$, and $f^\sharp_{\mathrm{Pol}} = 1$ for each annotated constant $f^\sharp$, i.e., we map each annotated symbol to its projected argument by the simple projection proj, and all other symbols are used to calculate the number of symbols in a term.
When choosing $\mathcal{A}_\succ$ to be $\mathcal{A}_\triangleright$, then the conditions of the reduction pair processor that

*Proof*

consider the $\sharp$-sums (Cases (1) and (2a)) are both satisfied with this interpretation. On the other hand, the conditions on the flattened ADPs without annotations (Case (2b) and (3)) may not be satisfied. However, due to the restriction to innermost rewriting and the restriction that ADPs only contain at most a single annotated symbol in each term on the right-hand side, those cases are irrelevant, because of the following observation.

By Lemma 7.3.3, it suffices to consider innermost $\mathcal{A}$-CTs $\mathfrak{T}$ that start with a term $t$ with $\mathrm{pos}_{\mathcal{D}^\sharp}(t) = \{\varepsilon\}$ and $\flat(t) \in \mathtt{ANF}_{\mathcal{A}}$. For such a CT, we show that every node $v$ satisfies the following invariant: for every annotated position $\tau \in \mathrm{pos}_{\mathcal{D}^\sharp}(t_v)$ whose symbol is not a constant, the projected argument $\mathrm{proj}(\flat(t_v|_\tau)^\sharp)$ is in $\mathtt{NF}_{\mathcal{A}}$. For the root, this holds since $\mathrm{proj}(\flat(t)^\sharp)$ is a proper subterm of $\flat(t) \in \mathtt{ANF}_{\mathcal{A}}$. Now consider an innermost rewrite step at a node $v$ with an ADP $\ell \overset{m}{\hookrightarrow} \mu \in \mathcal{A}$ at a position $\pi$ with the substitution $\sigma$, where $\ell\sigma \in \mathtt{ANF}_{\mathcal{A}}$. By the invariant, $\pi$ is not inside the projected argument of any annotated position above $\pi$, so the projected arguments of all annotated positions above $\pi$ or orthogonal to $\pi$ are not changed, and all annotations below $\pi$ are removed by the innermost step. If the step is of the form **at** or **af**, let $r \in \mathrm{Supp}(\mu)$ and, if $r$ contains an annotated symbol, let $t \trianglelefteq_\sharp r$ be its annotated subterm. Note that there exists at most one such $t$ for each $r$, as there is at most one annotation in $r$. Since $\mathrm{proj}(\ell^\sharp) \trianglerighteq \mathrm{proj}(t^\sharp)$ by definition of the subterm criterion, and $\mathrm{proj}(\ell^\sharp)\sigma$ is a proper subterm of $\ell\sigma \in \mathtt{ANF}_{\mathcal{A}}$, we have $\mathrm{proj}(t^\sharp)\sigma \in \mathtt{NF}_{\mathcal{A}}$, so the invariant holds again. Hence, we never rewrite inside the projected argument of an annotated symbol. This means that the $\sharp$-sum of the terms cannot increase by rewrite steps at non-annotated positions (they can only remove annotations). Thus, in this case the reduction pair processor works without requiring Case (2b) and (3), as these requirements are never used in its proof if the rules have no impact on the $\sharp$-sum of a term. *QED*

The idea of this processor is similar to the reduction pair processor. Since in all ADPs, the "value" (which is the term structure in this case) cannot increase, we can remove those ADPs where there is a chance to decrease the "value". But in contrast to the reduction pair processor, we do not have to impose any constraints resulting from disregarding the annotations of ADPs, i.e., we do not obtain constraints of the form (2b) or (3) in Theorem 7.5.1.

As noted in [Thi07], for non-probabilistic innermost rewriting, the subterm criterion is subsumed by the reduction pair processor when integrating projections (or "argument filterings") and an appropriate improved notion of usable rules into the reduction pair processor [GTSF06]. Such an improved reduction pair processor which subsumes the subterm criterion would most likely also be possible in the probabilistic setting, but the reduction pair processor of Theorem 7.5.1 does not subsume the subterm criterion.

Example 7.5.10 (Subterm Criterion vs. Reduction Pair Processor). Consider the ADP problem $\mathcal{A} = \{\mathsf{h}(\mathsf{s}(\mathsf{s}(x))) \overset{\mathrm{t}}{\hookrightarrow} \{^1\mathsf{h}^\sharp(\mathsf{s}(x))\},\ \mathsf{s}(\mathsf{a}) \overset{\mathrm{t}}{\hookrightarrow} \{^1\mathsf{s}(\mathsf{s}(\mathsf{a}))\}\}$. The subterm criterion with $\mathrm{proj}(\mathsf{h}^\sharp) = 1$ removes the annotation of the $\mathsf{h}$-ADP, since $\mathsf{s}(\mathsf{s}(x)) \rhd \mathsf{s}(x)$. However, there is no barycentric $\Sigma^\sharp$-algebra $(\mathfrak{J}, \succ)$ such that the reduction pair processor can remove this annotation: for $u = \mathsf{s}_\mathfrak{J}(\mathsf{a}_\mathfrak{J})$, Requirement (3) for the $\mathsf{s}$-ADP yields $u \succcurlyeq \mathsf{s}_\mathfrak{J}(u)$, hence $\mathsf{h}^\sharp_\mathfrak{J}(u) \succcurlyeq \mathsf{h}^\sharp_\mathfrak{J}(\mathsf{s}_\mathfrak{J}(u))$ by weak monotonicity. But instantiating $x$ by $\mathsf{a}_\mathfrak{J}$ in Requirement (2a) for the $\mathsf{h}$-ADP yields $\mathsf{h}^\sharp_\mathfrak{J}(\mathsf{s}_\mathfrak{J}(u)) \succ_\epsilon \mathsf{h}^\sharp_\mathfrak{J}(u)$, so that $\mathsf{h}^\sharp_\mathfrak{J}(u) \succ \mathsf{h}^\sharp_\mathfrak{J}(u)$, a contradiction. This also holds after applying the usable rules processor, since the $\mathsf{s}$-ADP is usable for $\mathsf{h}^\sharp(\mathsf{s}(x))$.

Example 7.5.11. Consider the ADP problem $\{(\sharp 7.7), \flat(\sharp 7.5), \flat(\sharp 7.6), \flat(\sharp 7.8)$–$\flat(\sharp 7.11)\}$ with

$$\mathsf{loop1}(y) \overset{\mathtt{t}}{\hookrightarrow} \{{}^{1/2}\mathsf{loop1}(\mathsf{double}(y)),\ {}^{1/2}\mathsf{loop2}(\mathsf{double}(y))\} \qquad (\flat(\sharp 7.5))$$

$$\mathsf{loop1}(y) \overset{\mathtt{t}}{\hookrightarrow} \{{}^{1/3}\mathsf{loop1}(\mathsf{triple}(y)),\ {}^{2/3}\mathsf{loop2}(\mathsf{triple}(y))\} \qquad (\flat(\sharp 7.6))$$

$$\mathsf{loop2}(\mathsf{s}(y)) \overset{\mathtt{t}}{\hookrightarrow} \{{}^{1}\mathsf{loop2}^{\sharp}(y)\} \qquad (\sharp 7.7)$$

$$\mathsf{double}(\mathsf{s}(y)) \overset{\mathtt{t}}{\hookrightarrow} \{{}^{1}\mathsf{s}(\mathsf{s}(\mathsf{double}(y)))\} \qquad (\flat(\sharp 7.8))$$

$$\mathsf{double}(0) \overset{\mathtt{t}}{\hookrightarrow} \{{}^{1}0\} \qquad (\flat(\sharp 7.9))$$

$$\mathsf{triple}(\mathsf{s}(y)) \overset{\mathtt{t}}{\hookrightarrow} \{{}^{1}\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{triple}(y))))\} \qquad (\flat(\sharp 7.10))$$

$$\mathsf{triple}(0) \overset{\mathtt{t}}{\hookrightarrow} \{{}^{1}0\} \qquad (\flat(\sharp 7.11))$$

that results from the innermost dependency graph processor when proving $\mathtt{AST}(\overset{\mathtt{i}}{\to}_{\mathcal{P}_{\mathsf{alg}}})$, see Examples 7.4.3 and 7.4.7. Here, we can use the simple projection proj that maps $\mathsf{loop2}^{\sharp}$ to its only argument. Then for the only ADP with annotations $(\sharp 7.7)$, the projection of the annotated subterm of the right-hand side $\mathsf{proj}(\mathsf{loop2}^{\sharp}(y)) = y$ is a proper subterm of the projected annotated left-hand side $\mathsf{proj}(\mathsf{loop2}^{\sharp}(\mathsf{s}(y))) = \mathsf{s}(y)$. Hence, by applying Theorem 7.5.9 we obtain $\{\flat(\sharp 7.5)$–$\flat(\sharp 7.11)\}$, which has no annotations anymore. Thus, we have directly shown that this ADP problem satisfies $\mathtt{AST}(\overset{\mathtt{i}}{\hookrightarrow}_{\mathcal{A}})$. So in contrast to the reduction pair processor of Theorem 7.5.1, here we can ignore the other ADPs $\flat(\sharp 7.5), \flat(\sharp 7.6), \flat(\sharp 7.8)$–$\flat(\sharp 7.11)$ completely even though their flags are $\mathtt{t}$.

> Remark 7.5.12
> While the subterm criterion itself is purely structural, its proof relies on polynomial interpretations in the probabilistic setting. For this reason, we decided to present the processor in this section instead of the previous one.

SECTION 7.6

## Transformational Processors

In the non-probabilistic DP framework, there exist several *transformational* processors that increase the power of the framework substantially [GTSF06]. To benefit from such transformations in the probabilistic setting as well, in Section 7.6.1–7.6.4, we introduce the *rewriting*, *instantiation*, *forward instantiation*, and *rule overlap instantiation* processors for our ADP framework. The latter corresponds to a weaker version of the *narrowing* processor from [GTSF06], which is unsound in the probabilistic setting in general, as we will see in Counterexample 7.6.18.

In this section, we introduce processors based on transforming ADPs into new ones that are easier to analyze. Once again, these processors are very fast to execute. However, these transformations may be applicable an infinite number of times, while they do not always help. Therefore, one needs strict thresholds and heuristics about when and how often to use such a transformational processor. In total, we consider four different processors.

1. The *rewriting processor* performs a rewrite step on a redex in a right-hand side of an ADP already at the level of the ADP problem, anticipating a step that the corresponding reduction would perform anyway, which requires non-overlapping usable rules (and only works for innermost rewriting).
2. The *instantiation processor* inspects all possible *predecessors* of an ADP in a chain and instantiates its left-hand side with the most general unifier of the

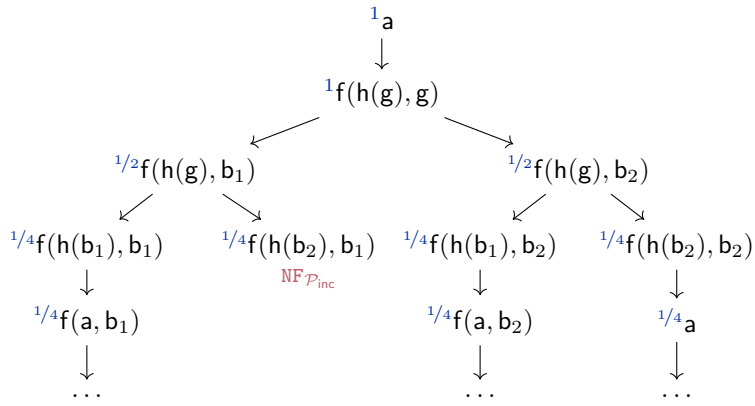



Figure 7.20: A $\to_{\mathcal{P}_{\mathsf{inc}}}$-RT that starts with $\mathsf{a}$.

skeleton that necessarily stays unchanged when rewriting from one ADP to the next, thereby making the ADP more specific.

3. The *forward instantiation processor* dually inspects all possible *successors* of an ADP in a chain (by reversing the rules) and instantiates the ADP accordingly, again restricting it to the instances that can actually occur.
4. The *rule overlap instantiation processor* is a weakened narrowing processor that only instantiates an ADP by the substitutions enabling the next rewrite step at or below an annotation, but does not perform that step itself, as doing so would be unsound in the probabilistic setting.

### 7.6.1 Rewriting Processor

We start by considering the *rewriting* processor. To illustrate the idea of this processor, we use the following example.

Example 7.6.1. Consider the PTRS $\mathcal{P}_{\mathsf{inc}}$ with the rules

$$\mathsf{a} \to \{{}^{1}\mathsf{f}(\mathsf{h}(\mathsf{g}),\mathsf{g})\} \qquad (7.45) \qquad\qquad \mathsf{h}(\mathsf{b}_1) \to \{{}^{1}\mathsf{a}\} \qquad (7.47)$$

$$\mathsf{g} \to \{{}^{1/2}\mathsf{b}_1, {}^{1/2}\mathsf{b}_2\} \qquad (7.46) \qquad\qquad \mathsf{f}(x,\mathsf{b}_2) \to \{{}^{1}\mathsf{a}\} \qquad (7.48)$$

and the $\to_{\mathcal{P}_{\mathsf{inc}}}$-RT in Figure 7.20. The term $\mathsf{a}$ can be rewritten to the normal form $\mathsf{f}(\mathsf{h}(\mathsf{b}_2),\mathsf{b}_1)$ with probability $1/4$ and to the terms $\mathsf{f}(\mathsf{a},\mathsf{b}_1)$, $\mathsf{f}(\mathsf{a},\mathsf{b}_2)$, and $\mathsf{a}$ that contain the redex $\mathsf{a}$ with probability $1/4 + 1/4 + 1/4 = 3/4$. In the term $\mathsf{f}(\mathsf{a},\mathsf{b}_2)$, one can rewrite the subterm $\mathsf{a}$, and if that ends in a normal form, one can still rewrite the outer $\mathsf{f}$ which will yield $\mathsf{a}$ again. So to over-approximate the probability of non-termination, one could consider the term $\mathsf{f}(\mathsf{a},\mathsf{b}_2)$ as if one had two occurrences of $\mathsf{a}$. Then this would correspond to a symmetric random walk where the number of $\mathsf{a}$ symbols is decreased by 1 with probability $1/4$, increased by 1 with probability $1/4$, and kept the same with probability $1/2$. Since a similar observation holds for all $\to_{\mathcal{P}_{\mathsf{inc}}}$-RTs, $\mathcal{P}_{\mathsf{inc}}$ is $\mathtt{AST}$. In order to detect this behavior, we have to apply the $\mathsf{g}$-rule to $\mathsf{f}(\mathsf{h}(\mathsf{g}),\mathsf{g})$ such that we can distinguish the possible results.

We restrict ourselves to innermost rewriting, as a corresponding processor for full rewriting is already unsound in the non-probabilistic setting. For that reason, our

processor can only be used to prove iAST (but not AST).

In the non-probabilistic setting [GTSF06], the rewriting processor may rewrite a redex in the right-hand side of a DP if this does not affect the construction of chains. To ensure that, the usable rules for this redex must be non-overlapping (NO). If the DP occurs in an innermost chain, then this redex is weakly innermost normalizing (or "weakly innermost terminating"), hence by NO also terminating and confluent, and thus, it has a unique normal form [Gra95].

In the probabilistic setting, to ensure that the probabilities for the normal forms stay the same, in addition to NO we require that the ADP used for the rewrite step is linear (L) (i.e., every variable occurs at most once in the left-hand side and in each term of the multi-distribution $\mu$ on the right-hand side) and non-erasing (NE) (i.e., each variable of the left-hand side occurs in each term of Supp($\mu$), as in Chapter 4).

Definition 7.6.2 – Rewriting Processor for iAST.

*Let $\mathcal{A}$ be an ADP problem with $\mathcal{A} = \mathcal{A}' \uplus \{\ell \xhookrightarrow{m} \{{}^{p_1}r_1, \ldots, {}^{p_k}r_k\}\}$. Let $\tau \in \mathrm{pos}_{\mathcal{D}}(r_j)$ for some $1 \leq j \leq k$ such that $r_j|_\tau \in \mathcal{T}$, i.e., there is no annotation below or at the position $\tau$. If $r_j \hookrightarrow_{\mathcal{A},\tau,\mathtt{t}} \{{}^{q_1}e_1, \ldots, {}^{q_h}e_h\}$, where $\hookrightarrow_{\mathcal{A},\tau,\mathtt{t}}$ denotes a rewrite step with $\hookrightarrow_{\mathcal{A}}$ at position $\tau$ and the applied ADP from $\mathcal{A}$ must have the flag* t*, then we define*

$$\mathrm{Proc}^{\mathbf{i}}_{\mathbf{r}}(\mathcal{A}) \;=\; \Big\{\mathcal{A}' \cup \{\;\; \ell \xhookrightarrow{m} \{{}^{p_1}\flat(r_1), \ldots, {}^{p_k}\flat(r_k)\},\;\; \ell \xhookrightarrow{m} \big(\{{}^{p_1}r_1, \ldots, {}^{p_k}r_k\} \setminus \{{}^{p_j}r_j\} \cup \{{}^{p_j \cdot q_1}e_1, \ldots, {}^{p_j \cdot q_h}e_h\}\big)\;\}\Big\}.$$

Here, $\{{}^{p_1}r_1, \ldots, {}^{p_k}r_k\} \setminus \{{}^{p_j}r_j\}$ only removes the $j$-th entry of the multi-distribution (even if it contains several copies of ${}^{p_j}r_j$).

In the non-probabilistic DP framework, when applying the rewriting processor to a DP problem $(\mathcal{D}, \mathcal{R})$, one only transforms the DPs $\mathcal{D}$ by rewriting, but the rules $\mathcal{R}$ are left unchanged. But since our ADPs represent both DPs and rules, when rewriting an ADP, we add a copy of the original ADP without any annotations (i.e., this corresponds to the original rule). Another change to the rewriting processor in the non-probabilistic DP framework is the requirement that there exists no annotation below $\tau$. Otherwise, rewriting would potentially remove annotations from $r_j$. For the soundness of our processor, we have to ensure that this cannot happen.

Theorem 7.6.3 – Soundness of the Rewriting Processor.

$\mathrm{Proc}^{\mathbf{i}}_{\mathbf{r}}$ as in Definition 7.6.2 is sound for iAST if one of the following cases holds:

1. $\mathcal{U}_{\mathcal{A}}(r_j|_\tau)$ is NO, and the rule used for rewriting $r_j|_\tau$ is L and NE.
2. $\mathcal{U}_{\mathcal{A}}(r_j|_\tau)$ is NO, and all its rules have the form $\ell' \xrightarrow{\mathtt{t}} \{{}^1r'\}$.
3. $\mathcal{U}_{\mathcal{A}}(r_j|_\tau)$ is NO, $r_j \hookrightarrow_{\mathcal{A},\tau,\mathtt{t}} \{{}^{q_1}e_1, \ldots, {}^{q_h}e_h\}$ is an innermost step, and $r_j|_\tau$ is a ground term.

However, the rewriting processor is not complete for iAST.

Counterexample 7.6.4 (Incompleteness of the Rewriting Processor). The ADP problem $\{\mathsf{a} \xrightarrow{\mathtt{t}} \{{}^1\mathsf{f}^\sharp(\mathsf{h}(\mathsf{g}))\}, \mathsf{h}(x) \xrightarrow{\mathtt{t}} \{{}^1\mathsf{b}\}, \mathsf{g} \xrightarrow{\mathtt{t}} \{{}^1\mathsf{g}\}, \mathsf{f}(\mathsf{b}) \xrightarrow{\mathtt{t}} \{{}^1\mathsf{a}^\sharp\}\}$ is iAST, since g has no normal form, so that the annotation of $\mathsf{f}^\sharp$ can never be used. But rewriting h(g) to b

(which is allowed by the second case of Theorem 7.6.3) yields the ADP $\mathsf{a} \overset{\mathtt{t}}{\hookrightarrow} \{{}^{1}\mathsf{f}^{\sharp}(\mathsf{b})\}$, and then $\mathsf{a}^{\sharp}$ and $\mathsf{f}^{\sharp}(\mathsf{b})$ rewrite to each other with innermost **at**-steps.

For completeness in the non-probabilistic setting [GTSF06], one uses a different definition of "non-terminating" (or "infinite") DP problems. In future work, one may examine if such a definition would also yield completeness of $\mathrm{Proc}^{\mathtt{i}}_{\mathtt{r}}$ in the probabilistic case.

*Proof of Theorem 7.6.3.* Let $\mathrm{Proc}^{\mathtt{i}}_{\mathtt{r}}(\mathcal{A}) = \{\overline{\mathcal{A}'}\}$ and $\overline{\mathcal{A}} = \overline{\mathcal{A}'} \cup \{\ell \overset{m}{\hookrightarrow} \{{}^{p_1}r_1, \ldots, {}^{p_k}r_k\}\}$. We call $\ell \overset{m}{\hookrightarrow} \{{}^{p_1}r_1, \ldots, {}^{p_k}r_k\}$ the *old* ADP, we call $\ell \overset{m}{\hookrightarrow} (\{{}^{p_1}r_1, \ldots, {}^{p_k}r_k\} \setminus \{{}^{p_j}r_j\} \cup \{{}^{p_j \cdot q_1}e_1, \ldots, {}^{p_j \cdot q_h}e_h\})$ the *new* ADP, and $\ell \overset{m}{\hookrightarrow} \{{}^{p_1}\flat(r_1), \ldots, {}^{p_k}\flat(r_k)\}$ is called the *non-annotated old* ADP. *Proof*

**First Case** The used rule is `L` and `NE`

We start with the case where $\mathcal{U}_{\mathcal{A}}(r_j|_\tau)$ is `NO` and the used rule $\hat{\ell} \overset{\mathtt{t}}{\hookrightarrow} \{{}^{\hat{p}_1}\hat{r}_1, \ldots, {}^{\hat{p}_h}\hat{r}_h\}$ is `L` and `NE`. First, note that the rule used for rewriting $r_j|_\tau$ in the rewrite processor is contained in $\mathcal{U}_{\mathcal{A}}(r_j|_\tau)$. (The rules that are applicable at position $\tau$ do not have to be applicable in an innermost RT, see [GA01, Sect. 6] and [Thi07, Ex. 5.14]. However, our over-approximation of the actual applicable rules $\mathcal{U}_{\mathcal{A}}(r_j|_\tau)$ contains the rule.) Since $\mathcal{U}_{\mathcal{A}}(r_j|_\tau)$ is `NO`, we can therefore be sure that there is only a single ADP with the flag `t` applicable at position $\tau$. ADPs with the flag `f` are not contained in $\mathcal{U}_{\mathcal{A}}(r_j|_\tau)$, and we deal with them in Step 1.1.2 below.
Assume that $\mathtt{AST}(\overset{\mathtt{i}}{\hookrightarrow}_{\mathcal{A}})$ does not hold. Then there exists an innermost $\mathcal{A}$-CT $\mathfrak{T} = (N, E)$ that converges with probability $c < 1$. We will now create an innermost $\overline{\mathcal{A}'}$-CT $\mathfrak{T}' = (N', E')$ such that $|\mathfrak{T}'| \leq |\mathfrak{T}| = c < 1$, and hence $\mathtt{AST}(\overset{\mathtt{i}}{\hookrightarrow}_{\overline{\mathcal{A}'}})$ does not hold either. The core steps of this construction are the following:

1 We iteratively remove usages of the old ADP using a transformation $\Phi$ on CTs. The limit $\mathfrak{T}^{(\infty)}$ of this iteration is an $\overline{\mathcal{A}'}$-CT that converges with probability at most $|\mathfrak{T}|$. The transformation $\Phi$ always removes one of the first usages of the old ADP at some node $x \in N$. There are two different cases, depending on whether the old ADP is applied at an annotated position or not:

   1.1 For an $\mathcal{A}$-CT $\mathfrak{T}_x$ that uses the old ADP at the root $x$ at a position $\pi \in \mathrm{pos}_{\mathcal{D}^\sharp}(t_x)$, we create a new $\overline{\mathcal{A}}$-CT $\Phi(\mathfrak{T}_x)$ that uses the new ADP at the root.

   1.2 For an $\mathcal{A}$-CT $\mathfrak{T}_x$ that uses the old ADP at the root $x$ at a position $\pi \notin \mathrm{pos}_{\mathcal{D}^\sharp}(t_x)$, we create a new $\overline{\mathcal{A}}$-CT $\Phi(\mathfrak{T}_x)$ that uses the non-annotated old ADP at the root.

This proof structure will also be used in the other cases for soundness.

**Step 1** Iteratively remove usages of the old ADP

W.l.o.g., in $\mathfrak{T}$ there exists at least one rewrite step performed at some node $x$ with the old ADP (otherwise, $\mathfrak{T}$ would already be an $\overline{\mathcal{A}'}$-CT). Furthermore, we can assume that this is the first such rewrite step in the path from the root to the node $x$ and that $x$ is a node of minimum depth with this property. We will now replace this rewrite step with a rewrite step using the new ADP such that we obtain a CT $\mathfrak{T}^{(1)}$ with the following connections between $\mathfrak{T}$ and $\mathfrak{T}^{(1)}$. Here, for an infinite path $p = v_1, v_2, \ldots$ in

a CT $\mathfrak{T} = (N, E)$, by $\mathcal{C}(p) = w_1, w_2, \ldots$ we denote the sequence of $\{\Gamma, \overline{\Gamma}\}$-labels for these nodes, i.e., we have $w_i = \Gamma$ if $v_i \in \Gamma$ or $w_i = \overline{\Gamma}$ otherwise for all $i$.

(a) $|\mathfrak{T}^{(1)}| \leq |\mathfrak{T}| = c$, and

(b) for every infinite path $p$ in $\mathfrak{T}^{(1)}$ we can find an infinite path $p'$ in $\mathfrak{T}$ such that $\mathcal{C}(p)$ and $\mathcal{C}(p')$ only differ in $\overline{\Gamma}$-nodes, i.e., we can remove $\overline{\Gamma}$-nodes from $\mathcal{C}(p')$ and add a finite number of $\overline{\Gamma}$-nodes to get $\mathcal{C}(p)$ (so $p$ contains infinitely many nodes from $\Gamma$ iff $p'$ does).

This construction only works because $\mathcal{U}_{\mathcal{A}}(r_j|_\tau)$ is `NO` and there exists no annotation below $\tau$, and the resulting CT has only at most the same probability of termination as the original one due to the fact that $\hat{\ell} \xhookrightarrow{\mathtt{t}} \{\hat{p}_1 \hat{r}_1, \ldots, \hat{p}_h \hat{r}_h\}$ is also `L` and `NE`. Let $\mathfrak{T}_x$ be the induced subtree that starts at node $x$, i.e., $\mathfrak{T}_x = \mathfrak{T}[xE^*]$. The construction defines a new tree $\Phi(\mathfrak{T}_x)$ such that (a) and (b) w.r.t. $\mathfrak{T}_x$ and $\Phi(\mathfrak{T}_x)$ holds, and where we use the new ADP at the root node $x$ instead of the old one (i.e., we pushed the first use of the old ADP deeper into the tree). Then, by replacing the subtree $\mathfrak{T}_x$ with the new tree $\Phi(\mathfrak{T}_x)$ in $\mathfrak{T}$, we get an $\mathcal{A}$-CT $\mathfrak{T}^{(1)}$, with (a) and (b) w.r.t. $\mathfrak{T}$ and $\mathfrak{T}^{(1)}$, and where we use the new ADP at node $x$ instead of the old one. We can then do this replacement iteratively for every use of the old ADP, i.e., we again replace the first use of the old ADP in $\mathfrak{T}^{(1)}$ to get $\mathfrak{T}^{(2)}$ with (a) and (b) w.r.t. $\mathfrak{T}^{(1)}$ and $\mathfrak{T}^{(2)}$, and so on. The limit of all these CTs $\lim_{i\to\infty} \mathfrak{T}^{(i)}$ is an $\overline{\mathcal{A}'}$-CT, which we denote by $\mathfrak{T}^{(\infty)}$ and that converges with probability at most $c < 1$, and hence, $\mathtt{AST}(\xrightarrow[\overline{\mathcal{A}'}]{\mathtt{i}})$ does not hold.

To see that $\mathfrak{T}^{(\infty)}$ is indeed a valid $\overline{\mathcal{A}'}$-CT, note that in every iteration of the construction we turn a use of the old ADP at minimum depth into a use of the new one. Hence, for every depth $H$ of the tree, we eventually turned every use of the old ADP up to depth $H$ into a use of the new one so that the construction will not change the tree above depth $H$ anymore, i.e., there exists an $m_H$ such that $\mathfrak{T}^{(\infty)}$ and $\mathfrak{T}^{(m_H)}$ are the same trees up to depth $H$. This means that the sequence $\lim_{i\to\infty} \mathfrak{T}^{(i)}$ really converges into a tree that satisfies all local conditions of an $\overline{\mathcal{A}'}$-CT (i.e., all conditions of Definition 7.2.12 except the liveness property). We only have to show that the last condition of a CT, namely that every infinite path in $\lim_{i\to\infty} \mathfrak{T}^{(i)}$ contains infinitely many nodes from $\Gamma$, holds as well. First, by induction on $n$ one can prove that all trees $\mathfrak{T}^{(i)}$ for all $1 \leq i \leq n$ satisfy the liveness property, because due to (b) we can find for each infinite path $p \in \mathfrak{T}^{(i)}$ an infinite path $p'$ in $\mathfrak{T}$ such that $\mathcal{C}(p)$ and $\mathcal{C}(p')$ only differ in $\overline{\Gamma}$-labels as in (b). Hence, if $p$ contains only finitely many nodes from $\Gamma$, then so does $p'$, which is a contradiction to $\mathfrak{T}$ satisfying the liveness property. Moreover, we only replace subtrees after a node in $\Gamma$, the node itself remains in $\Gamma$, and after we replaced the subtree at a node $v$, we will never replace a predecessor of $v$ anymore. This means that the subsequence between the $i$-th and $(i+1)$-th occurrence of $\Gamma$ in $p$ and the subsequence between the $i$-th and $(i+1)$-th occurrence of $\Gamma$ in $p'$ only differ in a finite number of $\overline{\Gamma}$ nodes again, for every $i$. Now, let $p = v_1, v_2, \ldots$ be an infinite path that starts at the root in $\mathfrak{T}^{(\infty)}$ and only contains finitely many nodes from $\Gamma$, i.e., there exists an $1 \leq i$ such that $v_i, v_{i+1}, \ldots$ contains no node from $\Gamma$. Again, let $m_H \in \mathbb{N}$ such that $\mathfrak{T}^{(\infty)}$ and $\mathfrak{T}^{(m_H)}$ are the same trees up to depth $H$. The path $v_1, \ldots, v_H$ must be a path in $\mathfrak{T}^{(m_H)}$ as well. For two different $H_1, H_2$ with $i < H_1 < H_2$ we know that since the path $v_i, \ldots, v_{H_2}$ contains no node from $\Gamma$, the path must also exist in $\mathfrak{T}^{(m_{H_1})}$. We can now construct an infinite path in $\mathfrak{T}^{(m_i)}$ that contains no nodes from $\Gamma$, which is a contradiction.

Next, we want to prove that we really have $|\mathfrak{T}^{(\infty)}| \leq c$. Again, by induction on $n$ one can prove that $|\mathfrak{T}^{(i)}| \leq c$ for all $1 \leq i \leq n$. Assume for a contradiction that $\mathfrak{T}^{(\infty)}$ converges

with probability greater than $c$, i.e., $|\mathfrak{T}^{(\infty)}| > c$. Then there exists a depth $H \in \mathbb{N}$ such that $\sum_{x \in \mathrm{Leaf}^{\mathfrak{T}^{(\infty)}}, \mathrm{d}(x) \leq H} p_x > c$. Here, $\mathrm{d}(x)$ denotes the depth of node $x$. Again, let $m_H \in \mathbb{N}$ such that $\mathfrak{T}^{(\infty)}$ and $\mathfrak{T}^{(m_H)}$ are the same trees up to depth $H$. But this would mean that $|\mathfrak{T}^{(m_H)}| \geq \sum_{x \in \mathrm{Leaf}^{\mathfrak{T}^{(m_H)}}, \mathrm{d}(x) \leq H} p_x = \sum_{x \in \mathrm{Leaf}^{\mathfrak{T}^{(\infty)}}, \mathrm{d}(x) \leq H} p_x > c$, which is a contradiction to $|\mathfrak{T}^{(m_H)}| \leq c$.

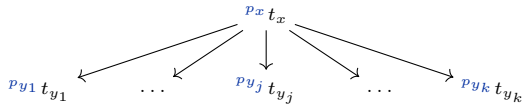
It remains to show the mentioned construction $\Phi(\circ)$.

**Step 1.1** Construction of $\Phi(\circ)$ if $\pi \in \mathrm{pos}_{\mathcal{D}^\sharp}(t_x)$

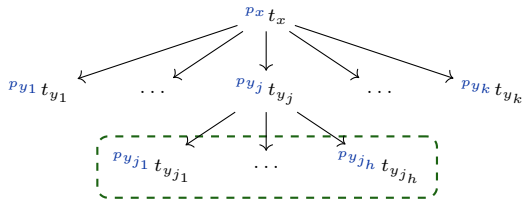
Let $\mathfrak{T}_x$ be an $\mathcal{A}$-CT that uses the old ADP at the root node $x$, i.e., $t_x \overset{\mathrm{i}}{\hookrightarrow}_{\mathcal{A}} \{p_{y_1} t_{y_1}, \ldots, p_{y_k} t_{y_k}\}$ using the ADP $\ell \overset{m}{\hookrightarrow} \{p_1 r_1, \ldots, p_k r_k\}$, the position $\pi$ with $\pi \in \mathrm{pos}_{\mathcal{D}^\sharp}(t_x)$, and a substitution $\sigma$ such that $\flat(t_x|_\pi) = \ell\sigma \in \mathtt{ANF}_{\mathcal{A}}$. Then $t_{y_j} = t_x[r_j\sigma]_\pi$ if $m = \mathtt{t}$, or $t_{y_j} = \flat^{\uparrow}_{\pi}(t_x[r_j\sigma]_\pi)$, otherwise.

**Step 1.1.1** General construction of $\Phi(\mathfrak{T}_x)$

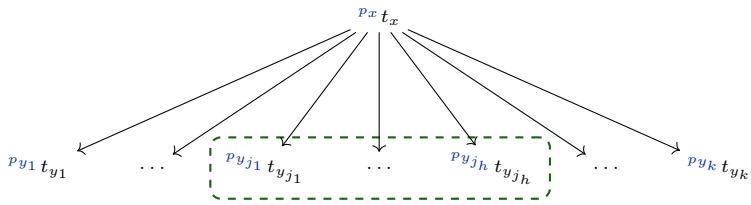
Instead of applying only the old ADP at the root $x$




where we use an arbitrary rewrite step at node $y_j$ afterwards, we want to directly apply the rewrite rule at position $\pi.\tau$ of the term $t_{y_j}$ in our CT, which we performed on $r_j$ at position $\tau$ to transform the old into the new ADP, to get

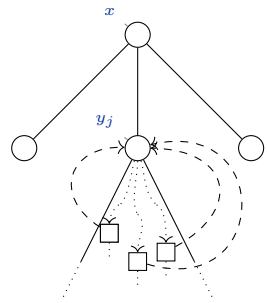



Then, we can contract the edge $(x, y_j)$ to get




and this is equivalent to applying the new ADP. Note that the rewrite step at position $\pi.\tau$ need not be an innermost rewrite step in the CT.

The subtrees that start at the nodes $y_1, \dots, y_{j-1}, y_{j+1}, \dots, y_k$ remain completely the same. We only have to construct a new subtree for node $y_j$, i.e., the node that really changed when applying the rewriting processor. To be precise, let $\mathfrak{T}_{y_j} = \mathfrak{T}_x[y_j E^*]$ be the subtree starting at node $y_j$. The construction first creates a new subtree $\Psi(\mathfrak{T}_{y_j})$ such that (a) and (b) hold w.r.t. $\mathfrak{T}_{y_j}$ and $\Psi(\mathfrak{T}_{y_j})$, and that directly performs the first rewrite step at position $\pi.\tau$ at the root of the tree, by pushing it from the original position in the tree $\mathfrak{T}_{y_j}$ to the root. This can be seen in the diagram on the side. Again, this push only results in the exact same termination probability due to our restriction that $\hat{\ell} \hookrightarrow^{\mathtt{t}} \{^{\hat{p}_1}\hat{r}_1, \dots, ^{\hat{p}_h}\hat{r}_h\}$ is $\mathtt{L}$ and $\mathtt{NE}$. Then, by replacing $\mathfrak{T}_{y_j}$ by $\Psi(\mathfrak{T}_{y_j})$ in $\mathfrak{T}_x$ we obtain a tree $\mathfrak{T}'_x$ such that (a) and (b) hold w.r.t. $\mathfrak{T}_x$ and $\mathfrak{T}'_x$, and such that we perform the desired rewrite step at node $y_j$. Finally, we contract the edge $(x, y_j)$ in $\mathfrak{T}'_x$, in order to get $\Phi(\mathfrak{T}_x)$. Again, (a) and (b) hold w.r.t. $\mathfrak{T}_x$ and $\Phi(\mathfrak{T}_x)$, and we use the new ADP at the root $x$ in $\Phi(\mathfrak{T}_x)$. Moreover, all rewrite steps in $\Phi(\mathfrak{T}_x)$ are innermost: the step at $x$ uses the new ADP with $\flat(t_x|_\pi) = \ell\sigma \in \mathtt{ANF}_{\mathcal{A}}$, and all other steps are mirrored or copied from $\mathfrak{T}_x$ with the same redexes (in the second case below, the inserted steps are innermost by construction). Before we construct $\Psi(\mathfrak{T}_{y_j})$ in Step 1.1.3, we show in Step 1.1.2 that we may ignore ADPs with the flag $\mathtt{f}$ at or below position $\pi.\tau$.




**Step 1.1.2** Removing steps with the flag $\mathtt{f}$ at or below $\pi.\tau$

Let $\kappa = \pi.\tau$. We show that we may assume the following property:

(U) If no proper predecessor of a node $z$ of $\mathfrak{T}_{y_j}$ performs a rewrite step strictly above $\kappa$, then $t_z$ has no annotations at or below $\kappa$, and if $z$ performs a rewrite step at or below $\kappa$, then it uses an ADP from $\mathcal{U}_{\mathcal{A}}(r_j|_\tau)$.

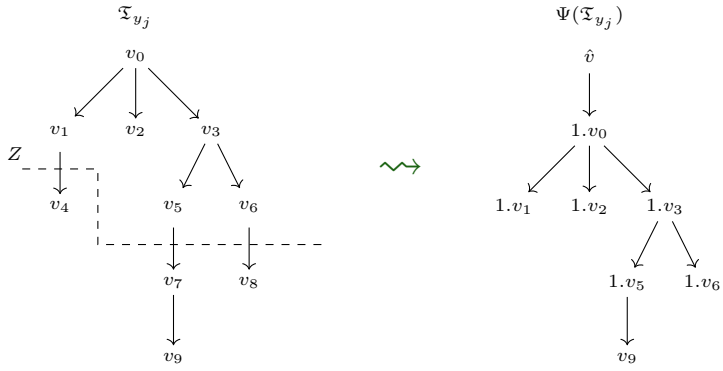
Let $z$ be such a node. Then $t_z$ has no annotations at or below $\kappa$, since $t_{y_j}$ has none and steps at non-annotated positions do not create annotations. If no node on the path from $y_j$ to $z$ (including $z$) uses an ADP with the flag $\mathtt{f}$ at a position $\geq \kappa$, then the steps at positions $\geq \kappa$ on this path are innermost steps with ADPs with the flag $\mathtt{t}$ that start with $t_{y_j}|_\kappa = r_j|_\tau\sigma$, where $\sigma$ instantiates all variables by normal forms as $\ell\sigma \in \mathtt{ANF}_{\mathcal{A}}$. So as in the proof of Theorem 7.4.14 (see also [GTSF06]), they use ADPs from $\mathcal{U}_{\mathcal{A}}(r_j|_\tau)$. Otherwise, let $v$ be the first node on this path that uses an ADP with the flag $\mathtt{f}$ at a position $\zeta \geq \kappa$. For each such node $v$ (these nodes are pairwise incomparable), we replace $\mathfrak{T}_v = \mathfrak{T}_{y_j}[vE^*]$ by an innermost $\mathcal{A}$-CT $\mathfrak{T}'_v$ that starts with $t_v$ and only performs rewrite steps at positions orthogonal to $\kappa$, such that $|\mathfrak{T}'_v| \leq |\mathfrak{T}_v|$, and such that for every infinite path $p$ in $\mathfrak{T}'_v$ there is an infinite path $p'$ in $\mathfrak{T}_v$ where $\mathcal{C}(p)$ results from $\mathcal{C}(p')$ by removing $\overline{\Gamma}$-nodes. Then the resulting tree satisfies (U), and (a) and (b) hold. This step uses none of the conditions of the theorem, so we can also use it in the other two cases.

To construct $\mathfrak{T}'_v$, we call a position $\chi$ *comparable* with $\kappa'$ if $\chi \leq \kappa'$ or $\kappa' \leq \chi$, and we let $\mathfrak{T}_w = \mathfrak{T}_{y_j}[wE^*]$ for all nodes $w$. Every node $u'$ of $\mathfrak{T}'_v$ (labeled with $^{p'_{u'}}t'_{u'}$) corresponds to a node $c(u')$ of $\mathfrak{T}_v$ and a position $\kappa_{u'} \leq \kappa$ such that $t_{c(u')}$ has no annotations at positions comparable with $\kappa_{u'}$, and $t'_{u'}$ and $t_{c(u')}$ only differ in the subterm at position $\kappa_{u'}$ and in annotations above $\kappa_{u'}$. For the root, labeled with $^1t_v$, we choose $\kappa$ and a

successor of $v$ with minimal $|\mathfrak{T}_w|$, since $v$ performs a **nf**-step at the non-annotated position $\zeta \geq \kappa$, which removes all annotations above $\zeta$. As long as $c(u')$ performs a rewrite step at a position $\chi$ comparable with $\kappa_{u'}$, this step is at a non-annotated position, so it is in $\overline{\Gamma}$ and creates no annotations (as innermost steps do not keep annotations below the redex). We skip it, i.e., we replace $c(u')$ by a successor $w$ with minimal $|\mathfrak{T}_w|$ and $\kappa_{u'}$ by the smaller one of $\chi$ and $\kappa_{u'}$. This terminates by the liveness property of $\mathfrak{T}_v$. If $c(u')$ is then a leaf, then so is $u'$. Otherwise, $c(u')$ performs a rewrite step at a position orthogonal to $\kappa_{u'}$, where $t'_{u'}$ and $t_{c(u')}$ have the same subterm, and we mirror it on $t'_{u'}$ with the same ADP, substitution, position, and case. This yields successors $u'_i$ of $u'$ with $c(u'_i) = w_i$, $\kappa_{u'_i} = \kappa_{u'}$, and $p'_{u'_i} = p'_{u'} \cdot {}^{p_{w_i}}\!/_{p_{c(u')}}$ for the successors $w_i$ of $c(u')$. In both cases, the correspondence is preserved, and the mirrored steps are innermost and at positions orthogonal to $\kappa$. Every infinite path of $\mathfrak{T}'_v$ results from an infinite path of $\mathfrak{T}_v$ by removing the skipped $\overline{\Gamma}$-nodes (which also yields the liveness property), and choosing successors with minimal $|\mathfrak{T}_w|$ yields $|\mathfrak{T}'_v| \leq |\mathfrak{T}_v|$ as in the proof of the Starting Lemma (Lemma 7.3.3).

**Step 1.1.3** Construction of $\Psi(\mathfrak{T}_{y_j})$

By Step 1.1.2, we may assume that $\mathfrak{T}_{y_j}$ satisfies (U). So before the first rewrite step at position $\pi.\tau$, every step strictly below $\pi.\tau$ uses an ADP from $\mathcal{U}_{\mathcal{A}}(r_j|_\tau)$ and thus takes place at or below a variable position of the linear term $\hat{\ell}$ (see Case (B) below). Hence, the first rewrite step at position $\pi.\tau$ uses $\hat{\ell} \overset{\mathsf{t}}{\hookrightarrow} \{{}^{\hat{p}_1}\hat{r}_1, \ldots, {}^{\hat{p}_h}\hat{r}_h\}$, the only ADP of the non-overlapping set $\mathcal{U}_{\mathcal{A}}(r_j|_\tau)$ whose left-hand side matches an instance of $\hat{\ell}$. We will move the first rewrite step that takes place at position $\pi.\tau$ from the original tree $\mathfrak{T}_{y_j}$ (example on the left below) to the top of the new tree $\Psi(\mathfrak{T}_{y_j})$ (example on the right below) and show that (a) and (b) both hold after this construction. Below, the circled nodes represent the nodes where we perform a rewrite step at position $\pi.\tau$.




We will define the $\mathcal{A}$-CT $\Psi(\mathfrak{T}_{y_j})$ that satisfies the properties (a) and (b) w.r.t. $\mathfrak{T}_{y_j}$ and $\Psi(\mathfrak{T}_{y_j})$, and that directly performs the rewrite step $t_{y_j} \hookrightarrow_{\mathcal{A},\pi.\tau,\mathsf{t}} \{{}^{\hat{p}_1}t_{y_{j_1}}, \ldots, {}^{\hat{p}_h}t_{y_{j_h}}\}$, with the rule $\hat{\ell} \overset{\mathsf{t}}{\hookrightarrow} \{{}^{\hat{p}_1}\hat{r}_1, \ldots, {}^{\hat{p}_h}\hat{r}_h\} \in \mathcal{A}$, a substitution $\hat{\sigma}$, and the position $\pi.\tau$, at the new root $\hat{v}$. Here, we have $\flat(t_{y_j}|_{\pi.\tau}) = \flat(r_j\sigma|_\tau) = \flat(r_j|_\tau\sigma) = \hat{\ell}\hat{\sigma}$. Let $Z$ be the set of all nodes $v$ of $\mathfrak{T}_{y_j}$ where we did not perform a rewrite step at position $\pi.\tau$ in the path from the root $y_j$ to the node $v$, or $v$ is the first node in the path that performs a rewrite step at position $\pi.\tau$. In the example we have $Z = \{v_0, \ldots, v_6\} \setminus \{v_4\}$. For each of these nodes $z \in Z$ and each $1 \leq e \leq h$, we create a new node $e.z \in N'$ with

edges as in $\mathfrak{T}_{y_j}$ for the nodes in $Z$, e.g., for the node $1.v_3$ we create edges to $1.v_5$ and $1.v_6$. Furthermore, we add the edges from the new root $\hat{v}$ to the nodes $e.y_j$ for all $1 \leq e \leq h$. Remember that $y_j$ was the root in the tree $\mathfrak{T}_{y_j}$ and has to be contained in $Z$. For example, for the node $\hat{v}$ we create an edge to $1.v_0$. For all these new nodes in $Z$, we show that the following holds:

(T1) $t_z[\hat{r}_e\gamma]_{\pi.\tau} \overset{\flat}{=} t'_{e.z}$ for the substitution $\gamma$ such that $\flat(t_z|_{\pi.\tau}) = \hat{\ell}\gamma$.

(T2) $\mathrm{pos}_{\mathcal{D}^\sharp}(t_z[\flat(\hat{r}_e)\gamma]_{\pi.\tau}) \subseteq \mathrm{pos}_{\mathcal{D}^\sharp}(t'_{e.z})$.

(T3) $p_{e.z}^{\Psi(\mathfrak{T}_{y_j})} = p_z^{\mathfrak{T}_{y_j}} \cdot \hat{p}_e$.

Note that we only regard the subtree of the $j$-th child of the root. By the prerequisite in the definition of the rewrite processor (there is no annotation below or at position $\tau$), there are no annotations on or below $\pi.\tau$ for all nodes in $Z$.

Now, for a leaf $e.z \in N'$ either $z \in N$ is also a leaf (e.g., node $v_2$) or we rewrite the position $\pi.\tau$ at node $z$ in $\mathfrak{T}_{y_j}$ (e.g., node $v_1$). If we rewrite $t_z \overset{\mathsf{i}}{\hookrightarrow}_{\mathcal{A},\pi.\tau,\mathsf{t}} \{^{\hat{p}_1}t_{w_1}, \ldots, ^{\hat{p}_h}t_{w_h}\}$, then we have $t_{w_e} = t_z[\flat(\hat{r}_e)\gamma]_{\pi.\tau}$. Here, we get $t_{w_e} \overset{\flat}{=} t_z[\flat(\hat{r}_e)\gamma]_{\pi.\tau} \overset{\flat}{=}_{(T1)} t'_{e.z}$, $\mathrm{pos}_{\mathcal{D}^\sharp}(t_{w_e}) \subseteq \mathrm{pos}_{\mathcal{D}^\sharp}(t_z[\flat(\hat{r}_e)\gamma]_{\pi.\tau}) \subseteq_{(T2)} \mathrm{pos}_{\mathcal{D}^\sharp}(t'_{e.z})$ and $p_{e.z}^{\Psi(\mathfrak{T}_{y_j})} = p_z^{\mathfrak{T}_{y_j}} \cdot \hat{p}_e =_{(T3)} p_{w_e}^{\mathfrak{T}_{y_j}}$, and we can again copy the rest of this subtree of $\mathfrak{T}_{y_j}$ in our newly generated tree $\Psi(\mathfrak{T}_{y_j})$. In our example, $v_1$ has the only successor $v_4$, hence we can copy the subtree starting at node $v_4$, which is only the node itself, to the node $1.v_1$ in $\Psi(\mathfrak{T}_{y_j})$. For $v_5$ we have the only successor $v_7$, hence we can copy the subtree starting at node $v_7$, which is the node itself together with its successor $v_9$, to the node $1.v_5$ in $\Psi(\mathfrak{T}_{y_j})$. So essentially, we just have to define the part of the tree before we reach the rewrite step in $\mathfrak{T}_{y_j}$, and then, we have to show that (a) and (b) for $\mathfrak{T}_{y_j}$ and $\Psi(\mathfrak{T}_{y_j})$ are satisfied. We show the latter first, and then explain why this labeling indeed gives us a valid $\mathcal{A}$-CT.

We start by showing (a) for $\mathfrak{T}_{y_j}$ and $\Psi(\mathfrak{T}_{y_j})$. Let $u$ be a leaf in $\Psi(\mathfrak{T}_{y_j})$. If $u = e.v$ for some node $v \in Z$ that is a leaf in $\mathfrak{T}_{y_j}$ (e.g., node $1.v_2$), then also $e.v$ must be a leaf in $\Psi(\mathfrak{T}_{y_j})$ for every $1 \leq e \leq h$. Here, we get $\sum_{1\leq e\leq h} p_{e.v}^{\Psi(\mathfrak{T}_{y_j})} \overset{(T3)}{=} \sum_{1\leq e\leq h} p_v^{\mathfrak{T}_{y_j}} \cdot \hat{p}_e = p_v^{\mathfrak{T}_{y_j}} \cdot \sum_{1\leq e\leq h} \hat{p}_e = p_v^{\mathfrak{T}_{y_j}} \cdot 1 = p_v^{\mathfrak{T}_{y_j}}$. If $u = e.v$ for some node $v \in Z$ that is not a leaf in $\mathfrak{T}_{y_j}$ (e.g., node $1.v_1$), then we know by construction that all successors of $v$ in $\mathfrak{T}_{y_j}$ are not contained in $Z$ and are leaves. Here, we get $p_{e.v}^{\Psi(\mathfrak{T}_{y_j})} \overset{(T3)}{=} p_v^{\mathfrak{T}_{y_j}} \cdot \hat{p}_e = p_w^{\mathfrak{T}_{y_j}}$ for the (unique) $e$-th successor $w$ of $v$. Finally, if $u$ does not have the form $u = e.v$, then $u$ is also a leaf in $\mathfrak{T}_{y_j}$ with $p_u^{\Psi(\mathfrak{T}_{y_j})} = p_u^{\mathfrak{T}_{y_j}}$. Note that these cases cover no leaf of $\mathfrak{T}_{y_j}$ twice. This implies that we have

$$
\begin{aligned}
|\Psi(\mathfrak{T}_{y_j})| &= \sum_{v \in \mathrm{Leaf}^{\Psi(\mathfrak{T}_{y_j})}} p_v^{\Psi(\mathfrak{T}_{y_j})} \\
&= \sum_{\substack{e.v \in \mathrm{Leaf}^{\Psi(\mathfrak{T}_{y_j})} \\ v \in Z}} p_{e.v}^{\Psi(\mathfrak{T}_{y_j})} + \sum_{\substack{e.v \in \mathrm{Leaf}^{\Psi(\mathfrak{T}_{y_j})} \\ v \in wE, w \notin Z, w \in Z}} p_{e.v}^{\Psi(\mathfrak{T}_{y_j})} + \sum_{\substack{v \in \mathrm{Leaf}^{\Psi(\mathfrak{T}_{y_j})} \\ v \in \mathrm{Leaf}^{\mathfrak{T}_{y_j}}, v \in wE, w \notin Z}} p_v^{\Psi(\mathfrak{T}_{y_j})} \\
&\leq \sum_{\substack{v \in \mathrm{Leaf}^{\mathfrak{T}_{y_j}} \\ v \in Z}} \left( \sum_{1\leq e\leq h} p_{e.v}^{\Psi(\mathfrak{T}_{y_j})} \right) + \sum_{\substack{v \in \mathrm{Leaf}^{\mathfrak{T}_{y_j}}, 1\leq e\leq h \\ v \in wE, v \notin Z, w \in Z}} p_{e.v}^{\Psi(\mathfrak{T}_{y_j})} + \sum_{\substack{v \in \mathrm{Leaf}^{\mathfrak{T}_{y_j}} \\ v \in wE, w \notin Z}} p_v^{\Psi(\mathfrak{T}_{y_j})}
\end{aligned}
$$

$$\leq \sum_{\substack{v \in \mathrm{Leaf}^{\mathfrak{T}_{y_j}} \\ v \in Z}} p_v^{\mathfrak{T}_{y_j}} + \sum_{\substack{v \in \mathrm{Leaf}^{\mathfrak{T}_{y_j}} \\ v \in wE, v \notin Z, w \in Z}} p_v^{\mathfrak{T}_{y_j}} + \sum_{\substack{v \in \mathrm{Leaf}^{\mathfrak{T}_{y_j}} \\ v \in wE, w \notin Z}} p_v^{\mathfrak{T}_{y_j}}$$
$$= \sum_{v \in \mathrm{Leaf}^{\mathfrak{T}_{y_j}}} p_v^{\mathfrak{T}_{y_j}} = |\mathfrak{T}_{y_j}|.$$

Next, we show (b) for $\mathfrak{T}_{y_j}$ and $\Psi(\mathfrak{T}_{y_j})$. Let $p = u_0, u_1, \ldots$ be an infinite path in $\Psi(\mathfrak{T}_{y_j})$ that starts at the root $\hat{v}$. If for all $1 \leq i$ we have $u_i = e.v_i$ for some node $v_i \in Z$ and $1 \leq e \leq h$, then $p' = v_1, \ldots$ is our desired path in $\mathfrak{T}_{y_j}$. Otherwise, there is a maximal $1 \leq o$ such that for all $1 \leq i \leq o$ we have $u_i = e.v_i$ for some node $v_i \in Z$ and $1 \leq e \leq h$. Then our desired path is $v_1, \ldots, v_o, w, u_{o+1}, \ldots$. Here, $w$ is the $e$-th successor of $v_o$ in $\mathfrak{T}_{y_j}$. Note that in the first case, we add one $\overline{\Gamma}$-label at the start of our path, while in the other case, we just move an $\overline{\Gamma}$-label from the first position to a later one in the path. This shows that (b) is satisfied.

Finally, we prove that $\Psi(\mathfrak{T}_{y_j})$ is a valid $\mathcal{A}$-CT. We only need to prove that the construction satisfies all of our conditions for the nodes in $Z$. As the rest of the tree is copied, we can be sure that all local conditions of an $\mathcal{A}$-CT are satisfied. Additionally, due to (b) w.r.t. $\mathfrak{T}_{y_j}$ and $\Psi(\mathfrak{T}_{y_j})$, we get the liveness property as well, because $\mathfrak{T}_{y_j}$ satisfies it.

At the root, after applying the old ADP, we directly perform the rewrite step $t_{y_j} \hookrightarrow_{\mathcal{A}} \{{}^{\hat{p}_1}t'_{1.y_j}, \ldots, {}^{\hat{p}_h}t'_{h.y_j}\}$, with the rule $\hat{\ell} \overset{\mathtt{t}}{\hookrightarrow} \{{}^{\hat{p}_1}\hat{r}_1, \ldots, {}^{\hat{p}_h}\hat{r}_h\} \in \mathcal{A}$, a substitution $\hat{\sigma}$, and the position $\pi.\tau$. Then, $t'_{e.y_j} = t_{y_j}[\flat(\hat{r}_e)\hat{\sigma}]_{\pi.\tau}$. Here, the conditions (T1)–(T3) are clearly satisfied. We now construct the rest of this subtree by mirroring the rewrite steps in the original tree (which is always possible due to the fact that $\mathcal{U}_{\mathcal{A}}(r_j|_\tau)$ is NO and $\mathfrak{T}_{y_j}$ satisfies (U)), and once we encounter the rewrite step that we moved to the top, i.e., once we use a rewrite step at position $\pi.\tau$, we skip this rewrite step and directly go on with the $e$-th successor if we are in a path that went to the $e$-th successor in the initial rewrite step, as described above. In the following, we distinguish between two different cases for a rewrite step at a node $u$.

(A) We use a step with $\overset{\mathtt{i}}{\hookrightarrow}_{\mathcal{A}}$ in $\mathfrak{T}_{y_j}$ at a position orthogonal to $\pi.\tau$.

(B) We use a step with $\overset{\mathtt{i}}{\hookrightarrow}_{\mathcal{A}}$ in $\mathfrak{T}_{y_j}$ at a position below $\pi.\tau$. Note that this is the more interesting case, where we need to use the properties L and NE.

Note that we cannot rewrite above $\pi.\tau$ before rewriting at position $\pi.\tau$ due to the innermost restriction.

**(A) If we have** $t_u \overset{\mathtt{i}}{\hookrightarrow}_{\mathcal{A}} \{{}^{p_{q_1}/p_u}t_{q_1}, \ldots, {}^{p_{q_{\overline{h}}}/p_u}t_{q_{\overline{h}}}\}$, then there is a rule $\bar{\ell} \overset{\overline{m}}{\hookrightarrow} \{{}^{\bar{p}_1}\bar{r}_1, \ldots, {}^{\bar{p}_{\overline{h}}}\bar{r}_{\overline{h}}\} \in \mathcal{A}$, a substitution $\delta$, and a position $\zeta \in \mathbb{N}^*$ with $\flat(t_u|_\zeta) = \bar{\ell}\delta \in \mathtt{ANF}_{\mathcal{A}}$. Furthermore, let $\zeta \perp \pi.\tau$. Then, we have $t'_{e.u}|_\zeta \overset{\flat}{=}_{(T1)} t_u[\hat{r}_e\gamma]_{\pi.\tau}|_\zeta = t_u|_\zeta$ for the substitution $\gamma$ such that $t_u|_{\pi.\tau} = \hat{\ell}\gamma$, and we can rewrite $t'_{e.u}$ using the same rule, same substitution, and same position. Then (T3) is again satisfied. Furthermore, we have $t'_{e.q_j} = t'_{e.u}[\bar{r}_j\delta]_\zeta$ if $\zeta \in \mathrm{pos}_{\mathcal{D}^\sharp}(t'_{e.u})$ and $\overline{m} = \mathtt{t}$, $t'_{e.q_j} = \flat^{\uparrow}_{\zeta}(t'_{e.u}[\bar{r}_j\delta]_\zeta)$ if $\zeta \in \mathrm{pos}_{\mathcal{D}^\sharp}(t'_{e.u})$ and $\overline{m} = \mathtt{f}$, $t'_{e.q_j} = t'_{e.u}[\flat(\bar{r}_j)\delta]_\zeta$ if $\zeta \notin \mathrm{pos}_{\mathcal{D}^\sharp}(t'_{e.u})$ and $\overline{m} = \mathtt{t}$, or $t'_{e.q_j} = \flat^{\uparrow}_{\zeta}(t'_{e.u}[\flat(\bar{r}_j)\delta]_\zeta)$ if $\zeta \notin \mathrm{pos}_{\mathcal{D}^\sharp}(t'_{e.u})$ and $\overline{m} = \mathtt{f}$. In all four cases we get (T1) (using the same substitution $\gamma$) and (T2) as well.

**(B) If we have** $t_u \overset{\mathtt{i}}{\hookrightarrow}_{\mathcal{A}} \{{}^{p_{q_1}/p_u}t_{q_1}, \ldots, {}^{p_{q_{\overline{h}}}/p_u}t_{q_{\overline{h}}}\}$, then there is a rule $\bar{\ell} \overset{\overline{m}}{\hookrightarrow} \{{}^{\bar{p}_1}\bar{r}_1, \ldots, {}^{\bar{p}_{\overline{h}}}\bar{r}_{\overline{h}}\} \in \mathcal{A}$, a substitution $\delta$, and a position $\zeta \in \mathbb{N}^*$ with $\flat(t_u|_\zeta) = \bar{\ell}\delta \in \mathtt{ANF}_{\mathcal{A}}$. Furthermore, let $\zeta > \pi.\tau$. Since $\hat{\ell} \overset{\mathtt{t}}{\hookrightarrow} \{{}^{\hat{p}_1}\hat{r}_1, \ldots, {}^{\hat{p}_h}\hat{r}_h\}$ is L and NE, we know that $\hat{\ell}$

contains exactly the same variables as $\hat{r}_e$ and all of them exactly once. Furthermore, by (U), the used ADP is contained in $\mathcal{U}_{\mathcal{A}}(r_j|_\tau)$. Since $\mathcal{U}_{\mathcal{A}}(r_j|_\tau)$ is non-overlapping, we know that the rewriting must be completely inside the substitution $\gamma$ for the substitution $\gamma$ such that $t_u|_{\pi.\tau} = \hat{\ell}\gamma$, i.e., there is a position $\eta_c$ of a variable $c$ in $\hat{\ell}$ and another position $\chi_c$ with $\pi.\tau.\eta_c.\chi_c = \zeta$. Let $\varphi_e(c)$ be the (unique) variable position of $c$ in $\hat{r}_e$. Then, we have $t'_{e.u}|_{\pi.\tau.\varphi_e(c).\chi_c} \overset{\flat}{=}_{(T1)} t_u[\hat{r}_e\gamma]_{\pi.\tau}|_{\pi.\tau.\varphi_e(c).\chi_c} = \hat{r}_e\gamma|_{\varphi_e(c).\chi_c} = \gamma(c)|_{\chi_c} = \hat{\ell}\gamma|_{\eta_c.\chi_c} = t_u|_{\pi.\tau.\eta_c.\chi_c}$, and we can rewrite $t'_{e.u}$ using the same rule, same substitution, and position $\pi.\tau.\varphi_e(c).\chi_c$. Again, in all four cases (T1)–(T3) are satisfied.

**Step 1.2** $\pi \notin \mathrm{pos}_{\mathcal{D}^\sharp}(t_x)$

If we have $\pi \notin \mathrm{pos}_{\mathcal{D}^\sharp}(t_x)$, then we can simply use the non-annotated old ADP instead of the old one. Since we would remove all annotations in the right-hand side of the rule anyway, due to $\pi \notin \mathrm{pos}_{\mathcal{D}^\sharp}(t_x)$, this leads to the same labels in the resulting $\overline{\mathcal{A}}$-CT.

**Second Case** All rules have the form $\ell' \xhookrightarrow{\mathtt{t}} \{^1 r'\}$

Next, let $\mathcal{U}_{\mathcal{A}}(r_j|_\tau)$ be NO such that all its rules have the form $\ell' \xhookrightarrow{\mathtt{t}} \{^1 r'\}$. Then the used ADP is $\hat{\ell} \xhookrightarrow{\mathtt{t}} \{^1 \hat{r}_1\}$, so $h = 1$ and $q_1 = 1$. Steps 1, 1.2, and 1.1.2 are as in the first case, so we may assume that $\mathfrak{T}_{y_j}$ satisfies (U). Let $\hat{\ell}\hat{\sigma} = r_j|_\tau\sigma = \flat(t_{y_j}|_{\pi.\tau})$. Using the new ADP at $x$ only replaces $y_j$ by a node $y'_j$ labeled with ${}^{p_{y_j}}t_{y_j}[\flat(\hat{r}_1)\hat{\sigma}]_{\pi.\tau}$, so it suffices to construct an innermost $\mathcal{A}$-CT $\mathfrak{T}'_{y_j}$ with the root $y'_j$ such that (a) and (b) hold w.r.t. $\mathfrak{T}_{y_j}$ and $\mathfrak{T}'_{y_j}$.

Let $\to_{\mathcal{U}}$ denote innermost rewriting (w.r.t. $\mathtt{ANF}_{\mathcal{A}}$) with the rules $\ell' \to \flat(r')$ for $\ell' \xhookrightarrow{\mathtt{t}} \{^1 r'\} \in \mathcal{U}_{\mathcal{A}}(r_j|_\tau)$. By NO, two different $\to_{\mathcal{U}}$-steps on the same term are at orthogonal positions (at the same position, the left-hand sides would overlap, and a redex in $\mathtt{ANF}_{\mathcal{A}}$ cannot contain another one). So $\to_{\mathcal{U}}$ has the diamond property, and by induction on $n$, if $s \to_{\mathcal{U}}^n u$ for a $\to_{\mathcal{U}}$-normal form $u$, then every maximal $\to_{\mathcal{U}}$-sequence that starts with $s$ ends in $u$ after $n$ steps (cf. [Gra95], [Gra96, Thm. 3.2.11]).

Let $Z$ be the set of all nodes $z$ of $\mathfrak{T}_{y_j}$ such that no proper predecessor of $z$ performs a rewrite step at a position $< \pi.\tau$. By (U), every step of a node $z \in Z$ at a position $\geq \pi.\tau$ is a $\to_{\mathcal{U}}$-step on $t_z|_{\pi.\tau}$ at a non-annotated position, i.e., $z \in \overline{\Gamma}$ has a single successor with the same probability. We obtain $\mathfrak{T}'_{y_j}$ by contracting the edges of these nodes (which is possible by the liveness property), by replacing the subterm at position $\pi.\tau$ by $\flat(\hat{r}_1)\hat{\sigma}$ in the labels of the remaining nodes from $Z$ (so that their steps at positions orthogonal to $\pi.\tau$ remain possible), and by inserting a sequence $\flat(\hat{r}_1)\hat{\sigma} \to_{\mathcal{U}}^* u$ at position $\pi.\tau$ before each $z \in Z$ that performs a step at a position $< \pi.\tau$, where $u = \flat(t_z|_{\pi.\tau})$. Then we reach $t_z$ again and copy $\mathfrak{T}_{y_j}[zE^*]$. The inserted steps are innermost **nt**-steps with a single successor, and every path contains at most one such insertion. So the leaves of $\mathfrak{T}'_{y_j}$ and $\mathfrak{T}_{y_j}$ correspond to each other with the same probabilities, which implies (a), and the infinite paths only differ in the contracted and finitely many inserted $\overline{\Gamma}$-nodes, which implies (b).

It remains to show $\flat(\hat{r}_1)\hat{\sigma} \to_{\mathcal{U}}^* u$. Since the step at $z$ is innermost, we have $u \in \mathtt{NF}_{\mathcal{A}}$, and by (U), $\hat{\ell}\hat{\sigma} \to_{\mathcal{U}}^* u$. So every maximal $\to_{\mathcal{U}}$-sequence that starts with a term reachable from $\hat{\ell}\hat{\sigma}$ is finite and ends in $u$. In particular, for every variable $y$ of $\hat{\ell}$, there is a $\to_{\mathcal{U}}$-normal form $\theta(y)$ with $\hat{\sigma}(y) \to_{\mathcal{U}}^* \theta(y)$, as these steps can also be performed in $\hat{\ell}\hat{\sigma}$. Performing them at all occurrences of the variables yields $\hat{\ell}\hat{\sigma} \to_{\mathcal{U}}^* \hat{\ell}\theta$ and $\flat(\hat{r}_1)\hat{\sigma} \to_{\mathcal{U}}^* \flat(\hat{r}_1)\theta$. As $\hat{\ell}\theta \notin \mathtt{NF}_{\mathcal{A}}$, we have $\hat{\ell}\theta \neq u$, so $\hat{\ell}\theta$ is no $\to_{\mathcal{U}}$-normal form. Since

the subterms $\theta(y)$ are $\to_{\mathcal{U}}$-normal forms, by NO the only possible step is $\hat{\ell}\theta \to_{\mathcal{U}} \flat(\hat{r}_1)\theta$ at the root. Hence, $\flat(\hat{r}_1)\hat{\sigma} \to_{\mathcal{U}}^* \flat(\hat{r}_1)\theta \to_{\mathcal{U}}^* u$.

**Third Case** Innermost step on a ground term

Finally, we prove the theorem in the case where $\mathcal{U}_{\mathcal{A}}(r_j|_\tau)$ is NO, $r_j|_\tau$ is a ground term, and we have $r_j \xhookrightarrow{\mathrm{i}}_{\mathcal{A},\tau,\mathtt{t}} \{^{q_1}e_1, \ldots, ^{q_h}e_h\}$, i.e., it is an innermost step.
Once again, we use the same idea as in the proof for the first case but the construction of $\Phi(\circ)$ is, again, easier. Note that if $\mathcal{U}_{\mathcal{A}}(r_j|_\tau)$ is non-overlapping, $r_j$ contains no variable below position $\tau$, and we perform an innermost rewrite step, then this is always an innermost rewrite step in every possible CT and this is the only possible rewrite step with an ADP with the flag t at this position, as all ADPs with the flag t whose left-hand side matches $r_j|_\tau$ are contained in $\mathcal{U}_{\mathcal{A}}(r_j|_\tau)$. By Step 1.1.2, we may assume that $\mathfrak{T}_{y_j}$ satisfies (U), i.e., no ADP with the flag f is used at position $\pi.\tau$ before a rewrite step above $\pi.\tau$. Hence, we can move this innermost step directly after the use of the ADP using the construction $\Psi(\circ)$. Here, only Case (A) can happen. The reason is that we have to perform this rewrite step eventually if we want to rewrite above position $\tau$, and all other rewrite steps that we can perform in such a situation would be at orthogonal positions. So we get the same normal forms in the leaves with the same probability. *QED*

In the last case of Theorem 7.6.3, for any instantiation only a single innermost rewrite step with an ADP with the flag t is possible for $r_j|_\tau$. The restriction to innermost rewrite steps is only useful if $r_j|_\tau$ is ground. Otherwise, an innermost step on $r_j|_\tau$ might become a non-innermost step when instantiating $r_j|_\tau$'s variables.

Next, we show why the rewriting processor needs the new requirement L in the first case that was not imposed in the non-probabilistic setting. More precisely, we give counterexamples for soundness if the used rule is not left-linear, i.e., a variable occurs more than once in the left-hand side, and if the used rule is not right-linear, i.e., a variable occurs more than once in a term on the right-hand side. The other new requirement NE is currently used in the soundness proof, but we were unable to find a counterexample to soundness if the used rule is not NE. In fact, we conjecture that one can omit this requirement, but then one needs a much more complicated construction and estimation of the resulting termination probability in the soundness proof. The reason is that with only L we can guarantee that performing this rewrite step at a (possibly) non-innermost redex can only increase the probability of innermost termination for the rewritten subterm but not decrease it. Increasing the probability of termination for a proper subterm without any annotations means that we have a higher probability to apply a rewrite step at the position of an annotated symbol above it. Recall that a CT requires that on each infinite path, redexes with annotated root symbol have to be rewritten infinitely often (they correspond to the rewrite steps with Case **at** or **af** of Definition 7.2.12). Thus, a higher probability of termination of the proper subterm leads to a lower probability of the leaves in the CT as the probability to rewrite redexes with annotated root is higher. However, proving this requires a much more involved approximation of the probability for termination than our current proof, where we additionally require NE.

Counterexample 7.6.5 (Left-Linearity is Required for Soundness). To see why left-linearity is required for soundness of the rewriting processor in the probabilistic setting,

consider the ADP problem $\mathcal{A}_{\mathsf{ll}}$ with

$$\mathsf{g}(\mathsf{f}(x,y)) \overset{\mathsf{f}}{\hookrightarrow} \{{}^{1}\mathsf{d}(\mathsf{g}^{\sharp}(\mathsf{f}(\mathsf{a},\mathsf{a})),\mathsf{g}^{\sharp}(\mathsf{f}(\mathsf{a},\mathsf{a})),\mathsf{g}^{\sharp}(\mathsf{f}(\mathsf{a},\mathsf{a})))\} \qquad \mathsf{f}(x,x) \overset{\mathsf{t}}{\hookrightarrow} \{{}^{1}\mathsf{e}(\mathsf{f}(\mathsf{a},\mathsf{a}))\} \quad (7.49)$$

$$\mathsf{a} \overset{\mathsf{t}}{\hookrightarrow} \{{}^{1/2}\mathsf{b}_1, {}^{1/2}\mathsf{b}_2\} \quad (7.50)$$

This example could also be made non-erasing by adding $x$ as an additional argument to $\mathsf{e}$ and by instantiating $x$ and $y$ by all possible values from $\{\mathsf{b}_1, \mathsf{b}_2\}$ in the g-ADP. The ADP problem $\mathcal{A}_{\mathsf{ll}}$ does not satisfy $\mathtt{AST}(\overset{\mathsf{i}}{\hookrightarrow}_{\mathcal{A}_{\mathsf{ll}}})$, as it allows for the $\mathcal{A}_{\mathsf{ll}}$-CT depicted in Figure 7.21 whose leaves have a probability $< 1$.

Here, $\mathsf{g}^{\sharp}(\mathsf{f}(\mathsf{a},\mathsf{a}))^3$ denotes the term $\mathsf{d}(\mathsf{g}^{\sharp}(\mathsf{f}(\mathsf{a},\mathsf{a})),\mathsf{g}^{\sharp}(\mathsf{f}(\mathsf{a},\mathsf{a})),\mathsf{g}^{\sharp}(\mathsf{f}(\mathsf{a},\mathsf{a})))$. Chain trees starting in $\mathsf{g}^{\sharp}(\mathsf{e}(\mathsf{f}(\mathsf{a},\mathsf{a})))$ can never use a rewrite step with the g-ADP anymore and therefore, they converge with probability 1. So we can rewrite a single $\mathsf{g}^{\sharp}$-term to a leaf with a probability of $1/2$ and to three copies of itself with a probability of $1/2$. Hence, the full CT corresponds to a random walk that terminates with probability $< 1$. But without the restriction to left-linearity, we could apply the rewriting processor and replace the g-ADP by $\mathsf{g}(\mathsf{f}(x,y)) \overset{\mathsf{f}}{\hookrightarrow} \{{}^{1}\mathsf{d}(\mathsf{g}^{\sharp}(\mathsf{e}(\mathsf{f}(\mathsf{a},\mathsf{a}))),\mathsf{g}^{\sharp}(\mathsf{e}(\mathsf{f}(\mathsf{a},\mathsf{a}))),\mathsf{g}^{\sharp}(\mathsf{e}(\mathsf{f}(\mathsf{a},\mathsf{a}))))\}$ and several g-ADPs without annotations, since the usable rules $\mathcal{U}_{\mathcal{A}_{\mathsf{ll}}}(\mathsf{f}(\mathsf{a},\mathsf{a})) = \{(7.49),(7.50)\}$ are $\mathtt{NO}$. Now in every path of the CT this ADP can be used at most once and hence, the resulting ADP problem satisfies $\mathtt{AST}(\overset{\mathsf{i}}{\hookrightarrow}_{\mathcal{A}})$, which shows unsoundness of the rewriting processor without left-linearity.

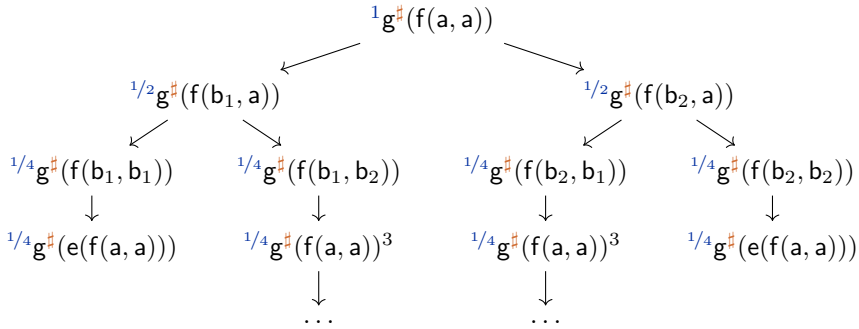



Figure 7.21: Innermost $\mathcal{A}_{\mathsf{ll}}$-CT starting with $\mathsf{g}^{\sharp}(\mathsf{f}(\mathsf{a},\mathsf{a}))$.

Counterexample 7.6.6 (Right-Linearity is Required for Soundness). To show the need for right-linearity, consider the ADP problem $\mathcal{A}_{\mathsf{rl}}$ with

$$\mathsf{f}(\mathsf{e}(\mathsf{b}_1,\mathsf{b}_1)) \overset{\mathsf{f}}{\hookrightarrow} \{{}^{1}\mathsf{h}(\mathsf{f}^{\sharp}(\mathsf{d}(\mathsf{a})),\mathsf{f}^{\sharp}(\mathsf{d}(\mathsf{a})),\mathsf{f}^{\sharp}(\mathsf{d}(\mathsf{a})),\mathsf{f}^{\sharp}(\mathsf{d}(\mathsf{a})))\} \qquad \mathsf{d}(x) \overset{\mathsf{t}}{\hookrightarrow} \{{}^{1}\mathsf{e}(x,x)\} \quad (7.51)$$

$$\mathsf{a} \overset{\mathsf{t}}{\hookrightarrow} \{{}^{1/2}\mathsf{b}_1, {}^{1/2}\mathsf{b}_2\} \quad (7.52)$$

Note that for the term $\mathsf{d}(\mathsf{a})$ we have the innermost $\mathcal{A}_{\mathsf{rl}}$-CT depicted in Figure 7.22.

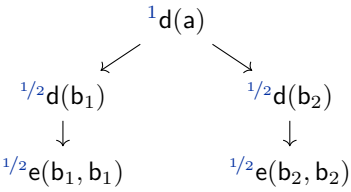



Figure 7.22: Innermost $\mathcal{A}_{\mathsf{rl}}$-CT starting with $\mathsf{d}(\mathsf{a})$.

Hence, the term $\mathsf{f}^\sharp(\mathsf{d}(\mathsf{a}))$ can rewrite to $\mathsf{f}^\sharp(\mathsf{e}(\mathsf{b}_1,\mathsf{b}_1))$ and then to four occurrences of $\mathsf{f}^\sharp(\mathsf{d}(\mathsf{a}))$ again with probability $1/2$, or it reaches a normal form with probability $1/2$. This is again a random walk that terminates with probability $< 1$, so that the ADP problem $\mathcal{A}_{\mathsf{rl}}$ does not satisfy $\mathtt{AST}(\overset{\mathsf{i}}{\hookrightarrow}_{\mathcal{A}_{\mathsf{rl}}})$. But without the restriction to right-linearity of the used rule, it would be possible to apply the rewriting processor and replace the $\mathsf{f}$-ADP by $\mathsf{f}(\mathsf{e}(\mathsf{b}_1,\mathsf{b}_1)) \overset{\mathsf{f}}{\hookrightarrow} \{^{1}\mathsf{h}(\mathsf{f}^\sharp(\mathsf{e}(\mathsf{a},\mathsf{a})),\mathsf{f}^\sharp(\mathsf{e}(\mathsf{a},\mathsf{a})),\mathsf{f}^\sharp(\mathsf{e}(\mathsf{a},\mathsf{a})),\mathsf{f}^\sharp(\mathsf{e}(\mathsf{a},\mathsf{a})))\}$ and several $\mathsf{f}$-ADPs without annotations, as $\mathcal{U}_{\mathcal{A}_{\mathsf{rl}}}(\mathsf{d}(\mathsf{a})) = \{(7.51),(7.52)\}$ is $\mathtt{NO}$. The term $\mathsf{e}(\mathsf{a},\mathsf{a})$ can now be rewritten to the term $\mathsf{e}(\mathsf{b}_1,\mathsf{b}_1)$ with probability $1/4$, whereas one obtains a term of the form $\mathsf{e}(\mathsf{b}_i,\mathsf{b}_j)$ with $i \neq 1$ or $j \neq 1$ with probability $3/4$. Hence, now $\mathsf{f}^\sharp(\mathsf{e}(\mathsf{a},\mathsf{a}))$ can rewrite to a term with four subterms $\mathsf{f}^\sharp(\mathsf{e}(\mathsf{a},\mathsf{a}))$ only with a probability of $1/4$, and it reaches a leaf with probability $3/4$. This is now a random walk that terminates with probability 1 and the same happens for all possible CTs. Hence, the resulting ADP problem satisfies $\mathtt{AST}(\overset{\mathsf{i}}{\hookrightarrow}_{\mathcal{A}})$, which shows unsoundness of the rewriting processor without right-linearity.

With the processor of Theorem 7.6.3 we can now prove $\mathtt{AST}(\overset{\mathsf{i}}{\to}_{\mathcal{P}_{\mathsf{inc}}})$.

Example 7.6.7. After applying the usable terms processor to $\mathcal{DP}(\mathcal{P}_{\mathsf{inc}})$, we obtain:

$$\mathsf{a} \overset{\mathsf{t}}{\hookrightarrow} \{^{1}\mathsf{f}^\sharp(\mathsf{h}^\sharp(\mathsf{g}),\mathsf{g})\} \qquad (7.53)$$

$$\mathsf{g} \overset{\mathsf{t}}{\hookrightarrow} \{^{1/2}\mathsf{b}_1,\ ^{1/2}\mathsf{b}_2\} \qquad (7.54)$$

$$\mathsf{h}(\mathsf{b}_1) \overset{\mathsf{t}}{\hookrightarrow} \{^{1}\mathsf{a}^\sharp\} \qquad (7.55)$$

$$\mathsf{f}(x,\mathsf{b}_2) \overset{\mathsf{t}}{\hookrightarrow} \{^{1}\mathsf{a}^\sharp\} \qquad (7.56)$$

Here, the usable terms processor has been used to replace $\mathsf{f}^\sharp(\mathsf{h}^\sharp(\mathsf{g}^\sharp),\mathsf{g}^\sharp)$ by $\mathsf{f}^\sharp(\mathsf{h}^\sharp(\mathsf{g}),\mathsf{g})$ in the right-hand side of (7.53). The reason is that $\mathsf{g}^\sharp$ is not usable as the $\mathsf{g}$-ADP (7.54) does not contain any annotations.

Next we can apply the rewriting processor on (7.53) repeatedly until all $\mathsf{g}$-symbols are rewritten, and replace it by the ADP

$$\mathsf{a} \overset{\mathsf{t}}{\hookrightarrow} \{^{1/4}\mathsf{f}^\sharp(\mathsf{h}^\sharp(\mathsf{b}_1),\mathsf{b}_1),\ ^{1/4}\mathsf{f}^\sharp(\mathsf{h}^\sharp(\mathsf{b}_2),\mathsf{b}_1),\ ^{1/4}\mathsf{f}^\sharp(\mathsf{h}^\sharp(\mathsf{b}_1),\mathsf{b}_2),\ ^{1/4}\mathsf{f}^\sharp(\mathsf{h}^\sharp(\mathsf{b}_2),\mathsf{b}_2)\}$$

as well as several ADPs $\mathsf{a} \overset{\mathsf{t}}{\hookrightarrow} \ldots$ without annotations in the right-hand side. Now in the subterms $\mathsf{f}^\sharp(\ldots,\mathsf{b}_1)$ and $\mathsf{h}^\sharp(\mathsf{b}_2)$, the annotations are removed from the roots by the usable terms processor, as these subterms cannot rewrite to annotated instances of left-hand sides of ADPs. So the $\mathsf{a}$-ADP is changed to $\mathsf{a} \overset{\mathsf{t}}{\hookrightarrow} \{^{1/4}\mathsf{f}(\mathsf{h}^\sharp(\mathsf{b}_1),\mathsf{b}_1),\ ^{1/4}\mathsf{f}(\mathsf{h}(\mathsf{b}_2),\mathsf{b}_1),\ ^{1/4}\mathsf{f}^\sharp(\mathsf{h}^\sharp(\mathsf{b}_1),\mathsf{b}_2),\ ^{1/4}\mathsf{f}^\sharp(\mathsf{h}(\mathsf{b}_2),\mathsf{b}_2)\}$ (7.53′). This ADP corresponds to the observations that explain why $\mathtt{AST}(\overset{\mathsf{i}}{\to}_{\mathcal{P}_{\mathsf{inc}}})$ holds in Example 7.6.1: we have two terms in the right-hand side that correspond to one $\mathsf{a}^\sharp$ each, both with probability $1/4$, one term that corresponds to a normal form with probability $1/4$, and one that corresponds to two $\mathsf{a}^\sharp$'s with probability $1/4$. So again this corresponds to a random walk where the number of $\mathsf{a}^\sharp$-symbols is decreased by 1 with probability $1/4$, increased by 1 with probability $1/4$, and kept the same with probability $1/2$.

Indeed, we can now use the reduction pair processor with the polynomial interpretation Pol that maps $\mathsf{a}^\sharp$, $\mathsf{f}^\sharp$, and $\mathsf{h}^\sharp$ to 1 and all other symbols to 0, to remove all annotations from the $\mathsf{a}$-ADP (7.53′), because it contains the right-hand side $\mathsf{f}(\mathsf{h}(\mathsf{b}_2), \mathsf{b}_1)$ without annotations and thus, $\mathrm{Pol}^\sharp_{sum}(\mathsf{a}^\sharp) = \mathrm{Pol}(\mathsf{a}^\sharp) = 1 > \mathrm{Pol}^\sharp_{sum}(\mathsf{f}(\mathsf{h}(\mathsf{b}_2), \mathsf{b}_1)) = 0$. Another application of the usable terms processor removes the remaining $\mathsf{a}^\sharp$-annotations from (7.55) and (7.56). Since there are no more annotations left, this proves $\mathtt{AST}(\xrightarrow{\mathsf{i}}_{\mathcal{P}_{\mathsf{inc}}})$.

### 7.6.2 Instantiation Processor

In the non-probabilistic setting [GTSF06], the idea of the *instantiation* processor is to consider all possible *predecessors* $s^\sharp \to t^\sharp$ of a dependency pair $u^\sharp \to v^\sharp$ in a chain and to compute the skeleton $\mathrm{Ren}(\mathrm{Cap}_\mathcal{R}(t^\sharp))$ of $t^\sharp$ (see Section 7.1) that remains unchanged when we reduce $t^\sharp\sigma$ to $u^\sharp\sigma$ for some instantiation $\sigma$, i.e., when going from one DP in a chain to the next. Then $\mathrm{Ren}(\mathrm{Cap}_\mathcal{R}(t^\sharp))$ and $u^\sharp$ must unify with some most general unifier (*mgu*) $\delta$, and $t^\sharp\sigma \to^*_\mathcal{R} u^\sharp\sigma$ implies that $\sigma$ is an instance of $\delta$. Hence, the instantiation processor replaces the DP $u^\sharp \to v^\sharp$ by $u^\sharp\delta \to v^\sharp\delta$.

To adapt the instantiation processor to ADPs and $\mathtt{AST}$, we consider all terms in the multi-distributions on the right-hand sides of all predecessors. In the non-probabilistic DP framework, the instantiation processor only instantiates the DPs, but the rules are left unchanged. Due to the duality of ADPs, similar to the rewriting processor, when instantiating an ADP, we add a copy of the original ADP without any annotations again (i.e., this corresponds to the original non-instantiated rule). In the following, $\mathrm{vr}(\mathcal{A})$ is a variable-renamed copy of $\mathcal{A}$ where all variables are replaced by fresh ones, and we use $\delta(\ell \xhookrightarrow{m} \{p_1 r_1, \dots, p_k r_k\}) = \ell\delta \xhookrightarrow{m} \{p_1 r_1\delta, \dots, p_k r_k\delta\}$ for any substitution $\delta$ and ADP $\ell \xhookrightarrow{m} \{p_1 r_1, \dots, p_k r_k\}$.

Theorem 7.6.8 – Instantiation Processor for $\mathtt{AST}$.
Let $\mathcal{A}$ be an ADP problem with $\mathcal{A} = \mathcal{A}' \uplus \{\ell \xhookrightarrow{m} \mu\}$. Then $\mathrm{Proc}_{\mathtt{ins}}$ is sound for $\mathtt{AST}$, and it is complete for $\mathtt{AST}$ if $\ell \xhookrightarrow{m} \mu$ is $\mathtt{L}$, where $\mathrm{Proc}_{\mathtt{ins}}(\mathcal{A}) = \{\mathcal{A}' \cup M \cup \{\ell \xhookrightarrow{m} \flat(\mu)\}\}$ with

$$M = \left\{ \delta(\ell \xhookrightarrow{m} \mu) \;\middle|\; \begin{array}{c} \ell' \xhookrightarrow{m'} \nu \in \mathrm{vr}(\mathcal{A}), r' \in \mathrm{Supp}(\nu), \\ t \trianglelefteq^\sharp r', \delta = mgu(\mathrm{Ren}(\mathrm{Cap}_\mathcal{A}(t^\sharp)), \ell^\sharp) \end{array} \right\}$$

For innermost rewriting, as in the estimation of the dependency graph, we do not need Ren, and only consider those mgus $\delta$ where the instantiated left-hand sides are in argument normal form.

Theorem 7.6.9 – Instantiation Processor for $\mathtt{iAST}$.
Let $\mathcal{A}$ be an ADP problem with $\mathcal{A} = \mathcal{A}' \uplus \{\ell \xhookrightarrow{m} \mu\}$. Then $\mathrm{Proc}^{\mathbf{i}}_{\mathtt{ins}}$ is sound and complete for $\mathtt{iAST}$, where $\mathrm{Proc}^{\mathbf{i}}_{\mathtt{ins}}(\mathcal{A}) = \{\mathcal{A}' \cup M \cup \{\ell \xhookrightarrow{m} \flat(\mu)\}\}$ with

$$M = \left\{ \delta(\ell \xhookrightarrow{m} \mu) \;\middle|\; \begin{array}{c} \ell' \xhookrightarrow{m'} \nu \in \mathrm{vr}(\mathcal{A}), r' \in \mathrm{Supp}(\nu), \\ t \trianglelefteq^\sharp r', \delta = mgu(\mathrm{Cap}_\mathcal{A}(t^\sharp), \ell^\sharp), \{\ell'\delta, \ell\delta\} \subseteq \mathtt{ANF}_\mathcal{A} \end{array} \right\}$$

For the proofs of the instantiation processors, we first show some basic properties of $\mathrm{Cap}_\mathcal{A}$ and Ren, similar to the known properties of the non-probabilistic versions of $\mathrm{Cap}_\mathcal{A}$ and Ren.

Lemma 7.6.10 – Properties of $\mathrm{Cap}_{\mathcal{A}}$ and Ren.

*Let $t, s \in \mathcal{T}^\sharp$.*

1. *If $t\sigma \xrightarrow{\mathrm{i}}{}^*_{\mathrm{np}(\mathcal{A})} s$ for some substitution $\sigma$ that maps each variable to a normal form, i.e., $\sigma(x) \in \mathtt{NF}_{\mathcal{A}}$ for all $x \in \mathcal{V}$, then $s = \mathrm{Cap}_{\mathcal{A}}(t)\delta$ for some substitution $\delta$ which only differs from $\sigma$ on the fresh variables that are introduced by $\mathrm{Cap}_{\mathcal{A}}$.*
2. *If $t\sigma \to^*_{\mathrm{np}(\mathcal{A})} s$ for some substitution $\sigma$, then $s = \mathrm{Ren}(\mathrm{Cap}_{\mathcal{A}}(t))\delta$ for some substitution $\delta$ which only differs from $\sigma$ on the fresh variables that are introduced by* Ren.

*Proof.* We prove both statements individually. *Proof*

1. Let $t\sigma \xrightarrow{\mathrm{i}}_{\mathrm{np}(\mathcal{A}),\rho_1} s_1 \xrightarrow{\mathrm{i}}_{\mathrm{np}(\mathcal{A}),\rho_2} \cdots \xrightarrow{\mathrm{i}}_{\mathrm{np}(\mathcal{A}),\rho_n} s_n = s$, where $\xrightarrow{\mathrm{i}}_{\mathrm{np}(\mathcal{A}),\rho}$ again denotes a rewrite step at position $\rho$. Let $\{\pi_1, \ldots, \pi_m\}$ be the set of positions where $\mathrm{Cap}_{\mathcal{A}}$ replaces the subterms of $t$ by corresponding fresh variables $x_1, \ldots, x_m$. Note that the rewrite steps cannot be "completely inside" the used substitution $\sigma$, as it maps every variable to a normal form. Hence, by the definition of $\mathrm{Cap}_{\mathcal{A}}$, for each $\rho_i$ there is a higher position $\pi_j \leq \rho_i$. Thus, $s$ can at most differ from $t\sigma$ on positions below a $\pi_j$. We define $\delta$ to be like $\sigma$ but on the fresh variables $x_1, \ldots, x_m$ we define $\delta(x_j) = s|_{\pi_j}$. Then by construction $\mathrm{Cap}_{\mathcal{A}}(t)\delta = s$ and $\delta$ and $\sigma$ differ only on the fresh variables.
2. Let $t\sigma \to_{\mathrm{np}(\mathcal{A}),\rho_1} s_1 \to_{\mathrm{np}(\mathcal{A}),\rho_2} \cdots \to_{\mathrm{np}(\mathcal{A}),\rho_n} s_n = s$. Let $\{\pi_1, \ldots, \pi_m\}$ be the set of positions where $\mathrm{Ren} \circ \mathrm{Cap}_{\mathcal{A}}$ replaces the subterms of $t$ by corresponding fresh variables $x_1, \ldots, x_m$. By the definition of Ren and $\mathrm{Cap}_{\mathcal{A}}$, for each $\rho_i$ there is a higher position $\pi_j \leq \rho_i$. (If the rewrite step is "completely inside" the substitution $\sigma$, then Ren replaces the corresponding variable in $t$ by a fresh one, otherwise $\mathrm{Cap}_{\mathcal{A}}$ replaces the subterm in $t$ by a new variable, which is again replaced by Ren afterwards.) Hence, $s$ can at most differ from $t\sigma$ on positions below a $\pi_j$. We define $\delta$ to be like $\sigma$ but on the fresh variables $x_1, \ldots, x_m$ we define $\delta(x_j) = s|_{\pi_j}$. Then by construction $\mathrm{Ren}(\mathrm{Cap}_{\mathcal{A}}(t))\delta = s$, and $\delta$ and $\sigma$ differ only on the fresh variables. *QED*

The proof of the second statement only uses that every subterm whose root symbol is the root of a left-hand side of the used rules is replaced by a fresh variable. Hence, the second statement also holds for an arbitrary TRS $\mathcal{Q}$ instead of $\mathrm{np}(\mathcal{A})$ (whose rules may violate the variable conditions) and every function that replaces at least all subterms whose root is a defined symbol of $\mathcal{Q}$ by fresh variables (and the whole term if $\mathcal{Q}$ has a rule whose left-hand side is a variable). We use this for reversed TRSs in Section 7.6.3.

*Proof of both Theorem 7.6.8 and Theorem 7.6.9.* We prove the statement for $\mathtt{AST}$. The argument for $\mathtt{iAST}$ is identical just considering innermost steps and always indicated in parentheses if something differs. As in the proof of Theorem 7.6.3, let $\mu = \{{}^{p_1}r_1, \ldots, {}^{p_k}r_k\}$, $\overline{\mathcal{A}'} = \mathcal{A}' \cup M \cup \{\ell \xhookrightarrow{m} \{{}^{p_1}\flat(r_1), \ldots, {}^{p_k}\flat(r_k)\}\}$, and $\overline{\mathcal{A}} = \overline{\mathcal{A}'} \cup \{\ell \xhookrightarrow{m} \{{}^{p_1}r_1, \ldots, {}^{p_k}r_k\}\}$. First, note that $\mathtt{ANF}_{\mathcal{A}} = \mathtt{ANF}_{\overline{\mathcal{A}}} = \mathtt{ANF}_{\overline{\mathcal{A}'}}$, since the left-hand sides in $\overline{\mathcal{A}}$ and $\overline{\mathcal{A}'}$ are either already from rules in $\mathcal{A}$ or instantiated left-hand sides from $\mathcal{A}$. So it suffices to consider only $\mathtt{ANF}_{\mathcal{A}}$. Second, assume that there exists an $\overline{\mathcal{A}}$-CT $\mathfrak{T}$ that converges with probability $< 1$ whose root is labeled with ${}^1t$ and $\flat(t|_{\pi_0}) = s\theta$ for a substitution $\theta$ and an ADP $s \xhookrightarrow{m'} \nu \in \overline{\mathcal{A}}$, and $\mathrm{pos}_{\mathcal{D}^\sharp}(t) = \{\pi_0\}$. If in addition, the ADP $\ell \xhookrightarrow{m} \{{}^{p_1}r_1, \ldots, {}^{p_k}r_k\}$ may only be used at the root, then we know *Proof*

that not only $\overline{\mathcal{A}}$ is not AST but also that $\overline{\mathcal{A}'}$ is not AST. The reason is that if only the root $v$ uses the ADP $\ell \xhookrightarrow{m} \{{}^{p_1}r_1, \dots, {}^{p_k}r_k\}$ then all the subtrees starting at one of its direct successors $vE = \{w_1, \dots, w_k\}$ are $\overline{\mathcal{A}'}$-CTs. Furthermore, since $\mathfrak{T}$ converges with probability $< 1$, there must be at least one subtree $\mathfrak{T}[w_i E^*]$ that starts at the node $w_i$ for some $1 \leq i \leq k$ and also converges with probability $< 1$.

▶ *Proof (Soundness)*: Let $\mathcal{A}$ not be (i)AST. Then by Lemma 7.3.3 there exists an (innermost) $\mathcal{A}$-CT $\mathfrak{T} = (N, E)$ that converges with probability $< 1$ that starts with $t$ and $\flat(t|_{\pi_0}) = s\theta$ for a substitution $\theta$ and an ADP $s \xhookrightarrow{m'} \nu \in \mathcal{A}$, and $\mathrm{pos}_{\mathcal{D}^\sharp}(t) = \{\pi_0\}$, and whose root performs its rewrite step at the position $\pi_0$. We will now create an (innermost) $\overline{\mathcal{A}}$-CT $\mathfrak{T}' = (N, E)$, with the same underlying tree structure, such that $p_v^{\mathfrak{T}} = p_v^{\mathfrak{T}'}$ for all $v \in N$. Furthermore, we will at most use the ADP $\ell \xhookrightarrow{m} \{{}^{p_1}r_1, \dots, {}^{p_k}r_k\}$ at the root. Since the tree structure and the probabilities are the same, we then get $|\mathfrak{T}'| = |\mathfrak{T}| < 1$, and hence, using our previous discussion, $\overline{\mathcal{A}'}$ is not (i)AST either.

The core idea is that every rewrite step with $\ell \xhookrightarrow{m} \{{}^{p_1}r_1, \dots, {}^{p_k}r_k\}$ at a node $v$ that is not the root can also be done with a rule from $M$, or we can use $\ell \xhookrightarrow{m} \{{}^{p_1}\flat(r_1), \dots, {}^{p_k}\flat(r_k)\}$ if the annotations do not matter, e.g., we rewrite at a position that is not annotated.

We construct the terms $t'_v$ for the nodes of the $\overline{\mathcal{A}}$-CT $\mathfrak{T}'$ inductively such that for all nodes $v \in N \setminus \mathrm{Leaf}$ with children nodes $vE = \{w_1, \dots, w_h\}$ we have $t'_v \hookrightarrow_{\overline{\mathcal{A}}} \{{}^{p_{w_1}/p_v}t'_{w_1}, \dots, {}^{p_{w_h}/p_v}t'_{w_h}\}$ and for all non-root nodes in $\Gamma$ we even have $t'_v \hookrightarrow_{\overline{\mathcal{A}'}} \{{}^{p_{w_1}/p_v}t'_{w_1}, \dots, {}^{p_{w_h}/p_v}t'_{w_h}\}$. Let $W \subseteq N$ be the set of nodes $v$ where we have already defined the corresponding term $t'_v$. For the construction, we track the annotations along the paths of $\mathfrak{T}$. Consider an inner node $u$ that rewrites $t_u$ at position $\pi$ using an ADP $\hat{\ell} \xhookrightarrow{\hat{m}} \{{}^{\hat{p}_1}\hat{r}_1, \dots, {}^{\hat{p}_h}\hat{r}_h\}$ and a VRF $(\varphi_j)_{1 \leq j \leq h}$, and let $w_j$ be its $j$-th child. An annotated position $\tau \in \mathrm{pos}_{\mathcal{D}^\sharp}(t_u)$ has the *successor* $\tau$ in $t_{w_j}$ if $\tau$ is orthogonal to $\pi$, or if $\tau$ is strictly above $\pi$ and $\hat{m} = \mathtt{t}$. It has the successor $\pi.\varphi_j(\rho).\kappa$ in $t_{w_j}$ if $\tau = \pi.\rho.\kappa$ for some $\rho \in \mathrm{pos}_{\mathcal{V}}(\hat{\ell})$ with $\varphi_j(\rho) \neq \bot$ (cf. the set $\Psi_j$ in Definition 7.2.11). Otherwise, $\tau$ has no successor in $t_{w_j}$. An annotated position $\tau \in \mathrm{pos}_{\mathcal{D}^\sharp}(t_v)$ is *used below* $v$ if there are a path $v = u_0, \dots, u_n$ in $\mathfrak{T}$ and positions $\tau = \tau_0, \dots, \tau_n$ such that $\tau_{i+1}$ is a successor of $\tau_i$ in $t_{u_{i+1}}$ for all $0 \leq i < n$, and $u_n$ performs a step of the form **at** or **af** at position $\tau_n$. Let $\mathrm{Unused}(v) \subseteq \mathrm{pos}_{\mathcal{D}^\sharp}(t_v)$ be the set of annotated positions of $t_v$ that are not used below $v$. If $v$ performs a step of the form **at** or **af** at position $\pi$, then $\pi \notin \mathrm{Unused}(v)$. Moreover, every $\tau' \in \mathrm{pos}_{\mathcal{D}^\sharp}(t_{w_j}) \setminus \mathrm{Unused}(w_j)$ is a successor of some $\tau \in \mathrm{pos}_{\mathcal{D}^\sharp}(t_u) \setminus \mathrm{Unused}(u)$, or $u$ performs a step of the form **at** or **af** and $\tau' = \pi.\chi$ for some $\chi \in \mathrm{pos}_{\mathcal{D}^\sharp}(\hat{r}_j)$. During our construction, we ensure that for every node $v \in W$ we have

$$t_v \overset{\flat}{=} t'_v \text{ and } \mathrm{pos}_{\mathcal{D}^\sharp}(t_v) \setminus \mathrm{Unused}(v) \subseteq \mathrm{pos}_{\mathcal{D}^\sharp}(t'_v). \tag{7.57}$$

This means that the corresponding term $t_v$ for the node $v$ in $\mathfrak{T}$ has the same structure as the term $t'_v$ in $\mathfrak{T}'$, and additionally, every annotation in $t_v$ that is used later on also exists in $t'_v$. The second condition ensures that if we rewrite using Case **af** or **at** in $\mathfrak{T}$, we do the same in $\mathfrak{T}'$, i.e., the corresponding node $v$ remains in $\Gamma$ in $\mathfrak{T}'$, so that $\mathfrak{T}'$ satisfies the liveness property. Moreover, if we perform the step at a node $u$ in $\mathfrak{T}'$ at the same position such that every $\tau \in \mathrm{pos}_{\mathcal{D}^\sharp}(t_u) \setminus \mathrm{Unused}(u)$ has the same successors in $\mathfrak{T}'$ as in $\mathfrak{T}$, and such that every annotation $\pi.\chi$ with $\chi \in \mathrm{pos}_{\mathcal{D}^\sharp}(\hat{r}_j)$ that is created in $\mathfrak{T}$ and used below $w_j$ is also created in $\mathfrak{T}'$, then (7.57) holds again for all children $w_j$ of $u$. We label the root of $\mathfrak{T}'$ by $t$. Here, (7.57) obviously holds. As long as there

is still an inner node $v \in W$ such that its successors are not contained in $W$, we do the following. Let $vE = \{w_1, \ldots, w_h\}$ be the set of its successors. We need to define the corresponding terms $t'_{w_1}, \ldots, t'_{w_h}$ for the nodes $w_1, \ldots, w_h$. Since $v$ is not a leaf and $\mathfrak{T}$ is an $\mathcal{A}$-CT, we have $t_v \hookrightarrow_{\mathcal{A}} \{{}^{p_{w_1}/p_v}t_{w_1}, \ldots, {}^{p_{w_h}/p_v}t_{w_h}\}$. We have the following three different cases:

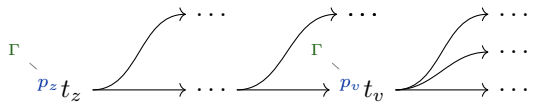
**(A) We have** $t_v \hookrightarrow_{\mathcal{A}} \{{}^{p_{w_1}/p_v}t_{w_1}, \ldots, {}^{p_{w_h}/p_v}t_{w_h}\}$ with an ADP $\ell' \xhookrightarrow{m'} \{{}^{p_1}r'_1, \ldots, {}^{p_h}r'_h\}$ that is either different from $\ell \xhookrightarrow{m} \{{}^{p_1}r_1, \ldots, {}^{p_k}r_k\}$ or the node $v$ is the root, using the position $\pi$, a VRF $(\varphi_j)_{1 \le j \le h}$, and a substitution $\sigma$ such that $\flat(t_v|_\pi) = \ell'\sigma$. Then, $t_v \overset{\flat}{=}_{(IH)} t'_v$, and hence also $\flat(t'_v|_\pi) = \ell'\sigma$. Thus, we can rewrite the term $t'_v$ using the same ADP, the same position, the same substitution, and the same VRF. This means that we have $t'_v \hookrightarrow_{\overline{\mathcal{A}}} \{{}^{p_{w_1}/p_v}t'_{w_1}, \ldots, {}^{p_{w_h}/p_v}t'_{w_h}\}$. If the step in $\mathfrak{T}$ is of the form **af** or **at**, then $\pi \notin \mathrm{Unused}(v)$, so $\pi \in \mathrm{pos}_{\mathcal{D}^\sharp}(t'_v)$ by (7.57), i.e., the step in $\mathfrak{T}'$ is of the same form and creates the same annotations. Since we use the same ADP, the same position, and the same VRF, every annotated position has the same successors in $\mathfrak{T}$ and $\mathfrak{T}'$. Therefore, (7.57) is again satisfied for all children $w_j$ of $v$.

**(B) We have** $t_v \hookrightarrow_{\mathcal{A}} \{{}^{p_{w_1}/p_v}t_{w_1}, \ldots, {}^{p_{w_k}/p_v}t_{w_k}\}$ using the ADP $\ell \xhookrightarrow{m} \{{}^{p_1}r_1, \ldots, {}^{p_k}r_k\}$, the position $\pi$, a VRF $(\varphi_j)_{1 \le j \le k}$, and a substitution $\sigma$ such that $\flat(t_v|_\pi) = \ell\sigma$, and $\pi \notin \mathrm{pos}_{\mathcal{D}^\sharp}(t_v)$. Since $t_v \overset{\flat}{=}_{(IH)} t'_v$, we can rewrite the term $t'_v$ using the ADP $\ell \xhookrightarrow{m} \{{}^{p_1}\flat(r_1), \ldots, {}^{p_k}\flat(r_k)\}$, the same position, the same substitution, and the same VRF. This means that we have $t'_v \hookrightarrow_{\overline{\mathcal{A}}} \{{}^{p_{w_1}/p_v}t'_{w_1}, \ldots, {}^{p_{w_k}/p_v}t'_{w_k}\}$ and (7.57) is again satisfied. In order to prove this, one can do a similar analysis as above. Note that only cases **nf** and **nt** can be applied in $\mathfrak{T}$ (since $\pi \notin \mathrm{pos}_{\mathcal{D}^\sharp}(t_v)$), so the step in $\mathfrak{T}$ creates no annotations of the terms $r_j$, and the used VRF yields the same successors in $\mathfrak{T}$ and $\mathfrak{T}'$.

**(C) Finally, we have** $t_v \hookrightarrow_{\mathcal{A}} \{{}^{p_{w_1}/p_v}t_{w_1}, \ldots, {}^{p_{w_k}/p_v}t_{w_k}\}$ using the ADP $\ell \xhookrightarrow{m} \{{}^{p_1}r_1, \ldots, {}^{p_k}r_k\}$, the position $\pi$, a VRF $(\varphi_j)_{1 \le j \le k}$, and a substitution $\sigma$ such that $\flat(t_v|_\pi) = \ell\sigma$, $\pi \in \mathrm{pos}_{\mathcal{D}^\sharp}(t_v)$, and $v$ is not the root. Then $t_{w_j} = \flat^{\uparrow}_{\pi}(t_v[\sharp_{\mathrm{pos}_{\mathcal{D}^\sharp}(r_j) \cup \Psi_j}(r_j\sigma)]_\pi)$ if $m = \mathtt{f}$ and $t_{w_j} = t_v[\sharp_{\mathrm{pos}_{\mathcal{D}^\sharp}(r_j) \cup \Psi_j}(r_j\sigma)]_\pi$, otherwise.

We now look at the (not necessarily direct) predecessor $z$ of $v$ that is in $\Gamma$, where the ADP is applied that is used to create the annotation of the function symbol at position $\pi$. There is always such a node $z$. The reason is that the root of $\mathfrak{T}$ performs its step at the only annotated position $\pi_0$ of $t$, so for every non-root node $v$, every annotated position of $t_v$ results (via successors) from an annotation that was created by a right-hand side at an earlier node in $\Gamma$. Furthermore, we only use rules with the flag $m = \mathtt{t}$ to evaluate the arguments of the annotated function symbol as otherwise, the position $\pi$ would not be annotated in $t_v$. We show that the path from $z$ to $v$ can also be taken when using one of the new instantiations of $\ell \xhookrightarrow{m} \{{}^{p_1}r_1, \ldots, {}^{p_k}r_k\}$ instead.

Let this predecessor $z \in \Gamma$ use the ADP $\ell' \xhookrightarrow{m'} \{{}^{p'_1}r'_1, \ldots, {}^{p'_h}r'_h\}$, the position $\pi'$, and the substitution $\sigma'$. Furthermore, let $1 \le j \le h$ and $\tau$ be the position of $r'_j$ such that $\sharp_{\{\varepsilon\}}(r'_j|_\tau)\sigma' \to^*_{\mathrm{np}(\mathcal{A})} \sharp_{\{\varepsilon\}}(t_v|_\pi) = \sharp_{\{\varepsilon\}}(\ell)\sigma$, which exists by the previous paragraph.




The following consideration now differs for `iAST` and `AST` due to the different approximations (`iAST` uses only $\mathrm{Cap}_{\mathcal{A}}$ and `AST` uses $\mathrm{Ren} \circ \mathrm{Cap}_{\mathcal{A}}$).

**AST:** From $\sharp_{\{\varepsilon\}}(r'_j|_\tau)\sigma' \to^*_{\mathrm{np}(\mathcal{A})} \sharp_{\{\varepsilon\}}(\ell)\sigma$ together with Lemma 7.6.10 we get $\sharp_{\{\varepsilon\}}(\ell)\sigma = \mathrm{Ren}(\mathrm{Cap}_{\mathcal{A}}(\sharp_{\{\varepsilon\}}(r'_j|_\tau)))\delta$ for some substitution $\delta$ that differs from $\sigma'$ at most on the variables that are introduced by Ren. W.l.o.g., we can assume that $\sigma'$ is equal to $\delta$ on all these fresh variables, and since the ADPs are variable-renamed, we can also assume that $\sigma$ is equal to $\delta$ on all the fresh variables and all the variables from $\ell'$. Hence, $\sharp_{\{\varepsilon\}}(\ell)\delta = \mathrm{Ren}(\mathrm{Cap}_{\mathcal{A}}(\sharp_{\{\varepsilon\}}(r'_j|_\tau)))\delta$ implies that we can find an mgu $\gamma$ of $\sharp_{\{\varepsilon\}}(\ell)$ and $\mathrm{Ren}(\mathrm{Cap}_{\mathcal{A}}(\sharp_{\{\varepsilon\}}(r'_j|_\tau)))$ with $\sigma = \gamma\zeta$ for some substitution $\zeta$. Hence, we can apply the new ADP $\ell\gamma \xhookrightarrow{m} \{p_1 r_1\gamma, \dots, p_k r_k\gamma\} \in M$ with the position $\pi$ and the substitution $\zeta$. As VRF, we use $(\varphi'_j)_{1\le j\le k}$ with $\varphi'_j(\rho.\rho') = \varphi_j(\rho).\rho'$ for every $\rho \in \mathrm{pos}_{\mathcal{V}}(\ell)$ with $\varphi_j(\rho) \neq \bot$ and every $\rho' \in \mathrm{pos}_{\mathcal{V}}(\gamma(\ell|_\rho))$, and $\varphi'_j(\rho.\rho') = \bot$ if $\varphi_j(\rho) = \bot$. This is indeed a VRF for the new ADP, since $r_j\gamma|_{\varphi_j(\rho).\rho'} = \gamma(\ell|_\rho)|_{\rho'} = \ell\gamma|_{\rho.\rho'}$. This means that we have $t'_v \hookrightarrow_{\overline{\mathcal{A}'}} \{p_{w_1}/p_v\, t'_{w_1}, \dots, p_{w_k}/p_v\, t'_{w_k}\}$ with $t'_{w_j} = t'_v[\sharp_{\mathrm{pos}_{\mathcal{D}^\sharp}(r_j)\cup\Psi'_j}(r_j\gamma\zeta)]_\pi$ if $m = \mathtt{t}$, or $t'_{w_j} = \flat^{\uparrow}_{\pi}(t'_v[\sharp_{\mathrm{pos}_{\mathcal{D}^\sharp}(r_j)\cup\Psi'_j}(r_j\gamma\zeta)]_\pi)$, otherwise, where $\Psi'_j$ is defined w.r.t. $(\varphi'_j)_{1\le j\le k}$. Since $\sigma = \gamma\zeta$, we get $t_{w_j} \overset{\flat}{=} t'_{w_j}$, the annotations of $r_j$ are created in both $\mathfrak{T}$ and $\mathfrak{T}'$, and every annotated position of $t_v$ below a position $\pi.\rho.\rho'$ with $\rho.\rho' \in \mathrm{pos}_{\mathcal{V}}(\ell\gamma)$ has the same successors in $\mathfrak{T}$ and $\mathfrak{T}'$. The remaining annotated positions of $t_v$ below a variable position $\pi.\rho$ with $\rho \in \mathrm{pos}_{\mathcal{V}}(\ell)$ are at non-variable positions of $\gamma(\ell|_\rho)$. Since the term $\mathrm{Ren}(\mathrm{Cap}_{\mathcal{A}}(\sharp_{\{\varepsilon\}}(r'_j|_\tau)))$ is linear and does not share variables with $\sharp_{\{\varepsilon\}}(\ell)$, the unification can only instantiate a variable $x$ of $\ell$ by (an instance of) a subterm of $\mathrm{Ren}(\mathrm{Cap}_{\mathcal{A}}(\sharp_{\{\varepsilon\}}(r'_j|_\tau)))$ at a position of $x$ in $\ell$, where the variables of this subterm can again only be instantiated by such subterms. So the function symbols of $\gamma(\ell|_\rho)$ are function symbols of $\mathrm{Ren}(\mathrm{Cap}_{\mathcal{A}}(\sharp_{\{\varepsilon\}}(r'_j|_\tau)))$ below its root, i.e., they are not the root symbol of any left-hand side of $\mathcal{A}$. Hence, the remaining annotated positions and all their successors can never be used for a step of the form **at** or **af**, i.e., they are in $\mathrm{Unused}(v)$. Thus, (7.57) is satisfied again.

**iAST:** From $\sharp_{\{\varepsilon\}}(r'_j|_\tau)\sigma' \xrightarrow{\mathtt{i}}{}^*_{\mathrm{np}(\mathcal{A})} \sharp_{\{\varepsilon\}}(\ell)\sigma$ and the fact that $\sigma'$ maps every variable to a normal form due to innermost rewriting, Lemma 7.6.10 implies $\sharp_{\{\varepsilon\}}(\ell)\sigma = \mathrm{Cap}_{\mathcal{A}}(\sharp_{\{\varepsilon\}}(r'_j|_\tau))\delta$ for some substitution $\delta$ that differs from $\sigma'$ at most on the variables that are introduced by $\mathrm{Cap}_{\mathcal{A}}$. W.l.o.g., we can assume that $\sigma'$ is equal to $\delta$ on all these fresh variables, and since the ADPs are variable-renamed, we can also assume that $\sigma$ is equal to $\delta$ on all the fresh variables and all the variables from $\ell'$. Hence, $\sharp_{\{\varepsilon\}}(\ell)\delta = \mathrm{Cap}_{\mathcal{A}}(\sharp_{\{\varepsilon\}}(r'_j|_\tau))\delta$ shows there is an mgu $\gamma$ of $\sharp_{\{\varepsilon\}}(\ell)$ and $\mathrm{Cap}_{\mathcal{A}}(\sharp_{\{\varepsilon\}}(r'_j|_\tau))$ with $\sigma = \gamma\zeta$ for some substitution $\zeta$. Moreover, the property $\{\ell'\sigma', \ell\sigma\} \subseteq \mathtt{ANF}_{\mathcal{A}}$ must remain true when replacing $\sigma$ and $\sigma'$ by the more general substitution $\gamma$, i.e., $\{\ell'\gamma, \ell\gamma\} \subseteq \mathtt{ANF}_{\mathcal{A}}$. The rest is analogous to the AST case, where all VRFs map every position to $\bot$, so that no annotations below $\pi$ are kept in $\mathfrak{T}$ and $\mathfrak{T}'$.

▶ *Proof (Completeness)*: Let $\overline{\mathcal{A}'}$ not be (i)AST. Then there exists an (innermost) $\overline{\mathcal{A}'}$-CT $\mathfrak{T} = (N, E)$ that converges with probability $< 1$. We will now create an (innermost) $\mathcal{A}$-CT $\mathfrak{T}' = (N, E)$, with the same underlying tree structure, such that $p_v^{\mathfrak{T}} = p_v^{\mathfrak{T}'}$ for all $v \in N$. Since the tree structure and the probabilities are the same, we then get $|\mathfrak{T}'| = |\mathfrak{T}|$, and thus $\mathcal{A}$ is not (i)AST either.

The core idea of this construction is that every rewrite step with an ADP from $M$ or the ADP $\ell \xhookrightarrow{m} \{p_1\flat(r_1), \dots, p_k\flat(r_k)\}$ is also possible with the more general ADP $\ell \xhookrightarrow{m} \{p_1 r_1, \dots, p_k r_k\}$ that may also contain more annotations. As for soundness, we

focus on AST, but the proof is analogous for $\mathtt{AST}(\xhookrightarrow{\mathrm{i}}_{\mathcal{A}})$. We construct the terms $t'_v$ for the nodes of the $\mathcal{A}$-CT $\mathfrak{T}'$ inductively such that for all inner nodes $v \in N \setminus \mathrm{Leaf}$ with children nodes $vE = \{w_1, \dots, w_h\}$ we have $t'_v \hookrightarrow_{\mathcal{A}} \{{}^{p_{w_1}/p_v}t'_{w_1}, \dots, {}^{p_{w_h}/p_v}t'_{w_h}\}$. Let $W \subseteq N$ be the set of nodes $v$ where we have already defined the corresponding term $t'_v$. During our construction, we ensure that for every $v \in W$ (analogous to the soundness construction) we have

$$t_v \overset{\flat}{=} t'_v \text{ and } \mathrm{pos}_{\mathcal{D}^\sharp}(t_v) \subseteq \mathrm{pos}_{\mathcal{D}^\sharp}(t'_v). \tag{7.58}$$

We label the root of $\mathfrak{T}'$ exactly as the root of $\mathfrak{T}$. Here, (7.58) obviously holds. As long as there is still an inner node $v \in W$ such that its successors are not contained in $W$, we do the following. Let $vE = \{w_1, \dots, w_h\}$ be the set of its successors. We need to define the corresponding terms $t'_{w_1}, \dots, t'_{w_h}$ for the nodes $w_1, \dots, w_h$. Since $v$ is not a leaf and $\mathfrak{T}$ is an $\overline{\mathcal{A}'}$-CT, we have $t_v \hookrightarrow_{\overline{\mathcal{A}'}} \{{}^{p_{w_1}/p_v}t_{w_1}, \dots, {}^{p_{w_h}/p_v}t_{w_h}\}$. We have the following three different cases:

(A) If it is a step with $\hookrightarrow_{\overline{\mathcal{A}'}}$ using an ADP from $\mathcal{A}'$ in $\mathfrak{T}$, then we perform a rewrite step with the same ADP, the same position, the same substitution, and the same VRF in $\mathfrak{T}'$. This is analogous to Case (A) of the soundness proof.

(B) If it is a step with $\hookrightarrow_{\overline{\mathcal{A}'}}$ using the ADP $\ell \xhookrightarrow{m} \{{}^{p_1}\flat(r_1), \dots, {}^{p_k}\flat(r_k)\}$ in $\mathfrak{T}$, then we use the ADP $\ell \xhookrightarrow{m} \{{}^{p_1}r_1, \dots, {}^{p_k}r_k\}$ that contains more annotations in $\mathfrak{T}'$, at the same position, using the same substitution, and the same VRF. Since we use the same rule but with more annotations, we end up with $t_{w_j} \overset{\flat}{=} t'_{w_j}$ and $\mathrm{pos}_{\mathcal{D}^\sharp}(t_{w_j}) \subseteq \mathrm{pos}_{\mathcal{D}^\sharp}(t'_{w_j})$.

(C) If it is a step with $\hookrightarrow_{\overline{\mathcal{A}'}}$ using an ADP $\ell\gamma \xhookrightarrow{m} \{{}^{p_1}r_1\gamma, \dots, {}^{p_k}r_k\gamma\}$ from $M$ in $\mathfrak{T}$, a VRF $(\varphi'_j)_{1\le j\le k}$, and a substitution $\delta$, then we use the more general ADP $\ell \xhookrightarrow{m} \{{}^{p_1}r_1, \dots, {}^{p_k}r_k\}$ in $\mathfrak{T}'$, at the same position, with the substitution $\gamma\delta$. For iAST, all VRFs map every position to $\bot$. For AST, we use that $\ell \xhookrightarrow{m} \mu$ is L. Then $\gamma$ is an mgu of two linear terms without common variables, hence $\ell\gamma$ is linear and the terms $\gamma(x)$ for the variables $x$ of $\ell$ are pairwise variable-disjoint. So for every $\rho \in \mathrm{pos}_{\mathcal{V}}(\ell)$ and $\rho' \in \mathrm{pos}_{\mathcal{V}}(\gamma(\ell|_\rho))$, the variable $\ell\gamma|_{\rho.\rho'}$ occurs in $r_j\gamma$ only at the position $\kappa.\rho'$, where $\kappa$ is the unique position of $\ell|_\rho$ in $r_j$ (if $\ell|_\rho \in \mathcal{V}(r_j)$). Thus, $\varphi'_j(\rho.\rho') \in \{\kappa.\rho', \bot\}$, and we use the VRF $(\varphi_j)_{1\le j\le k}$ with $\varphi_j(\rho) = \kappa$ (and $\varphi_j(\rho) = \bot$ if $\ell|_\rho \notin \mathcal{V}(r_j)$). Then every annotation that is kept in the step of $\mathfrak{T}$ is also kept in the step of $\mathfrak{T}'$, so that $t_{w_j} \overset{\flat}{=} t'_{w_j}$ and $\mathrm{pos}_{\mathcal{D}^\sharp}(t_{w_j}) \subseteq \mathrm{pos}_{\mathcal{D}^\sharp}(t'_{w_j})$ hold. *QED*

The following counterexample shows that the requirement L is needed for the completeness of $\mathrm{Proc}_{\mathtt{ins}}$ w.r.t. AST.

Counterexample 7.6.11 (Incompleteness of the Instantiation Processor for AST). Consider the ADP problem $\mathcal{A}$ with the ADPs

$$\mathsf{h} \xhookrightarrow{\mathrm{f}} \{{}^{2/5}\mathsf{f}^\sharp(\mathsf{g}(\mathsf{h}^\sharp, \mathsf{h}^\sharp), \mathsf{g}(\mathsf{h}, \mathsf{h})), {}^{3/5}\mathsf{d}\} \tag{7.59}$$

$$\mathsf{h} \xhookrightarrow{\mathrm{t}} \{{}^{1}\mathsf{h}\} \tag{7.60}$$

$$\mathsf{f}(x, x) \xhookrightarrow{\mathrm{f}} \{{}^{1}\mathsf{c}^\sharp(x, x)\} \tag{7.61}$$

$$\mathsf{c}(\mathsf{g}(y, \mathsf{h}), \mathsf{g}(\mathsf{h}, z)) \xhookrightarrow{\mathrm{f}} \{{}^{1}\mathsf{k}(y, z, \mathsf{h}^\sharp)\} \tag{7.62}$$

where $\mathcal{D} = \{\mathsf{h}, \mathsf{f}, \mathsf{c}\}$ and $\mathcal{C} = \{\mathsf{g}, \mathsf{k}, \mathsf{d}\}$. The ADP (7.61) is not L. As $\mathrm{Cap}_{\mathcal{A}}$ replaces

every $\mathsf{h}$ by a fresh variable, $\mathrm{Proc}_{\mathtt{ins}}$ replaces (7.61) by $\mathsf{f}(x,x) \overset{\mathsf{f}}{\hookrightarrow} \{^1\mathsf{c}(x,x)\}$ and

$$\mathsf{f}(\mathsf{g}(v_1,v_2),\mathsf{g}(v_1,v_2)) \overset{\mathsf{f}}{\hookrightarrow} \{^1\mathsf{c}^\sharp(\mathsf{g}(v_1,v_2),\mathsf{g}(v_1,v_2))\} \tag{7.63}$$

With (7.59), $\mathsf{h}^\sharp$ rewrites to $\mathsf{f}^\sharp(\mathsf{g}(\mathsf{h}^\sharp,\mathsf{h}^\sharp),\mathsf{g}(\mathsf{h},\mathsf{h}))$ with probability $2/5$. Now a VRF of (7.63) can distribute the two annotations of $\mathsf{h}^\sharp$ to the two copies, which yields $\mathsf{c}^\sharp(\mathsf{g}(\mathsf{h}^\sharp,\mathsf{h}),\mathsf{g}(\mathsf{h},\mathsf{h}^\sharp))$, and (7.62) then yields $\mathsf{k}(\mathsf{h}^\sharp,\mathsf{h}^\sharp,\mathsf{h}^\sharp)$. So in the resulting ADP problem, $\mathsf{h}^\sharp$ is replaced by three copies of $\mathsf{h}^\sharp$ with probability $2/5$, i.e., by $6/5 > 1$ copies in expectation, and thus, it is not $\mathtt{AST}$. In contrast, a VRF of (7.61) keeps both annotations of $\mathsf{h}^\sharp$ in the same copy of $x$. Then one of them is at position 1.2 or 2.1, where the left-hand side of (7.62) contains the constant $\mathsf{h}$, so that its annotation is removed by (7.62). So in $\mathcal{A}$, $\mathsf{h}^\sharp$ is replaced by at most two copies of $\mathsf{h}^\sharp$ with probability $2/5$, i.e., by at most $4/5 < 1$ copies in expectation, and indeed, $\mathcal{A}$ is $\mathtt{AST}$.

We illustrate the instantiation processor with the following example.

Example 7.6.12. Consider the PTRS $\mathcal{P}_{\mathsf{ins}}$ with the rules

$$\begin{aligned}
&\mathsf{f}(x,y,z) \to \{^1\mathsf{g}(x,y,z)\} \qquad \mathsf{c} \to \{^1\mathsf{a}\} \qquad \mathsf{c} \to \{^1\mathsf{b}\}\\
&\mathsf{g}(\mathsf{a},\mathsf{b},u) \to \{^{1/2}\mathsf{f}(u,u,u),\, ^{1/2}\mathsf{g}(\mathsf{a},\mathsf{b},u)\}
\end{aligned}$$

$\mathcal{P}_{\mathsf{ins}}$ is not $\mathtt{AST}$, since $\mathsf{g}(\mathsf{a},\mathsf{b},\mathsf{c})$ starts an RT without leaves. However, $\mathtt{AST}(\overset{\mathsf{i}}{\to}_{\mathcal{P}_{\mathsf{ins}}})$ holds. Its canonical ADPs $\mathcal{DP}(\mathcal{P}_{\mathsf{ins}})$ are

$$\begin{aligned}
&\mathsf{f}(x,y,z) \overset{\mathsf{t}}{\hookrightarrow} \{^1\mathsf{g}^\sharp(x,y,z)\} \qquad \mathsf{c} \overset{\mathsf{t}}{\hookrightarrow} \{^1\mathsf{a}\} \qquad \mathsf{c} \overset{\mathsf{t}}{\hookrightarrow} \{^1\mathsf{b}\}\\
&\mathsf{g}(\mathsf{a},\mathsf{b},u) \overset{\mathsf{t}}{\hookrightarrow} \{^{1/2}\mathsf{f}^\sharp(u,u,u),\, ^{1/2}\mathsf{g}^\sharp(\mathsf{a},\mathsf{b},u)\}
\end{aligned}$$

Using only the processors of Section 7.3–7.5, we cannot prove $\mathtt{AST}(\overset{\mathsf{i}}{\to}_{\mathcal{P}_{\mathsf{ins}}})$. However, we can apply the instantiation processor on the $\mathsf{f}$-ADP.

As the term $t^\sharp = \mathsf{f}^\sharp(u,u,u)$ in the right-hand side of the $\mathsf{g}$-ADP does not have any proper subterms with (possibly annotated) defined symbols, we have $\mathrm{Cap}_{\mathcal{DP}(\mathcal{P}_{\mathsf{ins}})}(t^\sharp) = t^\sharp$. For the left-hand side $\ell^\sharp = \mathsf{f}^\sharp(x,y,z)$, $\delta = mgu(t^\sharp,\ell^\sharp)$ instantiates $x$, $y$, and $z$ by $u$. So the instantiation processor replaces the $\mathsf{f}$-ADP by $\mathsf{f}(u,u,u) \overset{\mathsf{t}}{\hookrightarrow} \{^1\mathsf{g}^\sharp(u,u,u)\}$ (and moreover, we add $\mathsf{f}(x,y,z) \overset{\mathsf{t}}{\hookrightarrow} \{^1\mathsf{g}(x,y,z)\}$). We can now remove the annotation in the transformed $\mathsf{f}$-ADP by the dependency graph processor, and afterwards remove the annotation in the $\mathsf{g}$-ADP by applying the reduction pair processor with the polynomial interpretation that maps $\mathsf{g}^\sharp_{\mathrm{Pol}}(x,y,z) = 1$ and $f_{\mathrm{Pol}} = 0$ for every other function symbol $f$. As we removed all annotations, $\mathcal{DP}(\mathcal{P}_{\mathsf{ins}})$ is $\mathtt{iAST}$, and hence $\mathtt{AST}(\overset{\mathsf{i}}{\to}_{\mathcal{P}_{\mathsf{ins}}})$ holds.

### 7.6.3 Forward Instantiation Processor

Next we adapt the *forward instantiation* processor. For non-probabilistic DPs, the idea of the forward instantiation processor [GTSF06] is to consider all possible *successors* $u^\sharp \to v^\sharp$ of a DP $s^\sharp \to t^\sharp$ in a chain, in order to find the skeleton that remains unchanged when rewriting $t^\sharp\sigma$ to $u^\sharp\sigma$ for some substitution $\sigma$. To find this skeleton, one reverses the rules of the TRS and then proceeds as for the instantiation processor. Note that these reversed rules might violate the variable conditions of TRSs, i.e., the right-hand side of a reversed rule may contain variables that do not occur in the left-hand side or the left-hand side may be a variable; we ignore these restrictions here.

More precisely, we reverse the underlying TRS $\mathrm{np}(\mathcal{A})$, i.e., we consider the reversed TRS $\mathrm{np}(\mathcal{A})^{-1} = \{r \to \ell \mid \ell \to r \in \mathrm{np}(\mathcal{A})\}$, and then reuse the cap function Cap for

$\mathrm{np}(\mathcal{A})^{-1}$. Thus, for a term $t \in \mathcal{T}$, the term $\mathrm{Cap}_{\mathrm{np}(\mathcal{A})^{-1}}(t^\sharp)$ results from $t^\sharp$ by replacing all its subterms $f(\ldots)$ by fresh variables, where $f$ is a defined symbol of $\mathrm{np}(\mathcal{A})^{-1}$, i.e., where $f$ occurs on the root position of some right-hand side of $\mathrm{np}(\mathcal{A})$. If $\mathrm{np}(\mathcal{A})^{-1}$ contains a rule with a variable left-hand side (equivalently, if $\mathrm{np}(\mathcal{A})$ contains a rule with a variable right-hand side), then every term is reducible by $\mathrm{np}(\mathcal{A})^{-1}$, and we define $\mathrm{Cap}_{\mathrm{np}(\mathcal{A})^{-1}}(t^\sharp)$ to be a fresh variable for every $t \in \mathcal{T}$.

Example 7.6.13. Consider an ADP problem $\mathcal{A}$ that contains the ADP $\mathsf{h}(x) \overset{\mathtt{t}}{\hookrightarrow} \{{}^1\mathsf{g}(x)\}$, so that $\mathrm{np}(\mathcal{A})$ contains the rule $\mathsf{h}(x) \to \mathsf{g}(x)$ and the reversed TRS $\mathrm{np}(\mathcal{A})^{-1}$ contains $\mathsf{g}(x) \to \mathsf{h}(x)$. The only defined symbol of $\mathrm{np}(\mathcal{A})^{-1}$ is $\mathsf{g}$, as $\mathsf{g}$ occurs on the root position of the right-hand side of the $\mathsf{h}$-rule. Hence, for the annotated term $\mathsf{f}^\sharp(\mathsf{h}(\mathsf{a}), \mathsf{g}(\mathsf{a}))$ we obtain $\mathrm{Cap}_{\mathrm{np}(\mathcal{A})^{-1}}(\mathsf{f}^\sharp(\mathsf{h}(\mathsf{a}), \mathsf{g}(\mathsf{a}))) = \mathsf{f}^\sharp(\mathsf{h}(\mathsf{a}), z)$. In contrast, the ordinary cap function $\mathrm{Cap}_{\mathrm{np}(\mathcal{A})}$ for the (non-reversed) rewrite relation would replace $\mathsf{h}(\mathsf{a})$ instead, i.e., $\mathrm{Cap}_{\mathrm{np}(\mathcal{A})}(\mathsf{f}^\sharp(\mathsf{h}(\mathsf{a}), \mathsf{g}(\mathsf{a}))) = \mathsf{f}^\sharp(z, \mathsf{g}(\mathsf{a}))$. Finally, if $\mathcal{A}$ contains the collapsing ADP $\mathsf{g}(x) \overset{\mathtt{t}}{\hookrightarrow} \{{}^1x\}$ instead, then $\mathrm{np}(\mathcal{A})^{-1}$ would contain the rule $x \to \mathsf{g}(x)$ with a variable left-hand side. Here, we have $\mathrm{Cap}_{\mathrm{np}(\mathcal{A})^{-1}}(\mathsf{f}^\sharp(\mathsf{h}(\mathsf{a}), \mathsf{g}(\mathsf{a}))) = y \in \mathcal{V}$.

For full rewriting, we have to use a coarser skeleton: let $\mathrm{Cap}^{\mathcal{D}}_{\mathrm{np}(\mathcal{A})^{-1}}(t^\sharp)$ result from $\mathrm{Cap}_{\mathrm{np}(\mathcal{A})^{-1}}(t^\sharp)$ by additionally replacing all subterms whose root is a (non-annotated) defined symbol from $\mathcal{D}$ by fresh variables. The reason is that the VRFs of an instantiated ADP $\ell\delta \overset{m}{\hookrightarrow} \mu\delta$ cannot keep annotations at the non-variable positions of $\ell\delta$ that are variable positions of $\ell$. Since annotations only occur at defined symbols, this is no problem if $\delta$ instantiates the variables of $\ell$ by constructor terms.

Theorem 7.6.14 – Forward Instantiation Processor for AST.
Let $\mathcal{A}$ be an ADP problem with $\mathcal{A} = \mathcal{A}' \uplus \{\ell \overset{m}{\hookrightarrow} \mu\}$. Then $\mathrm{Proc}_{\mathtt{fins}}$ is sound for AST, and it is complete for AST if $\ell \overset{m}{\hookrightarrow} \mu$ is L, where $\mathrm{Proc}_{\mathtt{fins}}(\mathcal{A}) = \{\mathcal{A}' \cup M \cup \{\ell \overset{m}{\hookrightarrow} \flat(\mu)\}\}$. Here,

$$M = \left\{ \delta(\ell \overset{m}{\hookrightarrow} \mu) \;\middle|\; \begin{array}{c} \ell' \overset{m'}{\hookrightarrow} \nu \in \mathrm{vr}(\mathcal{A}), r \in \mathrm{Supp}(\mu), t \trianglelefteq_\sharp r, \\ \delta = mgu(t^\sharp, \mathrm{Ren}(\mathrm{Cap}^{\mathcal{D}}_{\mathrm{np}(\mathcal{A})^{-1}}(\ell'^\sharp))) \end{array} \right\}$$

As in the non-probabilistic setting, the variant of this processor for innermost rewriting still has to use the renaming of all occurrences of variables via Ren since reversing the rules does not necessarily preserve the innermost evaluation strategy. However, instead of reversing all of $\mathrm{np}(\mathcal{A})$, we can restrict ourselves to the usable rules of $t^\sharp$, i.e., we use $\mathrm{Cap}_{\mathrm{np}(\mathcal{U}_\mathcal{A}(t^\sharp))^{-1}}$. Moreover, we do not have to replace subterms with defined root symbols, since innermost rewrite steps do not keep any annotations below the redex.

Theorem 7.6.15 – Forward Instantiation Processor for iAST.
Let $\mathcal{A}$ be an ADP problem with $\mathcal{A} = \mathcal{A}' \uplus \{\ell \overset{m}{\hookrightarrow} \mu\}$. Then $\mathrm{Proc}^{\mathtt{i}}_{\mathtt{fins}}$ is sound and complete for iAST, where $\mathrm{Proc}^{\mathtt{i}}_{\mathtt{fins}}(\mathcal{A}) = \{\mathcal{A}' \cup M \cup \{\ell \overset{m}{\hookrightarrow} \flat(\mu)\}\}$. Here,

$$M = \left\{ \delta(\ell \overset{m}{\hookrightarrow} \mu) \;\middle|\; \begin{array}{c} \ell' \overset{m'}{\hookrightarrow} \nu \in \mathrm{vr}(\mathcal{A}), r \in \mathrm{Supp}(\mu), t \trianglelefteq_\sharp r, \\ \delta = mgu(t^\sharp, \mathrm{Ren}(\mathrm{Cap}_{\mathrm{np}(\mathcal{U}_\mathcal{A}(t^\sharp))^{-1}}(\ell'^\sharp))), \{\ell\delta, \ell'\delta\} \subseteq \mathtt{ANF}_\mathcal{A} \end{array} \right\}$$

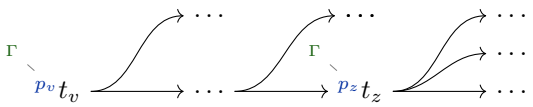
*Proof of both Theorem 7.6.14 and Theorem 7.6.15.* We prove both directions individually. As in the proof of Theorem 7.6.8, let $\mu = \{{}^{p_1}r_1, \ldots, {}^{p_k}r_k\}$, $\overline{\mathcal{A}'} = \mathcal{A}' \cup M \cup \{\ell \overset{m}{\hookrightarrow} \{{}^{p_1}\flat(r_1), \ldots, {}^{p_k}\flat(r_k)\}\}$, and $\overline{\mathcal{A}} = \overline{\mathcal{A}'} \cup \{\ell \overset{m}{\hookrightarrow} \{{}^{p_1}r_1, \ldots, {}^{p_k}r_k\}\}$. *Proof*

► *Proof* (*Soundness*): The core proof idea, the construction itself, the notion of unused annotations, and the induction hypothesis (7.57) are the same as for the soundness proof of Theorem 7.6.8, only the third case (C) changes.

**(C) If we have** $t_v \hookrightarrow_{\mathcal{A}} \{{}^{p_{w_1}/p_v}t_{w_1}, \dots, {}^{p_{w_k}/p_v}t_{w_k}\}$ using the ADP $\ell \xrightarrow{m} \{{}^{p_1}r_1, \dots, {}^{p_k}r_k\}$, the position $\pi \in \mathrm{pos}_{\mathcal{D}^\sharp}(t_v)$, a VRF $(\varphi_j)_{1\le j\le k}$, and a substitution $\sigma$ such that $\flat(t_v|_\pi) = \ell\sigma$ (where $\ell\sigma \in \mathtt{ANF}_{\mathcal{A}}$ in the innermost case), and $v$ is not the root, then $t_{w_j} = \flat_\pi^{\uparrow}(t_v[\sharp_{\mathrm{pos}_{\mathcal{D}^\sharp}(r_j)\cup\Psi_j}(r_j\sigma)]_\pi)$ if $m = \mathsf{f}$ and $t_{w_j} = t_v[\sharp_{\mathrm{pos}_{\mathcal{D}^\sharp}(r_j)\cup\Psi_j}(r_j\sigma)]_\pi$, otherwise.

First consider the case where none of the annotations created by the right-hand sides is used later on, i.e., $\pi.\chi \in \mathrm{Unused}(w_j)$ for all $1 \le j \le k$ and all $\chi \in \mathrm{pos}_{\mathcal{D}^\sharp}(r_j)$. Then, we can use $\ell \xrightarrow{m} \{{}^{p_1}\flat(r_1), \dots, {}^{p_k}\flat(r_k)\}$ with the same position, the same substitution, and the same VRF instead, because the annotations of the right-hand sides will never be used, and all other annotated positions have the same successors in $\mathfrak{T}$ and $\mathfrak{T}'$, so that (7.57) is satisfied again.

Otherwise, there exist a $1 \le j \le k$ and a $\chi \in \mathrm{pos}_{\mathcal{D}^\sharp}(r_j)$ such that $\pi.\chi$ is used below $w_j$. So there is a node $z$ in the subtree of $w_j$ that performs a step of the form **at** or **af** at a position $\tau_z$ that results from $\pi.\chi$ via successors, where we choose $z$ to be the first such node on the corresponding path. At node $z$ we use an ADP $\ell' \xrightarrow{m'} \{{}^{p'_1}r'_1, \dots, {}^{p'_h}r'_h\}$ and the substitution $\sigma'$. On the path from $w_j$ to $z$, the annotated subterm at (the successors of) $\pi.\chi$ is only moved by steps above it, and it is only modified by steps strictly below it, as a step at its position would be a step of the form **at** or **af**. These steps strictly below it use ADPs with the flag $\mathsf{t}$, as otherwise the annotation would be removed. Hence, we have $\sharp_{\{\varepsilon\}}(r_j|_\chi)\sigma \to^*_{\mathrm{np}(\mathcal{A})} \sharp_{\{\varepsilon\}}(\ell')\sigma'$ (with innermost steps in the innermost case).




Again we have two different cases:

AST: Let $\mathcal{Q}' = \mathrm{np}(\mathcal{A})$ and let $\mathcal{Q} = \mathcal{Q}'^{-1}$, i.e., $\mathcal{Q}$ is exactly the reversed system $\mathrm{np}(\mathcal{A})^{-1}$ used in the construction of $M$. Since $\sharp_{\{\varepsilon\}}(r_j|_\chi)\sigma \to^*_{\mathcal{Q}'} \sharp_{\{\varepsilon\}}(\ell')\sigma'$ only uses steps below the root, we have $\sharp_{\{\varepsilon\}}(\ell')\sigma' \to^*_{\mathcal{Q}} \sharp_{\{\varepsilon\}}(r_j|_\chi)\sigma$. By Lemma 7.6.10 (applied to the TRS $\mathcal{Q}$ and the function $\mathrm{Cap}^{\mathcal{D}}_{\mathcal{Q}}$), one directly obtains $\sharp_{\{\varepsilon\}}(r_j|_\chi)\sigma = \mathrm{Ren}(\mathrm{Cap}^{\mathcal{D}}_{\mathcal{Q}}(\sharp_{\{\varepsilon\}}(\ell')))\delta$ for some substitution $\delta$ that differs from $\sigma'$ at most on the variables that are introduced by Ren. Since the ADPs are variable-renamed, we can assume that $\sigma$ is equal to $\delta$ on all the fresh variables and all the variables from $\sharp_{\{\varepsilon\}}(\ell')$. Hence, $\sharp_{\{\varepsilon\}}(r_j|_\chi)\sigma = \mathrm{Ren}(\mathrm{Cap}^{\mathcal{D}}_{\mathcal{Q}}(\sharp_{\{\varepsilon\}}(\ell')))\sigma$ shows that there is an mgu $\gamma$ of $\sharp_{\{\varepsilon\}}(r_j|_\chi)$ and $\mathrm{Ren}(\mathrm{Cap}^{\mathcal{D}}_{\mathcal{Q}}(\sharp_{\{\varepsilon\}}(\ell')))$ with $\sigma = \gamma\zeta$ for some substitution $\zeta$, so that $\ell\gamma \xrightarrow{m} \{{}^{p_1}r_1\gamma, \dots, {}^{p_k}r_k\gamma\} \in M$ (up to variable renaming). As shown after Theorem 7.6.14, we have $\gamma(x) \in \mathcal{T}(\mathcal{C}, \mathcal{V})$ for all $x \in \mathcal{V}(\ell)$. Since annotations only occur at defined symbols, every annotated position of $t_v$ below a variable position $\pi.\rho$ with $\rho \in \mathrm{pos}_{\mathcal{V}}(\ell)$ is of the form $\pi.\rho.\rho'.\kappa$ for a variable position $\rho'$ of $\gamma(\ell|_\rho)$. Hence, we can apply the new ADP $\ell\gamma \xrightarrow{m} \{{}^{p_1}r_1\gamma, \dots, {}^{p_k}r_k\gamma\}$ with the position $\pi$, the substitution $\zeta$, and the VRF $(\varphi'_j)_{1\le j\le k}$ that is constructed from $(\varphi_j)_{1\le j\le k}$ as in Case (C) of the soundness proof of Theorem 7.6.8. Then every annotated position of $t_v$ has the same

successors in $\mathfrak{T}$ and $\mathfrak{T}'$, and the annotations of all $r_j$ are created in both $\mathfrak{T}$ and $\mathfrak{T}'$. So we have $t'_v \hookrightarrow_{\overline{\mathcal{A}'}} \{{}^{p_{w_1}/p_v}t'_{w_1}, \ldots, {}^{p_{w_k}/p_v}t'_{w_k}\}$ with $t_{w_j} \overset{\flat}{=} t'_{w_j}$, and (7.57) is satisfied again. As the step with the instantiated ADP yields the same terms with the same annotations as the original step in all children, it does not matter which of the nodes $z$ we use to determine $\gamma$.

iAST: Let $\mathcal{Q}' = \mathrm{np}(\mathcal{U}_{\mathcal{A}}(\sharp_{\{\varepsilon\}}(r_j|_\chi)))$ and let $\mathcal{Q} = \mathcal{Q}'^{-1}$, i.e., $\mathcal{Q}$ is exactly the reversed system $\mathrm{np}(\mathcal{U}_{\mathcal{A}}(t^\sharp))^{-1}$ used in the construction of $M$ (where $t^\sharp = \sharp_{\{\varepsilon\}}(r_j|_\chi)$). Due to innermost evaluation, the steps from $\sharp_{\{\varepsilon\}}(r_j|_\chi)\sigma$ to $\sharp_{\{\varepsilon\}}(\ell')\sigma'$ only use rules from $\mathcal{Q}'$, so $\sharp_{\{\varepsilon\}}(\ell')\sigma' \to^*_{\mathcal{Q}} \sharp_{\{\varepsilon\}}(r_j|_\chi)\sigma$, since $\sharp_{\{\varepsilon\}}(r_j|_\chi)\sigma \to^*_{\mathcal{Q}'} \sharp_{\{\varepsilon\}}(\ell')\sigma'$. By Lemma 7.6.10 (applied to the TRS $\mathcal{Q}$) one directly obtains $\sharp_{\{\varepsilon\}}(r_j|_\chi)\sigma = \mathrm{Ren}(\mathrm{Cap}_{\mathcal{Q}}(\sharp_{\{\varepsilon\}}(\ell')))\delta$ for some substitution $\delta$ that differs from $\sigma'$ at most on the variables that are introduced by Ren and $\mathrm{Cap}_{\mathcal{Q}}$. Since the ADPs are variable-renamed, we can assume that $\sigma$ is equal to $\delta$ on all the fresh variables and all the variables from $\sharp_{\{\varepsilon\}}(\ell')$. Hence, $\sharp_{\{\varepsilon\}}(r_j|_\chi)\sigma = \mathrm{Ren}(\mathrm{Cap}_{\mathcal{Q}}(\sharp_{\{\varepsilon\}}(\ell')))\sigma$ shows that there is an mgu $\gamma$ of $\sharp_{\{\varepsilon\}}(r_j|_\chi)$ and $\mathrm{Ren}(\mathrm{Cap}_{\mathcal{Q}}(\sharp_{\{\varepsilon\}}(\ell')))$ with $\sigma = \gamma\zeta$ for some substitution $\zeta$. Moreover, the property $\{\ell\sigma, \ell'\sigma'\} \subseteq \mathtt{ANF}_{\mathcal{A}}$ must remain true when replacing $\sigma$ and $\sigma'$ by the more general substitution $\gamma$, i.e., $\{\ell\gamma, \ell'\gamma\} \subseteq \mathtt{ANF}_{\mathcal{A}}$. Hence, we can apply the new ADP $\ell\gamma \overset{m}{\hookrightarrow} \{{}^{p_1}r_1\gamma, \ldots, {}^{p_k}r_k\gamma\} \in M$ with the position $\pi$ and the substitution $\zeta$ (all VRFs map every position to $\bot$ in innermost steps). This means that we have $t'_v \overset{\mathtt{i}}{\hookrightarrow}_{\overline{\mathcal{A}'}} \{{}^{p_{w_1}/p_v}t'_{w_1}, \ldots, {}^{p_{w_k}/p_v}t'_{w_k}\}$ with $t'_{w_j} = t'_v[r_j\gamma\zeta]_\pi$ if $m = \mathtt{t}$, or $t'_{w_j} = \flat^{\uparrow}_{\pi}(t'_v[r_j\gamma\zeta]_\pi)$, otherwise. Since $\sigma = \gamma\zeta$, we directly get $t_{w_j} \overset{\flat}{=} t'_{w_j}$, and (7.57) is satisfied again.

▶ *Proof* (*Completeness*): The same as for Theorem 7.6.8 (where for AST we again need that $\ell \overset{m}{\hookrightarrow} \mu$ is L). *QED*

The following counterexample shows that the restriction to $\mathrm{Cap}^{\mathcal{D}}_{\mathrm{np}(\mathcal{A})^{-1}}$ is needed for soundness w.r.t. AST.

Counterexample 7.6.16. Consider the ADP problem $\mathcal{A}$ with the ADPs

$$\mathsf{f}(x) \overset{\mathtt{t}}{\hookrightarrow} \{{}^{1/2}\mathsf{g}^\sharp(x), {}^{1/2}\mathsf{h}(x)\} \tag{7.64}$$

$$\mathsf{g}(\mathsf{k}(y)) \overset{\mathtt{t}}{\hookrightarrow} \{{}^{3/4}\mathsf{e}(\mathsf{f}^\sharp(\mathsf{k}^\sharp(\mathsf{c})), \mathsf{f}^\sharp(\mathsf{k}^\sharp(\mathsf{c}))), {}^{1/4}\mathsf{d}\} \tag{7.65}$$

$$\mathsf{k}(z) \overset{\mathtt{t}}{\hookrightarrow} \{{}^{3/4}\mathsf{f}^\sharp(\mathsf{k}^\sharp(\mathsf{c})), {}^{1/4}\mathsf{d}\} \tag{7.66}$$

where $\mathcal{D} = \{\mathsf{f}, \mathsf{g}, \mathsf{k}\}$ and $\mathcal{C} = \{\mathsf{h}, \mathsf{e}, \mathsf{c}, \mathsf{d}\}$. With (7.64) and the VRF that keeps the annotation of $\mathsf{k}^\sharp$, $\mathsf{f}^\sharp(\mathsf{k}^\sharp(\mathsf{c}))$ rewrites to $\{{}^{1/2}\mathsf{g}^\sharp(\mathsf{k}^\sharp(\mathsf{c})), {}^{1/2}\mathsf{h}(\mathsf{k}^\sharp(\mathsf{c}))\}$. Then (7.65) at the root of $\mathsf{g}^\sharp(\mathsf{k}^\sharp(\mathsf{c}))$ and (7.66) at the $\mathsf{k}^\sharp$ in $\mathsf{h}(\mathsf{k}^\sharp(\mathsf{c}))$ create new copies of $\mathsf{f}^\sharp(\mathsf{k}^\sharp(\mathsf{c}))$. So $\mathsf{f}^\sharp(\mathsf{k}^\sharp(\mathsf{c}))$ is replaced by $1/2 \cdot 3/4 \cdot 2 + 1/2 \cdot 3/4 = 9/8 > 1$ copies of itself in expectation, and thus, $\mathcal{A}$ is not AST.

If we used $\mathrm{Cap}_{\mathrm{np}(\mathcal{A})^{-1}}$ instead of $\mathrm{Cap}^{\mathcal{D}}_{\mathrm{np}(\mathcal{A})^{-1}}$ in Theorem 7.6.14 for (7.64), then we would obtain $\delta = mgu(\mathsf{g}^\sharp(x), \mathsf{g}^\sharp(\mathsf{k}(y'))) = [x/\mathsf{k}(y')]$ from (7.65), since $\mathsf{k}$ does not occur on the root position of any right-hand side of $\mathrm{np}(\mathcal{A})$. Then (7.64) is replaced by

$$\mathsf{f}(\mathsf{k}(y)) \overset{\mathtt{t}}{\hookrightarrow} \{{}^{1/2}\mathsf{g}^\sharp(\mathsf{k}(y)), {}^{1/2}\mathsf{h}(\mathsf{k}(y))\} \tag{7.67}$$

$$\mathsf{f}(x) \overset{\mathtt{t}}{\hookrightarrow} \{{}^{1/2}\mathsf{g}(x), {}^{1/2}\mathsf{h}(x)\} \tag{7.68}$$

Now $\mathsf{f}^\sharp(\mathsf{k}^\sharp(\mathsf{c}))$ can only use one of the two possibilities to create new copies of itself: (7.67) removes the annotation of $\mathsf{k}^\sharp$, since position 1 is not a variable position of $\mathsf{f}(\mathsf{k}(y))$,

(7.68) creates no annotated $\mathsf{g}^\sharp$, and rewriting $\mathsf{k}^\sharp$ first makes (7.67) inapplicable. In each case, $\mathsf{f}^\sharp(\mathsf{k}^\sharp(\mathsf{c}))$ is replaced by at most $3/4 < 1$ copies of itself in expectation, and indeed, the resulting ADP problem is $\mathtt{AST}$, i.e., using $\mathrm{Cap}_{\mathrm{np}(\mathcal{A})^{-1}}$ would be unsound. In contrast, $\mathrm{Cap}^{\mathcal{D}}_{\mathrm{np}(\mathcal{A})^{-1}}(\mathsf{g}^\sharp(\mathsf{k}(y'))) = \mathsf{g}^\sharp(y'')$, so with Theorem 7.6.14, $\delta$ is just a variable renaming, and $\mathrm{Proc}_{\mathtt{fins}}$ only adds (7.68) to $\mathcal{A}$.

We illustrate the forward instantiation processor with the following example.

Example 7.6.17. Consider the PTRS $\mathcal{P}_{\mathsf{fins}}$ with the rules

$$\begin{aligned}
&\mathsf{f}(x) \to \{^{1/2}\mathsf{g}(x), ^{1/2}\mathsf{h}(x)\} && \mathsf{q}(\mathsf{a}) \to \{^{1}\mathsf{a}\}\\
&\mathsf{g}(\mathsf{a}) \to \{^{1}\mathsf{f}(\mathsf{q}(\mathsf{a}))\} && \mathsf{q}(\mathsf{b}) \to \{^{1}\mathsf{b}\}\\
&\mathsf{h}(\mathsf{b}) \to \{^{1}\mathsf{f}(\mathsf{q}(\mathsf{b}))\}
\end{aligned}$$

When trying to prove $\mathtt{AST}(\xrightarrow{\mathsf{i}}_{\mathcal{P}_{\mathsf{fins}}})$, by the usable rules and the usable terms processor, we obtain the ADP problem $\mathcal{A}_{\mathsf{fins}}$

$$\begin{aligned}
&\mathsf{f}(x) \overset{\mathsf{f}}{\hookrightarrow} \{^{1/2}\mathsf{g}^\sharp(x), ^{1/2}\mathsf{h}^\sharp(x)\} && \mathsf{q}(\mathsf{a}) \overset{\mathsf{t}}{\hookrightarrow} \{^{1}\mathsf{a}\}\\
&\mathsf{g}(\mathsf{a}) \overset{\mathsf{f}}{\hookrightarrow} \{^{1}\mathsf{f}^\sharp(\mathsf{q}(\mathsf{a}))\} && \mathsf{q}(\mathsf{b}) \overset{\mathsf{t}}{\hookrightarrow} \{^{1}\mathsf{b}\}\\
&\mathsf{h}(\mathsf{b}) \overset{\mathsf{f}}{\hookrightarrow} \{^{1}\mathsf{f}^\sharp(\mathsf{q}(\mathsf{b}))\}
\end{aligned}$$

Here, the instantiation processor is useless, as both the mgu of $\mathrm{Cap}_{\mathcal{A}_{\mathsf{fins}}}(\mathsf{f}^\sharp(\mathsf{q}(\mathsf{a}))) = \mathsf{f}^\sharp(y)$ and $\mathsf{f}^\sharp(x)$, and the mgu of $\mathrm{Cap}_{\mathcal{A}_{\mathsf{fins}}}(\mathsf{f}^\sharp(\mathsf{q}(\mathsf{b}))) = \mathsf{f}^\sharp(z)$ and $\mathsf{f}^\sharp(x)$ do not modify $\mathsf{f}^\sharp(x)$. However, the forward instantiation processor can replace the original $\mathsf{f}$-ADP with $\mathsf{f}(\mathsf{a}) \overset{\mathsf{f}}{\hookrightarrow} \{^{1/2}\mathsf{g}^\sharp(\mathsf{a}), ^{1/2}\mathsf{h}^\sharp(\mathsf{a})\}$ and $\mathsf{f}(\mathsf{b}) \overset{\mathsf{f}}{\hookrightarrow} \{^{1/2}\mathsf{g}^\sharp(\mathsf{b}), ^{1/2}\mathsf{h}^\sharp(\mathsf{b})\}$. The reason is that the only possible successors of the $\mathsf{f}$-ADP are the $\mathsf{g}$- and the $\mathsf{h}$-ADP. For the $\mathsf{g}$-ADP, we have $\mathcal{U}_{\mathcal{A}_{\mathsf{fins}}}(\mathsf{g}^\sharp(x)) = \varnothing$ and thus, $mgu(\mathsf{g}^\sharp(x), \mathrm{Ren}(\mathrm{Cap}_{\mathrm{np}(\mathcal{U}_{\mathcal{A}_{\mathsf{fins}}}(\mathsf{g}^\sharp(x)))^{-1}}(\mathsf{g}^\sharp(\mathsf{a})))) = mgu(\mathsf{g}^\sharp(x), \mathsf{g}^\sharp(\mathsf{a}))$ instantiates $x$ with $\mathsf{a}$. For the $\mathsf{h}$-ADP, the mgu instantiates $x$ with $\mathsf{b}$. (Note that here the restriction to the usable rules is crucial in the innermost forward instantiation processor, because $\mathrm{Cap}_{\mathrm{np}(\mathcal{A}_{\mathsf{fins}})^{-1}}$ would replace the symbol $\mathsf{a}$ by a fresh variable, since it occurs on the root position in the right-hand side of an ADP from $\mathcal{A}_{\mathsf{fins}}$. Hence, $mgu(\mathsf{g}^\sharp(x), \mathrm{Ren}(\mathrm{Cap}_{\mathrm{np}(\mathcal{A}_{\mathsf{fins}})^{-1}}(\mathsf{g}^\sharp(\mathsf{a})))) = mgu(\mathsf{g}^\sharp(x), \mathsf{g}^\sharp(y))$ would not modify $x$.)

We can now remove the annotations of the normal forms $\mathsf{g}^\sharp(\mathsf{b})$ and $\mathsf{h}^\sharp(\mathsf{a})$ from the new instantiated $\mathsf{f}$-ADPs with the usable terms processor, and apply the reduction pair processor with the polynomial interpretation with $f_{\mathrm{Pol}} = 1$ for every function symbol $f$, in order to remove all annotations of both new $\mathsf{f}$-ADPs. Finally, we can remove all remaining annotations and prove $\mathtt{AST}(\xrightarrow{\mathsf{i}}_{\mathcal{P}_{\mathsf{fins}}})$ using the dependency graph processor. Since the PTRS is non-overlapping and linear, this also implies $\mathtt{AST}(\to_{\mathcal{P}_{\mathsf{fins}}})$, see Theorem 4.2.2. Note that without the forward instantiation processor, we would have to find a polynomial interpretation that is at least linear.

Example 7.6.17 shows that the processors that instantiate a given ADP are not only sometimes needed for a successful termination proof (as in Example 7.6.12), but sometimes they can also ease the search for algebras (e.g., polynomial interpretations) in the reduction pair processor, the most time-consuming part of the ADP framework.

### 7.6.4 Rule Overlap Instantiation Processor

The *narrowing* processor [AG00, GTSF06] can only be used in a weaker version in the probabilistic setting, and only for innermost rewriting. Let $\mathcal{A} = \mathcal{A}' \uplus \{\ell \xhookrightarrow{m} \{^{p_1}r_1, \ldots, ^{p_k}r_k\}\}$ be an ADP problem. For each $1 \leq j \leq k$ and each $t \trianglelefteq_\sharp r_j$, we define its *narrowing substitutions* and its *narrowing results* (as in [NEG13], where narrowing was adapted to dependency tuples for complexity analysis of ordinary term rewriting). If we have to perform rewrite steps on (an instance of) $t$ in order to enable the next application of an ADP at an annotated position, then the idea of the narrowing processor is to perform the first step of this reduction already on the ADP $\ell \xhookrightarrow{m} \{^{p_1}r_1, \ldots, ^{p_k}r_k\}$. So whenever there is a $t \trianglelefteq_\sharp r_j$ and a non-variable position $\tau$ in $t$ such that $t|_\tau$ unifies with the left-hand side $\ell'$ of some ADP $\ell' \xhookrightarrow{m'} \{^{p'_1}r'_1, \ldots, ^{p'_{k'}}r'_{k'}\} \in \mathrm{vr}(\mathcal{A})$ using an mgu $\delta$ (such that $\ell\delta, \ell'\delta \in \mathtt{ANF}_\mathcal{A}$ if we consider innermost rewriting), then $\delta$ is a *narrowing substitution* of $t$. While the corresponding *narrowing result* could also be defined for ADPs with non-trivial probabilities, to simplify the presentation let us assume for the moment that the ADP just has the form $\ell' \xhookrightarrow{t} \{^1r'\}$. Then the corresponding *narrowing result* is $s = t[r']_\tau\delta$ if we rewrite at a position of $t$ that used to be annotated in $r_j$, and $s = t[\flat(r')]_\tau\delta$ otherwise.

If $\delta_1, \ldots, \delta_d$ are all narrowing substitutions of $t$ with the corresponding narrowing results $s_1, \ldots, s_d$, then one would like to define a narrowing processor that replaces $\ell \xhookrightarrow{m} \{^{p_1}r_1, \ldots, ^{p_k}r_k\}$ by $\ell\delta_e \xhookrightarrow{m} \{^{p_1}r_1\delta_e, \ldots, ^{p_j}s_e, \ldots, ^{p_k}r_k\delta_e\}$, for all $1 \leq e \leq d$. However, the main idea of the narrowing processor, i.e., performing the first rewrite step directly on the ADPs, is unsound for probabilistic ADP problems, as shown by the following counterexample.

Counterexample 7.6.18. Consider the following ADP problem $\mathcal{A}$:

$$\begin{aligned} \mathsf{f}(\mathsf{b}_1, \mathsf{d}_1) &\xhookrightarrow{\mathsf{f}} \{^1\mathsf{f}^\sharp(\mathsf{a}, \mathsf{e})\} & \mathsf{e} &\xhookrightarrow{\mathsf{t}} \{^1\mathsf{d}_1\} \\ \mathsf{f}(\mathsf{b}_2, \mathsf{d}_2) &\xhookrightarrow{\mathsf{f}} \{^1\mathsf{f}^\sharp(\mathsf{a}, \mathsf{e})\} & \mathsf{e} &\xhookrightarrow{\mathsf{t}} \{^1\mathsf{d}_2\} \\ \mathsf{a} &\xhookrightarrow{\mathsf{t}} \{^{1/2}\mathsf{b}_1, ^{1/2}\mathsf{b}_2\} \end{aligned}$$

This ADP problem does not satisfy $\mathtt{AST}(\xhookrightarrow{\mathtt{i}}_\mathcal{A})$, because it allows for the CT in Figure 7.23 without any leaves. Note that all occurring terms are ground terms, hence all narrowing substitutions are just the identity function.

But if we apply the narrowing processor to the ADPs in order to rewrite $\mathsf{e}$, then we replace the two $\mathsf{f}$-ADPs by the following four new ADPs.

$$\begin{aligned} \mathsf{f}(\mathsf{b}_1, \mathsf{d}_1) &\xhookrightarrow{\mathsf{f}} \{^1\mathsf{f}^\sharp(\mathsf{a}, \mathsf{d}_1)\} & \mathsf{f}(\mathsf{b}_2, \mathsf{d}_2) &\xhookrightarrow{\mathsf{f}} \{^1\mathsf{f}^\sharp(\mathsf{a}, \mathsf{d}_1)\} \\ \mathsf{f}(\mathsf{b}_1, \mathsf{d}_1) &\xhookrightarrow{\mathsf{f}} \{^1\mathsf{f}^\sharp(\mathsf{a}, \mathsf{d}_2)\} & \mathsf{f}(\mathsf{b}_2, \mathsf{d}_2) &\xhookrightarrow{\mathsf{f}} \{^1\mathsf{f}^\sharp(\mathsf{a}, \mathsf{d}_2)\} \end{aligned}$$

This new ADP problem $\mathcal{A}'$ satisfies $\mathtt{AST}(\xhookrightarrow{\mathtt{i}}_{\mathcal{A}'})$, as we reach a normal form with probability $1/2$ after each application of an $\mathsf{f}$-ADP. For example, if we use the first ADP, then we get the CT depicted in Figure 7.24. There we reach the normal form $\mathsf{f}^\sharp(\mathsf{b}_2, \mathsf{d}_1)$ with probability $1/2$.

The difference is that when narrowing the ADP, we have to decide how to rewrite $\mathsf{e}$ before we "split" the term into two different successors with a certain probability, i.e., before we rewrite $\mathsf{a}$ to $\{^{1/2}\mathsf{b}_1, ^{1/2}\mathsf{b}_2\}$.

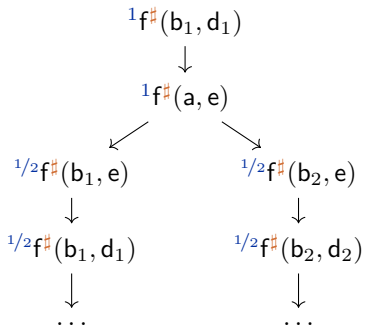



Figure 7.23: Innermost $\mathcal{A}$-CT without leaves.

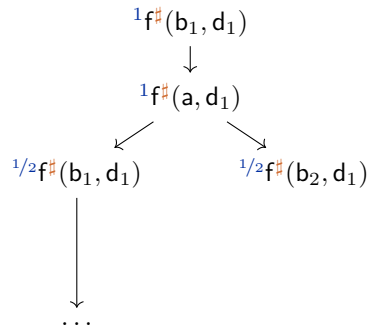



Figure 7.24: Innermost $\mathcal{A}'$-CT after narrowing.

Thus, in the probabilistic setting, we can only transform the ADP by applying a rewrite rule if we can ensure the same conditions as for the rewriting processor in Section 7.6.1. So in the probabilistic setting, the narrowing processor can only instantiate the ADP by the narrowing substitutions, but it must not perform any rewrite step. Instead, the rewrite steps have to be done via the rewriting processor afterwards. Thus, instead of calling it *narrowing processor*, we call it *rule overlap instantiation processor* as it only instantiates the ADPs but does not perform any rewrite steps.

**Rule Overlap Instantiation Processor for `iAST`** Unfortunately, considering only the narrowing substitutions $\delta_1,\dots,\delta_d$ of $t$ is not sufficient. There could be another subterm $t' \trianglelefteq_\sharp r_j$ (with $t' \neq t$) which was involved in a CT (i.e., $t'^{\sharp}\sigma \to^*_{\mathrm{np}(\mathcal{A})} \tilde{\ell}\tilde{\sigma}$ for some substitutions $\sigma,\tilde{\sigma}$ and a left-hand side $\tilde{\ell}$ of an ADP, where in the innermost case we require $\xrightarrow{\mathtt{i}}{}^*_{\mathrm{np}(\mathcal{A})}$ instead of $\to^*_{\mathrm{np}(\mathcal{A})}$), but this CT is no longer possible when instantiating $t'$ to $t'\delta_1,\dots,t'\delta_d$. We say that $t'$ is *covered*[11] by $\delta_1,\dots,\delta_d$ if for each narrowing substitution $\rho$ of $t'$, there is a $\delta_e$ with $1 \le e \le d$ such that $\delta_e$ is more general than $\rho$, i.e., $\rho = \delta_e\rho'$ for some substitution $\rho'$. So the narrowing processor has to add another ADP $\ell \xhookrightarrow{m} \{^{p_1}\sharp_{\mathrm{capt}_1(\delta_1,\dots,\delta_d)}(r_1),\dots,{}^{p_k}\sharp_{\mathrm{capt}_k(\delta_1,\dots,\delta_d)}(r_k)\}$, where $\mathrm{capt}_i(\delta_1,\dots,\delta_d)$ contains all positions of subterms $t' \trianglelefteq_\sharp r_i$ which are not covered by $\delta_1,\dots,\delta_d$. (Therefore, in contrast to instantiation and forward instantiation, here we do not have to add another copy of the original rule without annotations, because the original rule is represented by this additional ADP.) Thus, for innermost rewriting, we obtain the following processor.

Theorem 7.6.19 – Rule Overlap Instantiation Processor for `iAST`.
Let $\mathcal{A}$ be an ADP problem with $\mathcal{A} = \mathcal{A}' \uplus \{\ell \xhookrightarrow{m} \{^{p_1}r_1,\dots,{}^{p_k}r_k\}\}$, let $1 \le j \le k$, and let $t \trianglelefteq_\sharp r_j$. Let $\delta_1,\dots,\delta_d$ be all narrowing substitutions of $t$, where $d \ge 0$. Then $\mathrm{Proc}^{\mathtt{i}}_{\mathtt{roi}}(\mathcal{A}) = \{\mathcal{A}' \cup M\}$ is sound and complete for `iAST`, where

$$M = \left\{\ell\delta_e \xhookrightarrow{m} \{^{p_1}r_1\delta_e,\dots,{}^{p_k}r_k\delta_e\} \,\middle|\, 1 \le e \le d\right\} \cup \left\{\ \ell \xhookrightarrow{m} \{^{p_1}\sharp_{\mathrm{capt}_1(\delta_1,\dots,\delta_d)}(r_1),\dots,{}^{p_k}\sharp_{\mathrm{capt}_k(\delta_1,\dots,\delta_d)}(r_k)\}\ \right\}$$

[11] This was called "captured" in [NEG13].

*Proof.* We prove both directions individually. *Proof*

▶ *Proof*(*Soundness*): We use the same idea as in the proof of soundness for Theorem 7.6.14 but at a node $v$ that uses the position $\pi \in \mathrm{pos}_{\mathcal{D}^\sharp}(t_v)$ we do not only look at the next node that rewrites at an annotated position created by this step, but at the first node that rewrites at or below such a position, regardless of whether the position of this rewrite step is annotated or not.

Let $\mathtt{AST}(\overset{\mathtt{i}}{\hookrightarrow}_{\mathcal{A}})$ not hold. Then by Lemma 7.3.3 there exists an innermost $\mathcal{A}$-CT $\mathfrak{T} = (N, E)$ that converges with probability $< 1$ that starts with a term $u$ where $\flat(u) = s\theta \in \mathtt{ANF}_{\mathcal{A}}$ for a substitution $\theta$ and an ADP $s \overset{m'}{\hookrightarrow} \nu \in \mathcal{A}$, and $\mathrm{pos}_{\mathcal{D}^\sharp}(u) = \{\varepsilon\}$. Let $\overline{\mathcal{A}'} = \mathcal{A}' \cup M$. We will now create an innermost $\overline{\mathcal{A}'}$-CT $\mathfrak{T}' = (N, E)$, with the same underlying tree structure, such that $p_v^{\mathfrak{T}} = p_v^{\mathfrak{T}'}$ for all $v \in N$. Since the tree structure and the probabilities are the same, we then get $|\mathfrak{T}'| = |\mathfrak{T}|$, and hence $\mathtt{AST}(\overset{\mathtt{i}}{\hookrightarrow}_{\overline{\mathcal{A}'}})$ does not hold either.

The core idea of this construction is that every rewrite step with $\ell \overset{m}{\hookrightarrow} \{{}^{p_1}r_1, \dots, {}^{p_k}r_k\}$ can also be done with a rule from $M$. If we use $\ell \overset{m}{\hookrightarrow} \{{}^{p_1}\sharp_{\mathrm{capt}_1(\delta_1,\dots,\delta_d)}(r_1), \dots, {}^{p_k}\sharp_{\mathrm{capt}_k(\delta_1,\dots,\delta_d)}(r_k)\} \in M$, we may create fewer annotations than we did when using the old ADP $\ell \overset{m}{\hookrightarrow} \{{}^{p_1}r_1, \dots, {}^{p_k}r_k\}$. However, we only do this if the annotations that are not created are never used in the $\mathcal{A}$-CT $\mathfrak{T}$. We construct the terms $t'_v$ for the nodes of the $\overline{\mathcal{A}'}$-CT $\mathfrak{T}'$ inductively such that for all nodes $v \in N \setminus \mathrm{Leaf}$ with $vE = \{w_1, \dots, w_h\}$ we have $t'_v \overset{\mathtt{i}}{\hookrightarrow}_{\overline{\mathcal{A}'}} \{{}^{p_{w_1}/p_v}t'_{w_1}, \dots, {}^{p_{w_h}/p_v}t'_{w_h}\}$. Let $W \subseteq N$ be the set of nodes $v$ where we have already defined the corresponding term $t'_v$. During our construction, we ensure the same induction hypothesis (7.57) as in the soundness proof of Theorem 7.6.8, i.e., $t_v \overset{\flat}{=} t'_v$ and $\mathrm{pos}_{\mathcal{D}^\sharp}(t_v) \setminus \mathrm{Unused}(v) \subseteq \mathrm{pos}_{\mathcal{D}^\sharp}(t'_v)$, where $\mathrm{Unused}(v)$ consists of all annotated positions of $t_v$ that are not used below $v$. Recall that in innermost steps, all VRFs map every position to $\bot$, so an annotated position only has itself as successor (if it is orthogonal to the position of the step, or above it and the used ADP has the flag $\mathtt{t}$).

For the construction, we start with the same term at the root. Here, (7.57) obviously holds. As long as there is still an inner node $v \in W$ such that its successors are not contained in $W$, we do the following. Let $vE = \{w_1, \dots, w_h\}$ be the set of its successors. We need to define the corresponding terms $t'_{w_1}, \dots, t'_{w_h}$ for the nodes $w_1, \dots, w_h$. Since $v$ is not a leaf and $\mathfrak{T}$ is an $\mathcal{A}$-CT, we have $t_v \overset{\mathtt{i}}{\hookrightarrow}_{\mathcal{A}} \{{}^{p_{w_1}/p_v}t_{w_1}, \dots, {}^{p_{w_h}/p_v}t_{w_h}\}$. We have the following three cases:

(A) If it is a step with $\overset{\mathtt{i}}{\hookrightarrow}_{\mathcal{A}}$ using an ADP that is different from $\ell \overset{m}{\hookrightarrow} \{{}^{p_1}r_1, \dots, {}^{p_k}r_k\}$ in $\mathfrak{T}$, then we perform a rewrite step with the same ADP, the same redex, and the same substitution in $\mathfrak{T}'$. Analogous to Case (A) of the soundness proof for Theorem 7.6.8, we can show (7.57) for the resulting terms.

(B) If it is a step with $\overset{\mathtt{i}}{\hookrightarrow}_{\mathcal{A}}$ using the ADP $\ell \overset{m}{\hookrightarrow} \{{}^{p_1}r_1, \dots, {}^{p_k}r_k\}$ at a position $\pi \notin \mathrm{pos}_{\mathcal{D}^\sharp}(t_v)$ in $\mathfrak{T}$, then we perform a rewrite step with $\ell \overset{m}{\hookrightarrow} \{{}^{p_1}\sharp_{\mathrm{capt}_1(\delta_1,\dots,\delta_d)}(r_1), \dots, {}^{p_k}\sharp_{\mathrm{capt}_k(\delta_1,\dots,\delta_d)}(r_k)\}$, same redex, same substitution, and same position in $\mathfrak{T}'$. Analogous to Case (B) of the soundness proof for Theorem 7.6.8, we can show (7.57) for the resulting terms. Note that the rule that we use contains fewer annotations than the original rule, but since $\pi \notin \mathrm{pos}_{\mathcal{D}^\sharp}(t_v)$, no annotations of the right-hand sides are created in $\mathfrak{T}$ anyway.

(C) If it is a step with $\overset{\mathtt{i}}{\hookrightarrow}_{\mathcal{A}}$ using the ADP $\ell \overset{m}{\hookrightarrow} \{{}^{p_1}r_1, \dots, {}^{p_k}r_k\}$ at a position $\pi \in \mathrm{pos}_{\mathcal{D}^\sharp}(t_v)$ in $\mathfrak{T}$, then we use either an ADP $\ell\delta_e \overset{m}{\hookrightarrow} \{{}^{p_1}r_1\delta_e, \dots, {}^{p_k}r_k\delta_e\} \in M$ or the ADP $\ell \overset{m}{\hookrightarrow} \{{}^{p_1}\sharp_{\mathrm{capt}_1(\delta_1,\dots,\delta_d)}(r_1), \dots, {}^{p_k}\sharp_{\mathrm{capt}_k(\delta_1,\dots,\delta_d)}(r_k)\}$ at the same position in $\mathfrak{T}'$.

So it remains to consider Case (C) in detail. Here, we have $t_v \overset{\mathsf{i}}{\hookrightarrow}_{\mathcal{A}} \{{}^{p_{w_1}/p_v}t_{w_1}, \dots, {}^{p_{w_k}/p_v}t_{w_k}\}$ using the ADP $\ell \overset{m}{\hookrightarrow} \{{}^{p_1}r_1, \dots, {}^{p_k}r_k\}$, the position $\pi \in \mathrm{pos}_{\mathcal{D}^\sharp}(t_v)$, and a substitution $\sigma$ such that $\flat(t_v|_\pi) = \ell\sigma \in \mathtt{ANF}_{\mathcal{A}}$, and we have $t_{w_i} = t_v[r_i\sigma]_\pi$ (or $t_{w_i} = \flat^{\uparrow}_{\pi}(t_v[r_i\sigma]_\pi)$ if $m = \mathsf{f}$) for all $1 \le i \le k$. Note that $\pi \in \mathrm{pos}_{\mathcal{D}^\sharp}(t'_v)$ by (7.57), since $\pi$ is used at $v$.

**(C1)** First, assume that for all $1 \le i \le k$ and all positions $\chi \in \mathrm{pos}_{\mathcal{D}^\sharp}(r_i)$ of subterms of $r_i$ that are covered by $\delta_1, \dots, \delta_d$, the annotated position $\pi.\chi$ is not used below $w_i$. Then we use the ADP $\ell \overset{m}{\hookrightarrow} \{{}^{p_1}\sharp_{\mathrm{capt}_1(\delta_1,\dots,\delta_d)}(r_1), \dots, {}^{p_k}\sharp_{\mathrm{capt}_k(\delta_1,\dots,\delta_d)}(r_k)\}$ with the position $\pi$ and the substitution $\sigma$. It creates all annotations of the right-hand sides that are not covered, and the annotations that are not created are in $\mathrm{Unused}(w_i)$. All other annotated positions have the same successors as in $\mathfrak{T}$. Hence, (7.57) is satisfied for our resulting terms.

**(C2)** Otherwise, there are an $1 \le i \le k$ and a position $\chi \in \mathrm{pos}_{\mathcal{D}^\sharp}(r_i)$ such that $t' = \flat(r_i|_\chi)$ is covered by $\delta_1, \dots, \delta_d$ and $\pi.\chi$ is used below $w_i$. Let $z$ be the first node on a corresponding path that performs a step of the form **at** or **af** at position $\pi.\chi$ (in innermost steps, annotated positions are never moved). Then on the path from $w_i$ to $z$, there is no step at a position above $\pi.\chi$, as such an innermost step would remove the annotation at $\pi.\chi$. Consider the first node $y$ on this path (where $y = z$ is possible) that performs a step at a position $\pi.\chi.\tau$ for some $\tau$, using an ADP $\ell' \overset{m'}{\hookrightarrow} \nu$ and a substitution $\sigma'$. Then $t_y|_{\pi.\chi} \overset{\flat}{=} t'\sigma$, so we have $t'|_\tau\sigma = \ell'\sigma' \in \mathtt{ANF}_{\mathcal{A}}$ if $\tau \in \mathrm{pos}(t')$. Since $\sigma$ instantiates all variables of $\ell$ by normal forms (as $\ell\sigma \in \mathtt{ANF}_{\mathcal{A}}$), $\tau$ must indeed be a non-variable position of $t'$. Hence, $t'|_\tau$ and $\ell'$ (after renaming its variables apart) are unifiable with an mgu $\rho$ such that $\sigma \cup \sigma'$ is an instance of $\rho$. Since $\ell\sigma, \ell'\sigma' \in \mathtt{ANF}_{\mathcal{A}}$ and these terms are instances of $\ell\rho$ and $\ell'\rho$, we also have $\ell\rho, \ell'\rho \in \mathtt{ANF}_{\mathcal{A}}$, so $\rho$ is a narrowing substitution of $t'$. As $t'$ is covered by $\delta_1, \dots, \delta_d$, there is a $1 \le e \le d$ such that $\rho = \delta_e\rho'$ for some substitution $\rho'$, and thus $\sigma = \delta_e\gamma$ on $\mathcal{V}(\ell)$ for some substitution $\gamma$. Now, we use the ADP $\ell\delta_e \overset{m}{\hookrightarrow} \{{}^{p_1}r_1\delta_e, \dots, {}^{p_k}r_k\delta_e\}$ with the position $\pi$ and the substitution $\gamma$ such that $\flat(t_v|_\pi) = \ell\delta_e\gamma = \ell\sigma \in \mathtt{ANF}_{\mathcal{A}}$. This step yields the same terms with the same annotations as the step in $\mathfrak{T}$, so (7.57) is satisfied for our resulting terms.

► *Proof (Completeness)*: The proof is analogous to the completeness proof of Theorem 7.6.8. We can replace each ADP $\ell\delta_e \overset{m}{\hookrightarrow} \{{}^{p_1}r_1\delta_e, \dots, {}^{p_k}r_k\delta_e\}$ with the more general one $\ell \overset{m}{\hookrightarrow} \{{}^{p_1}r_1, \dots, {}^{p_k}r_k\}$, and each ADP $\ell \overset{m}{\hookrightarrow} \{{}^{p_1}\sharp_{\mathrm{capt}_1(\delta_1,\dots,\delta_d)}(r_1), \dots, {}^{p_k}\sharp_{\mathrm{capt}_k(\delta_1,\dots,\delta_d)}(r_k)\}$ can be replaced by $\ell \overset{m}{\hookrightarrow} \{{}^{p_1}r_1, \dots, {}^{p_k}r_k\}$ as well, leading to more annotations than before. *QED*

We illustrate the rule overlap instantiation processor with the following example.

Example 7.6.20. Consider the PTRS $\mathcal{P}_{\mathsf{roi}}$.

$$\mathsf{f}(\mathsf{d}(x)) \to \{{}^{3/4}\mathsf{e}(\mathsf{f}(\mathsf{g}(x)), \mathsf{f}(\mathsf{h}(x))), {}^{1/4}\mathsf{a}\} \qquad \begin{aligned}\mathsf{g}(\mathsf{a}) &\to \{{}^{1}\mathsf{d}(\mathsf{a})\}\\ \mathsf{h}(\mathsf{b}) &\to \{{}^{1}\mathsf{d}(\mathsf{b})\}\end{aligned}$$

The usable terms processor transforms the canonical ADP problem into the following ADP problem $\mathcal{A}_{\mathsf{roi}}$.

$$\mathsf{f}(\mathsf{d}(x)) \overset{\mathsf{t}}{\hookrightarrow} \{{}^{3/4}\mathsf{e}(\mathsf{f}^\sharp(\mathsf{g}(x)), \mathsf{f}^\sharp(\mathsf{h}(x))), {}^{1/4}\mathsf{a}\} \qquad \begin{aligned}\mathsf{g}(\mathsf{a}) &\overset{\mathsf{t}}{\hookrightarrow} \{{}^{1}\mathsf{d}(\mathsf{a})\}\\ \mathsf{h}(\mathsf{b}) &\overset{\mathsf{t}}{\hookrightarrow} \{{}^{1}\mathsf{d}(\mathsf{b})\}\end{aligned}$$

The ADP problem $\mathcal{A}_{\mathsf{roi}}$ satisfies $\mathtt{AST}(\hookrightarrow_{\mathcal{A}_{\mathsf{roi}}})$ and hence also $\mathtt{AST}(\overset{\mathsf{i}}{\hookrightarrow}_{\mathcal{A}_{\mathsf{roi}}})$, because for every instantiation, at most one of the two "recursive $\mathsf{f}^\sharp$-calls" in the right-hand side of the $\mathsf{f}$-ADP can be applied. The reason is that we can either use the $\mathsf{g}$-rule if the variable $x$ is instantiated with $\mathsf{a}$, or we can apply the $\mathsf{h}$-rule if the variable is instantiated with $\mathsf{b}$, but not both. We apply $\mathrm{Proc}^{\mathsf{i}}_{\mathtt{roi}}$ using the term $t^\sharp = \mathsf{f}^\sharp(\mathsf{g}(x))$, where the only narrowing substitution of $t = \mathsf{f}(\mathsf{g}(x))$ is $\delta = [x/\mathsf{a}]$. For the other subterm $\mathsf{f}^\sharp(\mathsf{h}(x))$ with annotated root, its flattened version $\mathsf{f}(\mathsf{h}(x))$ is not covered by this substitution, because the substitution $\delta$ is not more general than the narrowing substitution $\delta' = [x/\mathsf{b}]$ of $\mathsf{f}(\mathsf{h}(x))$. Hence, we have to generate an additional ADP where this second subterm is annotated. Thus, we replace the former $\mathsf{f}$-ADP by the following two new ADPs.

$$\mathsf{f}(\mathsf{d}(\mathsf{a})) \overset{\mathsf{t}}{\hookrightarrow} \{^{3/4}\mathsf{e}(\mathsf{f}^\sharp(\mathsf{g}(\mathsf{a})), \mathsf{f}^\sharp(\mathsf{h}(\mathsf{a}))), {}^{1/4}\mathsf{a}\}$$
$$\mathsf{f}(\mathsf{d}(x)) \overset{\mathsf{t}}{\hookrightarrow} \{^{3/4}\mathsf{e}(\mathsf{f}(\mathsf{g}(x)), \mathsf{f}^\sharp(\mathsf{h}(x))), {}^{1/4}\mathsf{a}\}$$

Now one can remove the annotation of $\mathsf{f}^\sharp(\mathsf{h}(\mathsf{a}))$ from the first ADP by the usable terms processor and then apply the reduction pair processor with the polynomial interpretation that maps $\mathsf{f}^\sharp_{\mathrm{Pol}}(x) = 1$ and $f_{\mathrm{Pol}} = 0$ for all other symbols $f$ to remove all annotations, which proves $\mathtt{AST}(\overset{\mathsf{i}}{\to}_{\mathcal{P}_{\mathsf{roi}}})$. Again, proving $\mathtt{AST}(\overset{\mathsf{i}}{\to}_{\mathcal{P}_{\mathsf{roi}}})$ with such a simple polynomial interpretation would not be possible without the rule overlap instantiation processor.

**Rule Overlap Instantiation Processor for AST** For full rewriting, even a processor like the one of Theorem 7.6.19 which only applies the narrowing substitutions but does not perform any rewrite steps is unsound (where for full rewriting, the narrowing substitutions are computed without the requirement $\ell\delta, \ell'\delta \in \mathtt{ANF}_{\mathcal{A}}$), as shown by the following counterexample.

Counterexample 7.6.21. Consider the following variant of the ADP problem from Counterexample 7.4.17, where the $\mathsf{f}$-ADP is not left-linear:

$$\mathsf{a} \overset{\mathsf{t}}{\hookrightarrow} \{^{5/8}\mathsf{c}(\mathsf{a},\mathsf{a}),\ {}^{3/8}\mathsf{0}\} \qquad\qquad \mathsf{f}(\mathsf{c}(x,x)) \overset{\mathsf{t}}{\hookrightarrow} \{^{1}\mathsf{c}(\mathsf{f}^\sharp(x), \mathsf{f}^\sharp(x))\}$$

This ADP problem $\mathcal{A}$ is not $\mathtt{AST}$: as in Counterexample 7.4.17, we can rewrite $\mathsf{f}^\sharp(\mathsf{a})$ to $\{^{5/8}\mathsf{f}^\sharp(\mathsf{c}(\mathsf{a},\mathsf{a})), {}^{3/8}\mathsf{f}^\sharp(\mathsf{0})\}$, and $\mathsf{f}^\sharp(\mathsf{c}(\mathsf{a},\mathsf{a}))$ to $\{^{1}\mathsf{c}(\mathsf{f}^\sharp(\mathsf{a}), \mathsf{f}^\sharp(\mathsf{a}))\}$, while $\mathsf{f}^\sharp(\mathsf{0})$ is a normal form. So every subterm $\mathsf{f}^\sharp(\mathsf{a})$ is replaced by two copies of $\mathsf{f}^\sharp(\mathsf{a})$ with probability $5/8$, and it is removed with probability $3/8$.

The only narrowing substitution of $\mathsf{f}(x)$ is $\delta = [x/\mathsf{c}(y,y)]$, since $\delta$ is the mgu of $\mathsf{f}(x)$ and the (variable-renamed) left-hand side $\mathsf{f}(\mathsf{c}(y,y))$. Thus, both annotated subterms are covered by $\delta$, and the rule overlap instantiation processor replaces the $\mathsf{f}$-ADP by the following two ADPs.

$$\mathsf{f}\big(\mathsf{c}(\mathsf{c}(y,y),\mathsf{c}(y,y))\big) \overset{\mathsf{t}}{\hookrightarrow} \big\{^{1}\mathsf{c}\big(\mathsf{f}^\sharp(\mathsf{c}(y,y)), \mathsf{f}^\sharp(\mathsf{c}(y,y))\big)\big\}$$
$$\mathsf{f}(\mathsf{c}(x,x)) \overset{\mathsf{t}}{\hookrightarrow} \{^{1}\mathsf{c}(\mathsf{f}(x), \mathsf{f}(x))\}$$

In the resulting ADP problem $\mathcal{A}'$, we can only rewrite $\mathsf{f}^\sharp\big(\mathsf{c}(\mathsf{a},\mathsf{a})\big)$ to two copies of itself after both $\mathsf{a}$-subterms have been rewritten to $\mathsf{c}(\mathsf{a},\mathsf{a})$. This only happens with probability $(5/8)^2 = 25/64 < 1/2$, since the two $\mathsf{a}$-subterms are rewritten independently.

All in all, $\mathcal{A}'$ is $\mathtt{AST}$, while $\mathcal{A}$ is not $\mathtt{AST}$, i.e., $\mathrm{Proc}_{\mathtt{roi}}$ is unsound for $\mathtt{AST}$.

To see why $\text{Proc}_{\texttt{roi}}$ is unsound for AST, note that after applying an ADP, the next rewrite step can be "completely inside" the used substitution if one does not impose an innermost rewriting strategy. So for the "non-AST" reduction in Counterexample 7.6.21, after using the (original) f-ADP to rewrite $\mathsf{f}^\sharp(\mathsf{c}(\mathsf{a},\mathsf{a}))$ to $\mathsf{c}(\mathsf{f}^\sharp(\mathsf{a}),\mathsf{f}^\sharp(\mathsf{a}))$, the next "inner" rewrite step rewrites the a-subterms which did not occur in the right-hand side of the f-ADP but which were "completely inside" the substitution used for $x$. However, if we only consider the right-hand side of the f-ADP, as it does not contain any a-symbol, we only note that after one application of the f-ADP we eventually have to use the f-ADP again, leading to the narrowing substitution $\delta$ in Counterexample 7.6.21. But when instantiating the f-ADP with this substitution, in order to simulate the previous reduction with the new f-ADP, we now need two instead of one "inner" rewrite step for the a-subterms. In the probabilistic setting, this is a problem if the critical rewrite step (e.g., from a to $\mathsf{c}(\mathsf{a},\mathsf{a})$) is only chosen with a probability $< 1$, because the requirement of choosing the critical step two times has a lower probability than only choosing it once.

SECTION 7.7

## All Processors at One Glance

Table 7.1 shows all processors of the ADP framework for AST at one glance. For each processor, we give its name, its formal definition Proc, its *status*, and a short explanation of its underlying idea. Here, the status of a processor is depicted as a $2 \times 2$ grid whose columns correspond to full rewriting (AST) and innermost rewriting (iAST), and whose rows correspond to *soundness* (top) and *completeness* (bottom). A green entry S and C indicates that the processor is sound and complete for the corresponding evaluation strategy (under the conditions given in the explanation), whereas a red, struck-through entry ~~S~~ and ~~C~~ indicates that the respective property does not hold. For the processors that are only defined for innermost evaluation, a struck-through ~~S~~ in the AST column indicates that their natural adaptation to full rewriting is unsound. An entry n/a indicates that the property is not applicable, since the processor is not defined for this evaluation strategy.

SECTION 7.8

## Evaluation

As in Section 4.5, we implemented all techniques of this chapter in AProVE and applied them to all 199 PTRSs from the *Termination Problem Database* [TPDB]. All experiments were performed on a machine with 16 GB of RAM and a timeout of 30 seconds for each example.

To isolate the framework, in this experiment AProVE proves AST *exclusively* via the ADP framework of this chapter, without any direct interpretations or the decision procedure of Chapter 6. We consider both innermost AST (iAST) and full AST: for innermost evaluation the ADP framework always applies, whereas for full evaluation it applies directly (i.e., without first switching to innermost evaluation) only if the PTRS is ND. In this setting, AProVE proves iAST for 144 (72.4%) and full AST for 60 (30.2%) of the 199 PTRSs.

To assess the contribution of each processor, we re-ran both experiments with each

Table 7.1: Overview of the processors in the probabilistic ADP framework for AST.

| Processor | Definition of Proc | Status AST/iAST | Explanation |
|---|---|---|---|
| Chain Criterion (Section 7.2) | $\mathcal{P} \mapsto \mathcal{DP}(\mathcal{P})$ | S S / C C | Reduces proving AST or iAST of a PTRS to proving the corresponding property for its canonical ADP problem. For AST, soundness requires ND PTRSs. |
| Removing Irrelevant ADPs (Section 7.3) | $\mathrm{Proc}^{\mathtt{i}}_{\mathtt{irr}}(\mathcal{A}) = \{(\mathcal{A} \setminus \mathit{Irr}(\mathcal{A})) \cup \{\ell \xhookrightarrow{\mathtt{f}} \{^1\mathsf{stop}\} \mid \ell \xhookrightarrow{\mathtt{f}} \mu \in \mathit{Irr}(\mathcal{A})\}\}$ | ~~S~~ S / n/a C | Replaces ADPs with flag f and no annotations by trivially terminating ones to preserve the normal forms. |
| Dependency Graph Processor (Section 7.4.1) | $\mathrm{Proc}_{\mathtt{DG}}(\mathcal{A}) = \{\mathcal{A}_1 \cup \flat(\mathcal{A} \setminus \mathcal{A}_1), \dots, \mathcal{A}_n \cup \flat(\mathcal{A} \setminus \mathcal{A}_n)\}$ <br> $\mathrm{Proc}^{\mathtt{i}}_{\mathtt{DG}}(\mathcal{A}) = \{\mathcal{A}_1 \cup \flat(\mathcal{A} \setminus \mathcal{A}_1), \dots, \mathcal{A}_n \cup \flat(\mathcal{A} \setminus \mathcal{A}_n)\}$ | S S / C C | Keeps only the annotations of one SCC of the (innermost) $\mathcal{A}$-dependency graph per subproblem. |
| Usable Terms Processor (Section 7.4.2) | $\mathrm{Proc}_{\mathtt{UT}}(\mathcal{A}) = \{\mathcal{T}_{\mathtt{UT}}(\mathcal{A})\}$ <br> $\mathrm{Proc}^{\mathtt{i}}_{\mathtt{UT}}(\mathcal{A}) = \{\mathcal{T}^{\mathtt{i}}_{\mathtt{UT}}(\mathcal{A})\}$ | S S / C C | Removes annotations from right-hand-side subterms that can never contribute to an annotated recursive step. |
| Usable Rules Processor (Section 7.4.3) | $\mathrm{Proc}^{\mathtt{i}}_{\mathtt{UR}}(\mathcal{A}) = \{\mathcal{U}(\mathcal{A}) \cup \{\ell \xhookrightarrow{\mathtt{f}} \mu \mid \ell \xhookrightarrow{m} \mu \in \mathcal{A} \setminus \mathcal{U}(\mathcal{A})\}\}$ | ~~S~~ S / ~~C~~ C | Detects rules that are irrelevant below annotations and changes their flag to f. |
| Probability Removal Processor (Section 7.4.4) | $\mathrm{Proc}_{\mathtt{PR}}(\mathcal{A}) = \mathrm{Proc}^{\mathtt{i}}_{\mathtt{PR}}(\mathcal{A}) = \varnothing$ if $(\mathrm{dp}(\mathcal{A}), \mathrm{np}(\mathcal{A}))$ is (i)SN | S S / C C | Applies when every ADP has non-probabilistic structure and transfers the task to a non-probabilistic DP problem. |
| Potential Switch from AST to iAST (Section 7.4.5) | $\mathcal{A} \mapsto \mathcal{A}$ (prove iAST instead of AST) | ~~S~~ n/a / C n/a | Unsound for ADP problems. Completeness is not relevant for practice. |
| Switching Back to PTRSs (Section 7.4.6) | $\mathrm{Proc}_{\mathtt{back}}(\mathcal{A}) = \mathrm{Proc}^{\mathtt{i}}_{\mathtt{back}}(\mathcal{A}) = \varnothing$ if $\mathcal{DP}^{-1}(\mathcal{A})$ is (i)AST | S S / C C | Switches back to the underlying PTRS $\mathcal{DP}^{-1}(\mathcal{A})$ to use, e.g., our decision procedure when $\mathcal{DP}^{-1}(\mathcal{A})$ is RNT. |
| Reduction Pair Processor (Section 7.5.1) | $\mathrm{Proc}_{\mathtt{RP}}(\mathcal{A}) = \mathrm{Proc}^{\mathtt{i}}_{\mathtt{RP}}(\mathcal{A}) = \{\mathcal{A}_{\succcurlyeq} \cup \flat(\mathcal{A}_{\succ})\}$ | S S / C C | Splits the ADP problem into weakly and strictly oriented rules and removes the annotations from the strict part. |
| Subterm Criterion (Section 7.5.2) | $\mathrm{Proc}^{\mathtt{i}}_{\mathtt{SC}}(\mathcal{A}) = \{(\mathcal{A} \setminus \mathcal{A}_{\triangleright}) \cup \flat(\mathcal{A}_{\triangleright})\}$ | ~~S~~ S / ~~C~~ C | Compares annotated left-hand sides with annotated subterms on right-hand sides w.r.t. the subterm relation. |
| Rewriting Processor (Section 7.6.1) | $\mathrm{Proc}^{\mathtt{i}}_{\mathtt{r}}(\mathcal{A}) = \{\mathcal{A}' \cup \{\ell \xhookrightarrow{m} \{^{p_1}\flat(r_1), \dots, {}^{p_k}\flat(r_k)\}, \ell \xhookrightarrow{m} (\{^{p_1}r_1, \dots, {}^{p_k}r_k\} \setminus \{^{p_j}r_j\} \cup \{^{p_j \cdot q_1}e_1, \dots, {}^{p_j \cdot q_h}e_h\})\}\}$ | n/a S / n/a ~~C~~ | Performs a concrete rewrite step directly inside a right-hand side of an ADP and replaces the old ADP by a rewritten variant plus its non-annotated copy. |
| Instantiation Processor (Section 7.6.2) | $\mathrm{Proc}_{\mathtt{ins}}(\mathcal{A}) = \{\mathcal{A}' \cup M \cup \{\ell \xhookrightarrow{m} \flat(\mu)\}\}$ <br> $\mathrm{Proc}^{\mathtt{i}}_{\mathtt{ins}}(\mathcal{A}) = \{\mathcal{A}' \cup M \cup \{\ell \xhookrightarrow{m} \flat(\mu)\}\}$ | S S / C C | Instantiates one ADP using overlaps with possible predecessors, making the recursive call pattern more concrete. For AST, completeness requires a linear ADP. |
| Forward Instantiation Processor (Section 7.6.3) | $\mathrm{Proc}_{\mathtt{fins}}(\mathcal{A}) = \{\mathcal{A}' \cup M \cup \{\ell \xhookrightarrow{m} \flat(\mu)\}\}$ <br> $\mathrm{Proc}^{\mathtt{i}}_{\mathtt{fins}}(\mathcal{A}) = \{\mathcal{A}' \cup M \cup \{\ell \xhookrightarrow{m} \flat(\mu)\}\}$ | S S / C C | Like instantiation, but using possible successors. For AST, defined symbols are capped as well, and completeness requires a linear ADP. |
| Rule Overlap Instantiation Processor (Section 7.6.4) | $\mathrm{Proc}^{\mathtt{i}}_{\mathtt{roi}}(\mathcal{A}) = \{\mathcal{A}' \cup M\}$ | ~~S~~ S / ~~C~~ C | Instantiates one ADP by all narrowing substitutions caused by rule overlaps, but unlike classical narrowing it does not already execute the rewrite step. |

processor disabled in turn. Table 7.2 lists, for each disabled processor, the number of `iAST` and full-`AST` proofs that remain (as a percentage of the 199 benchmarks).

| Disabled processor | `iAST` proofs | full `AST` proofs |
|---|---|---|
| – (full framework) | 144 (72.4%) | 60 (30.2%) |
| reduction pair | 11 (5.5%) | 1 (0.5%) |
| usable rules | 110 (55.3%) | 60 (30.2%) |
| rewriting | 123 (61.8%) | 60 (30.2%) |
| usable terms | 124 (62.3%) | 58 (29.1%) |
| rule overlap instantiation | 129 (64.8%) | 60 (30.2%) |
| forward instantiation | 143 (71.9%) | 60 (30.2%) |
| dependency graph | 142 (71.4%) | 59 (29.6%) |
| probability removal | 141 (70.9%) | 60 (30.2%) |
| instantiation | 144 (72.4%) | 60 (30.2%) |
| subterm criterion | 144 (72.4%) | 60 (30.2%) |

Table 7.2: Evaluation of the processors of the ADP framework for `AST`.

For *innermost* evaluation, the *reduction pair processor* is by far the most important technique: without it, only 11 of the 144 proofs remain, since it is the only processor that can actually orient and remove annotated dependency pairs via (polynomial) interpretations. The *usable rules* and *usable terms* processors, the *rewriting* processor, and the *rule overlap instantiation* processor each contribute substantially, as they simplify the ADP problem so that the reduction pair processor becomes applicable more often. The remaining processors each account for a few additional proofs, except for the *instantiation* processor and the *subterm criterion*, which yield no additional proofs on this benchmark set (the remaining processors also solve all examples that they solve). For *full* evaluation the reduction pair processor is again essential (only a single proof survives without it). Apart from the reduction pair processor, only two *structural* processors affect the number of proofs for full evaluation: disabling the *dependency graph* processor loses one proof, and disabling the *usable terms* processor loses two. Hence, even though the reduction pair processor dominates, the structural processors are not only an optimization but can be necessary to obtain a proof at all. All processors moreover remain valuable in practice, as they are often much cheaper than the reduction pair processor and thereby speed up the overall proof search.

SECTION 7.9

## Related Work & Open Problems

The DP framework is one of the most powerful techniques to analyze termination of TRSs. Therefore, DPs are used in essentially all current termination and complexity tools for TRSs, e.g., AProVE [GAB+17], MuTerm [GL20b], NaTT [YKS14], TcT [AMS16], $T_TT_2$ [KSZM09], etc. Dependency pairs have been introduced in [AG00], and the DP framework for ordinary termination of TRSs has been developed and improved in [GTS04, HM04, GTSF06, HM05, HM07]. There exist several related DP frameworks targeting special forms of termination, e.g., for relative termination [KVG24], for associative and commutative rewriting [ALM10], for context-sensitive rewriting [AEF+08, AGL10], for equational rewriting [GK01], for higher-order rewriting [Bla06, Kv12, FK19], etc. In addition to termination, dependency pairs are also

successfully used for complexity analysis. In [NEG13] a DP framework for innermost runtime complexity of TRSs has been presented. Additionally, *weak* dependency pairs were introduced in [HM08] and are used in [AM16] in order to analyze derivational and runtime complexity. Apart from DP-based techniques, an induction-based method for proving PAST of probabilistic rewrite systems under strategies was proposed in [Gna07].

To allow certification of proofs with DPs, they have been formalized in several proof assistants (e.g., in Rocq (formerly Coq) [CCF+07, BK11], Isabelle [TS09a], and recently together with the size-change principle [LJB01, TG05, MV06] in PVS [AA20, MAM+23]), and there exist several corresponding certification tools for termination and complexity proofs with DPs (e.g., CeTA [TS09a]).

Compared to our first adaptation of DPs to PTRSs via dependency tuples (DTs) in [KG23], the ADP framework presented in this chapter has several advantages: it is easier, more elegant, and yields a complete chain criterion. Additionally, it also has important practical advantages, because every processor that performs a rewrite step benefits from our novel definition of rewriting with ADPs (whereas the rewrite relation with DTs in [KG23] was an "incomplete over-approximation" of the rewrite relation of the original TRS).

Apart from the obvious future work to develop further processors for the ADP framework, we want to mention one more subtle open problem regarding the framework.

> Open Problem 7.9.1 – Find the Best Strategy.
>
> Optimize the strategy that determines when to apply which processor in the ADP framework.

It is an interesting idea to utilize artificial intelligence to optimize the current strategy developed by human experts.

# 8 Proving Strong Almost-Sure Termination

Dependency pairs can not only be used to prove termination of TRSs, but they can also be used to analyze the runtime complexity of TRSs. However, instead of considering each function call individually as for termination of TRSs, one has to consider all of them simultaneously. Our adaptation of dependency pairs for AST, namely annotated dependency pairs, already considered all the recursive function calls simultaneously. In this chapter, we show that ADPs can also be used to analyze the expected runtime of PTRSs. To be precise, we combine our ADP framework from Chapter 7 with the dependency pair framework for runtime complexity of [NEG13]. This yields the first modular framework for the analysis of the expected innermost runtime complexity of PTRSs, and hence of SAST of innermost evaluation w.r.t. basic start terms (for imperative probabilistic programs, a modular expected cost analysis was developed in [AMS20]). For brevity, we restrict ourselves to a selection of the most important processors, and the processors that change the most when moving from AST to SAST. As in Chapter 7, we assume that the signature $\Sigma$ and the decomposition $\Sigma = \mathcal{D} \uplus \mathcal{C}$ into defined symbols and constructors is fixed.

As in [NEG13], we have to restrict the analysis to innermost runtime complexity, i.e., the bounds we derive only hold for innermost evaluations that start with a basic term. However, under the respective conditions of Chapter 4, we can reuse the relations and transformations from that chapter to lift the results to derivational complexity and to other evaluation strategies. Throughout this chapter, we abbreviate the expected innermost runtime complexity of a PTRS $\mathcal{P}$ by $\iota_{\mathcal{P}} = \iota(\mathrm{erc}_{\xrightarrow{\mathtt{i}}_{\mathcal{P}}})$.

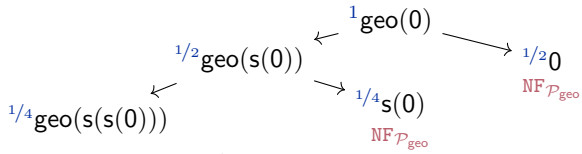



Figure 8.1: The $\xrightarrow{\mathtt{i}}_{\mathcal{P}_{\mathsf{geo}}}$-RT starting with $\mathsf{geo}(0)$.

We use the following examples as running examples for the first sections of this chapter.

Example 8.0.1. Consider the following PTRS $\mathcal{P}_{\mathsf{quot}}$ containing the five rules

$$\mathcal{P}_{\mathsf{start}}: \quad \begin{aligned} \mathsf{start}(x,y) &\to \{{}^{1}\mathsf{q}(\mathsf{geo}(x),y,y)\} \\ \mathsf{geo}(x) &\to \{{}^{1/2}\mathsf{geo}(\mathsf{s}(x)),\ {}^{1/2}x\} \end{aligned} \qquad \mathcal{P}_{\mathsf{q}}: \quad \begin{aligned} \mathsf{q}(\mathsf{s}(x),\mathsf{s}(y),z) &\to \{{}^{1}\mathsf{q}(x,y,z)\} \\ \mathsf{q}(x,0,\mathsf{s}(z)) &\to \{{}^{1}\mathsf{s}(\mathsf{q}(x,\mathsf{s}(z),\mathsf{s}(z)))\} \\ \mathsf{q}(0,\mathsf{s}(y),\mathsf{s}(z)) &\to \{{}^{1}0\}. \end{aligned}$$

When starting with $\mathsf{start}(\mathsf{s}^n(0), \mathsf{s}^m(0))$, $\mathcal{P}_{\mathsf{quot}}$ computes $\lfloor \frac{n+\mathsf{geo}(0)}{m} \rfloor$, i.e., it first increases $n$ according to a geometric distribution (rules $\mathcal{P}_{\mathsf{start}}$ on the left), and then computes the quotient (rules $\mathcal{P}_{\mathsf{q}}$ on the right). Thus, $\iota_{\mathcal{P}_{\mathsf{quot}}} = \mathbf{Pol}_1$, since $\mathcal{P}_{\mathsf{q}}$ has linear runtime complexity and the geometric distribution only increases $n$ by 1 in expectation. So in particular, innermost evaluation of $\mathcal{P}_{\mathsf{quot}}$ is SAST w.r.t. basic start terms.

Example 8.0.2. As another example, consider the PTRS $\mathcal{P}_{\mathsf{exp}}$ with the four rules

$$\mathsf{start} \to \{^1\mathsf{f}(\mathsf{geo}(0))\} \qquad \mathsf{f}(\mathsf{s}(x)) \to \{^1\mathsf{f}(\mathsf{c}(x,x))\}$$
$$\mathsf{geo}(x) \to \{^{1/2}\mathsf{geo}(\mathsf{s}(x)),\ ^{1/2}x\} \qquad \mathsf{f}(\mathsf{c}(x,y)) \to \{^1\mathsf{c}(\mathsf{f}(x),\mathsf{f}(y))\}.$$

The two f-rules have exponential runtime complexity, as a reduction starting in $\mathsf{f}(\mathsf{s}^n(0))$ creates a full binary tree of height $n$ and visits every inner node once. When beginning with the term start, $\mathcal{P}_{\mathsf{exp}}$ first generates the term $\mathsf{f}(\mathsf{s}^k(0))$ with probability $(1/2)^{k+1}$ and then takes at least $2^k$ steps to terminate. The expected derivation length of the corresponding RT is at least $\sum_{k=0}^{\infty} (1/2)^{k+1} \cdot 2^k = \sum_{k=0}^{\infty} 1/2 = \infty$. Hence, $\iota_{\mathcal{P}_{\mathsf{exp}}} = \mathbf{Inf}$, i.e., $\mathcal{P}_{\mathsf{exp}}$ is not SAST.

### Contributions

We present the following contributions in this chapter. The results marked with !★! are novel, while the remaining contributions can be found in the following peer-reviewed publication: ★ [KSG25].

- ★ Adjusting the *ADP framework* for analyzing the expected innermost runtime complexity of PTRSs (and hence SAST w.r.t. basic start terms), using processors to decompose problems into simpler sub-problems (Section 8.1).
- ★ The *reduction pair processor*, using barycentric algebras to derive complexity bounds by comparing left-hand sides to the value of the annotated subterms in the corresponding right-hand sides (Section 8.3.1).
- ★ The *dependency graph processor*, adapted from [NEG13], which splits problems based on strongly connected components of the dependency graph and their prefixes for modular analysis (Section 8.2.2).
- ★ The *knowledge propagation processor*, adapted from [NEG13], which removes ADPs from the set of counted rules when all their predecessors in the dependency graph have already been accounted for (Section 8.2.3).
- ★ The *usable rules processor*, which detects rules that cannot be used to evaluate the arguments of annotated symbols and sets their flag to f (Section 8.2.1).
- !★! The *rule overlap instantiation processor*, which handles overlaps between rules by instantiating variables (Section 8.4).

For brevity, we only introduce the most important processors and the processors that changed substantially compared to the ADP framework for AST.

## Structure

This chapter follows the same structure as Chapter 7 regarding the ADP framework for AST. We first present the general idea of *annotated dependency pairs* and the *ADP framework* for SAST in Section 8.1. Afterwards, we present the different *processors* that are usable within the framework. We first present *structural* processors in Section 8.2, followed by processors based on algebras in Section 8.3, followed by *transformational* processors in Section 8.4. In Section 8.5 we give a table containing the most important information for each processor. We end this chapter by performing an experimental evaluation in Section 8.6, and discussing related work and giving important open problems for future work in Section 8.7.

SECTION 8.1

# The ADP Framework for SAST

In this section we present the *annotated dependency pair framework* for upper bounds on the expected innermost runtime complexity (and hence for SAST of innermost evaluation w.r.t. basic start terms). The basic definitions regarding annotations, (canonical) ADPs, and rewriting with ADPs are as in Chapter 7. Only the chains and the ADP problems themselves differ slightly.

Example 8.1.1 (Canonical Annotated Dependency Pairs). Reconsider $\mathcal{P}_{\mathsf{quot}}$ and $\mathcal{P}_{\mathsf{exp}}$ from Examples 8.0.1 and 8.0.2. The canonical ADPs $\mathcal{DP}(\mathcal{P}_{\mathsf{quot}})$ for $\mathcal{P}_{\mathsf{quot}}$ are

$$\mathsf{start}(x,y) \hookrightarrow^{\mathsf{t}} \{{}^{1}\mathsf{q}^{\sharp}(\mathsf{geo}^{\sharp}(x),y,y)\} \tag{8.1}$$
$$\mathsf{geo}(x) \hookrightarrow^{\mathsf{t}} \{{}^{1/2}\mathsf{geo}^{\sharp}(\mathsf{s}(x)), {}^{1/2}x\} \tag{8.2}$$
$$\mathsf{q}(\mathsf{s}(x),\mathsf{s}(y),z) \hookrightarrow^{\mathsf{t}} \{{}^{1}\mathsf{q}^{\sharp}(x,y,z)\} \tag{8.3}$$
$$\mathsf{q}(x,0,\mathsf{s}(z)) \hookrightarrow^{\mathsf{t}} \{{}^{1}\mathsf{s}(\mathsf{q}^{\sharp}(x,\mathsf{s}(z),\mathsf{s}(z)))\} \tag{8.4}$$
$$\mathsf{q}(0,\mathsf{s}(y),\mathsf{s}(z)) \hookrightarrow^{\mathsf{t}} \{{}^{1}0\}. \tag{8.5}$$

Example 8.1.2 (Canonical Annotated Dependency Pairs, Continued). The canonical ADPs $\mathcal{DP}(\mathcal{P}_{\mathsf{exp}})$ for $\mathcal{P}_{\mathsf{exp}}$ from Example 8.0.2 are

$$\mathsf{start} \hookrightarrow^{\mathsf{t}} \{{}^{1}\mathsf{f}^{\sharp}(\mathsf{geo}^{\sharp}(0))\} \tag{8.6}$$
$$\mathsf{geo}(x) \hookrightarrow^{\mathsf{t}} \{{}^{1/2}\mathsf{geo}^{\sharp}(\mathsf{s}(x)), {}^{1/2}x\} \tag{8.7}$$
$$\mathsf{f}(\mathsf{s}(x)) \hookrightarrow^{\mathsf{t}} \{{}^{1}\mathsf{f}^{\sharp}(\mathsf{c}(x,x))\} \tag{8.8}$$
$$\mathsf{f}(\mathsf{c}(x,y)) \hookrightarrow^{\mathsf{t}} \{{}^{1}\mathsf{c}(\mathsf{f}^{\sharp}(x),\mathsf{f}^{\sharp}(y))\}. \tag{8.9}$$

Chain trees are defined as in Definition 7.2.12, but with innermost steps only and without the liveness property.

**Definition 8.1.3** – Chain Tree (for SAST).

*Let $\mathcal{A}$ be a set of ADPs. A labeled tree $\mathfrak{T} = (N, E)$ is an $\mathcal{A}$-chain tree (CT) if the nodes $v \in N$ are labeled by pairs ${}^{p_v}t_v$ of a non-zero probability $0 < p_v \leq 1$ and a term $t_v \in \mathcal{T}^{\sharp}$, where the root $\mathfrak{r}$ is always labeled with the probability $p_{\mathfrak{r}} = 1$. For each inner node $v \in N$ with the successors $w_1, \ldots, w_k \in N$, the edge relation represents a reduction step with the relation $\hookrightarrow^{\mathsf{i}}_{\mathcal{A}}$, i.e., $t_v \hookrightarrow^{\mathsf{i}}_{\mathcal{A}} \{{}^{p_{w_1}/p_v}t_{w_1}, \ldots, {}^{p_{w_k}/p_v}t_{w_k}\}$. Moreover, every inner node $v$ is additionally labeled with the pair $\mathrm{Step}(v) \in \mathcal{A} \times \{\mathbf{at}, \mathbf{af}, \mathbf{nt}, \mathbf{nf}\}$ of the ADP and the kind of step used for rewriting $t_v$.*

The $\mathcal{DP}(\mathcal{P}_{\mathsf{geo}})$-CT in Figure 8.2 corresponds to the $\to^{\mathsf{i}}_{\mathcal{P}_{\mathsf{geo}}}$-RT in Figure 8.1, where $\mathcal{P}_{\mathsf{geo}}$ is the PTRS from Example 3.4.3 that consists only of the geo-rule.

In contrast to the *expected derivation length* of RTs, for the expected derivation length

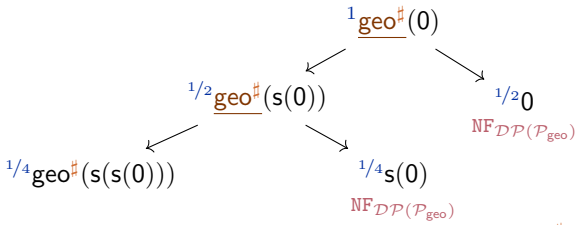



Figure 8.2: $\mathcal{DP}(\mathcal{P}_{\mathsf{geo}})$-chain tree starting with $\mathsf{geo}^\sharp(0)$.

of CTs, we only consider **at**- or **af**-steps, i.e., steps at the position of an annotated symbol. Since **nt**- and **nf**-steps are disregarded for the expected derivation length of CTs, in contrast to the chain trees used for proving `AST` in Chapter 7, we do not have to require the liveness property, i.e., that every infinite path contains infinitely many **at**- or **af**-steps, anymore. Moreover, sometimes we do not want to count the application of *all* ADPs, but only the ADPs from some subset. Thus, similar to the adaptation of DPs for complexity analysis in [NEG13], in our ADP framework we do not consider a single set of ADPs $\mathcal{A}$, but we use a second set $\mathcal{S} \subseteq \mathcal{A}$ of those ADPs that (still) have to be taken into account. So one only has to add the probabilities of **a**-steps with ADPs from $\mathcal{S}$ in the chain tree to determine its *expected derivation length*.

Definition 8.1.4 – ADP Problem.
*Let $\mathcal{A}$ and $\mathcal{S}$ be finite sets of ADPs where $\mathcal{S} \subseteq \mathcal{A}$. The pair $\langle \mathcal{A}, \mathcal{S} \rangle$ is called an ADP problem. An ADP problem with $\mathcal{S} = \varnothing$ is called solved.*

Remark 8.1.5
With a slight abuse of notation we refer to the pair $\langle \mathcal{A}, \mathcal{S} \rangle$ as an ADP problem (for `SAST` and expected complexity), and we refer to a single set of ADPs $\mathcal{A}$ as an ADP problem (for `AST`) as well. Several other names from Chapter 7 are overloaded in the same way in this chapter. However, it should always be clear from the context, which ADP framework (for `SAST` or `AST`) we are referring to.

Initially, we have $\mathcal{S} = \mathcal{A}$. Our analysis ends once all of the ADP problems are solved, i.e., once we have $\mathcal{S} = \varnothing$. In the remainder, we fix an ADP problem $\langle \mathcal{A}, \mathcal{S} \rangle$.

Definition 8.1.6 – Expected Derivation Length for Chain Trees.
*Let $\mathfrak{T} = (N, E)$ be an $\mathcal{A}$-chain tree. The expected derivation length of $\mathfrak{T}$, where we only count steps with $\mathcal{S}$ at annotated symbols, is*

$$\mathrm{edl}_{\langle \mathcal{A}, \mathcal{S} \rangle}(\mathfrak{T}) = \sum_{v \in N \setminus \mathrm{Leaf}^{\mathfrak{T}},\ \mathrm{Step}(v) \in \mathcal{S} \times \{\mathbf{at}, \mathbf{af}\}} p_v.$$

Example 8.1.7 (Expected Derivation Length for Chain Trees). Reconsider the PTRS $\mathcal{P}_{\mathsf{quot}}$ and the $\mathcal{DP}(\mathcal{P}_{\mathsf{quot}})$-chain tree $\mathfrak{T}$ depicted in Figure 8.3. Let $\mathcal{S}_{\mathsf{geo}} = \{(8.2)\}$ contain only the geo-ADP and let $\mathcal{S}_{\mathsf{q}} = \{(8.3), (8.4), (8.5)\}$ contain the q-ADPs. Computing

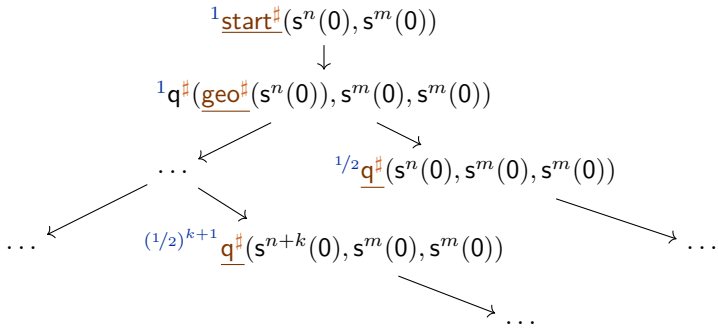



Figure 8.3: $\mathcal{DP}(\mathcal{P}_{\mathsf{quot}})$-chain tree $\mathfrak{T}$.

the expected derivation length of $\mathfrak{T}$ w.r.t. $\mathcal{S}_{\mathsf{geo}}$ or w.r.t. all ADPs results in

$$\begin{aligned}
\mathrm{edl}_{\langle\mathcal{DP}(\mathcal{P}_{\mathsf{quot}}),\mathcal{S}_{\mathsf{geo}}\rangle}(\mathfrak{T}) &= 1 + 1/2 + 1/4 + 1/8 + \ldots = 1 + \textstyle\sum_{k=1}^{\infty} 1/2^k = 2\\
\mathrm{edl}_{\langle\mathcal{DP}(\mathcal{P}_{\mathsf{quot}}),\mathcal{DP}(\mathcal{P}_{\mathsf{quot}})\rangle}(\mathfrak{T}) &= 1 + \mathrm{edl}_{\langle\mathcal{DP}(\mathcal{P}_{\mathsf{quot}}),\mathcal{S}_{\mathsf{geo}}\rangle}(\mathfrak{T}) + \mathrm{edl}_{\langle\mathcal{DP}(\mathcal{P}_{\mathsf{quot}}),\mathcal{S}_{\mathsf{q}}\rangle}(\mathfrak{T})\\
&\leq 1 + 2 + \textstyle\sum_{k=0}^{\infty} (1/2)^{k+1} \cdot (2(n+k)+2)\\
&= 3 + \textstyle\sum_{k=0}^{\infty} k/2^k + \sum_{k=0}^{\infty} (1/2)^k \cdot (n+1)\\
&= 3 + 2 + 2 \cdot (n+1) = 2n + 7.
\end{aligned}$$

Next, we define *expected derivation height* via chain trees by considering all possible CTs with the root $t^\sharp$ for a basic term $t$, and taking the supremum of their expected derivation lengths.

Definition 8.1.8 – Expected Derivation Height via Chain Trees.

*Let $\langle\mathcal{A},\mathcal{S}\rangle$ be an ADP problem. For $t \in \mathcal{T}_{\mathcal{B}}$, the expected derivation height $\mathrm{edh}_{\langle\mathcal{A},\mathcal{S}\rangle}(t) \in \mathbb{R}_{\geq 0} \cup \{\infty\}$ is the supremum obtained when adding all probabilities for **a**-steps with $\mathcal{S}$ in any chain tree $\mathfrak{T}$ with root $t^\sharp$:*

$$\mathrm{edh}_{\langle\mathcal{A},\mathcal{S}\rangle}(t) = \sup\{\mathrm{edl}_{\langle\mathcal{A},\mathcal{S}\rangle}(\mathfrak{T}) \mid \mathfrak{T} \text{ is an } \mathcal{A}\text{-CT with } \mathrm{root}(\mathfrak{T}) = t^\sharp\}.$$

Example 8.1.9 (Expected Derivation Height via Chain Trees). Consider the term $t = \mathsf{start}(\mathsf{s}^n(0),\mathsf{s}^m(0))$ and its corresponding $\mathcal{DP}(\mathcal{P}_{\mathsf{quot}})$-CT from Example 8.1.7. As this is the only maximal $\mathcal{DP}(\mathcal{P}_{\mathsf{quot}})$-CT with root $t^\sharp$, we obtain $\mathrm{edh}_{\langle\mathcal{DP}(\mathcal{P}_{\mathsf{quot}}),\mathcal{DP}(\mathcal{P}_{\mathsf{quot}})\rangle}(t) \leq 2n+7$ and $\mathrm{edh}_{\langle\mathcal{DP}(\mathcal{P}_{\mathsf{quot}}),\mathcal{S}_{\mathsf{geo}}\rangle}(t) = 2$.

Now we can define expected runtime complexity for ADP problems.

Definition 8.1.10 – Expected Runtime Complexity for ADP Problems.

*Let $\langle\mathcal{A},\mathcal{S}\rangle$ be an ADP problem. The expected runtime complexity function $\mathrm{eirc}_{\langle\mathcal{A},\mathcal{S}\rangle} : \mathbb{N} \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ of $\langle\mathcal{A},\mathcal{S}\rangle$ is defined as*

$$\mathrm{eirc}_{\langle\mathcal{A},\mathcal{S}\rangle}(n) = \sup\{\mathrm{edh}_{\langle\mathcal{A},\mathcal{S}\rangle}(t) \mid t \in \mathcal{T}_{\mathcal{B}}, |t| \leq n\},$$

*and we define the (asymptotic) expected runtime complexity $\iota_{\langle\mathcal{A},\mathcal{S}\rangle}$ of $\langle\mathcal{A},\mathcal{S}\rangle$ as $\iota_{\langle\mathcal{A},\mathcal{S}\rangle} = \iota(\mathrm{eirc}_{\langle\mathcal{A},\mathcal{S}\rangle})$.*

Example 8.1.11 (Expected Runtime Complexity for ADP Problems). For a basic term

$\mathsf{start}(t_1, t_2)$, $\mathcal{DP}(\mathcal{P}_{\mathsf{quot}})$ first computes a geometric distribution starting in $t_1$. This needs 2 steps in expectation, and increases $t_1$ by only 1 in expectation. In the resulting term $\mathsf{q}(t_{\mathsf{geo}}, t_2, t_2)$, where $t_{\mathsf{geo}}$ is the normal form resulting from $\mathsf{geo}(t_1)$, we decrease $t_{\mathsf{geo}}$ until we reach $\mathsf{0}$. Therefore, the expected derivation height is linear in the size of the start term, i.e., $\iota_{\langle \mathcal{DP}(\mathcal{P}_{\mathsf{quot}}), \mathcal{DP}(\mathcal{P}_{\mathsf{quot}})\rangle} = \mathbf{Pol}_1$. If we only consider $\mathcal{S}_{\mathsf{geo}}$ for the complexity, then $\iota_{\langle \mathcal{DP}(\mathcal{P}_{\mathsf{quot}}), \mathcal{S}_{\mathsf{geo}}\rangle} = \mathbf{Pol}_0$.

With our new concepts, we obtain the following *chain criterion* for complexity analysis of PTRSs. It shows that to analyze the expected runtime complexity of a PTRS $\mathcal{P}$, it suffices to analyze the expected runtime complexity of its *canonical ADP problem* $\langle \mathcal{DP}(\mathcal{P}), \mathcal{DP}(\mathcal{P})\rangle$, i.e., in the beginning all ADPs are considered for complexity. In the canonical ADP problem, all defined symbols in right-hand sides are annotated. Thus, one can only perform **at**-steps: due to innermost evaluation, annotations are only removed from subterms in normal form. Hence, the rewrite steps with $\mathcal{P}$ and the ones with $\mathcal{DP}(\mathcal{P})$ directly correspond to each other.

Theorem 8.1.12 – Chain Criterion.
Let $\mathcal{P}$ be a finite PTRS. Then for all basic terms $t \in \mathcal{T}_{\mathcal{B}}$ we have

$$\mathrm{edh}_{\xrightarrow{\mathtt{i}}_{\mathcal{P}}}(t) = \mathrm{edh}_{\langle \mathcal{DP}(\mathcal{P}), \mathcal{DP}(\mathcal{P})\rangle}(t)$$

and therefore $\iota_{\mathcal{P}} = \iota_{\langle \mathcal{DP}(\mathcal{P}), \mathcal{DP}(\mathcal{P})\rangle}$.

So in contrast to the chain criterion of [NEG13] for complexity analysis in the non-probabilistic setting, Theorem 8.1.12 yields a *tight* bound ($\iota_{\mathcal{P}} = \iota_{\langle \mathcal{DP}(\mathcal{P}), \mathcal{DP}(\mathcal{P})\rangle}$) for arbitrary finite PTRSs due to the usage of ADPs instead of dependency tuples (with dependency tuples, the complexity of the canonical dependency tuple problem would only be an upper bound on $\iota_{\mathcal{P}}$).

The proof of Theorem 8.1.12 is analogous to the proof of the chain criterion for innermost $\mathtt{AST}$ (Theorem 7.2.16). Given an $\xrightarrow{\mathtt{i}}_{\mathcal{P}}$-RT $\mathfrak{T}$ that starts with the basic term $t$, we create a $\mathcal{DP}(\mathcal{P})$-CT $\mathfrak{T}'$ that starts with $t^\sharp$ and mirrors every rewrite step by using the corresponding ADP for the used rewrite rule. We show that each rewrite step takes place at an annotated position, i.e., we always use case **at** with the ADPs. Since the resulting CT has the same structure as the original RT (same nodes with the same probabilities) and we only have **at**-steps (that we count for the expected derivation length of a CT), we get $\mathrm{edl}(\mathfrak{T}) = \mathrm{edl}_{\langle \mathcal{DP}(\mathcal{P}), \mathcal{DP}(\mathcal{P})\rangle}(\mathfrak{T}')$ for every such RT $\mathfrak{T}$, and hence $\mathrm{edh}_{\xrightarrow{\mathtt{i}}_{\mathcal{P}}}(t) \leq \mathrm{edh}_{\langle \mathcal{DP}(\mathcal{P}), \mathcal{DP}(\mathcal{P})\rangle}(t)$ for every basic term $t$. For the other direction, i.e., $\mathrm{edh}_{\xrightarrow{\mathtt{i}}_{\mathcal{P}}}(t) \geq \mathrm{edh}_{\langle \mathcal{DP}(\mathcal{P}), \mathcal{DP}(\mathcal{P})\rangle}(t)$, we transform every $\mathcal{DP}(\mathcal{P})$-CT $\mathfrak{T}$ into an $\xrightarrow{\mathtt{i}}_{\mathcal{P}}$-RT $\mathfrak{T}'$ by simply removing all occurring annotations. Then, we directly get $\mathrm{edl}(\mathfrak{T}') \geq \mathrm{edl}_{\langle \mathcal{DP}(\mathcal{P}), \mathcal{DP}(\mathcal{P})\rangle}(\mathfrak{T})$, since $\mathrm{edl}(\mathfrak{T}')$ counts all steps of $\mathfrak{T}'$.

Since applying an ADP may remove annotations of normal forms, we use the set of all positions of defined symbols whose subterms are not in normal form. During the construction of the proof of the chain criterion, we show that at least those positions are annotated. Remember that $\mathrm{pos}_{\mathcal{D}\wedge\neg\mathtt{NF}_{\mathcal{P}}}(t) = \{\pi \mid \pi \in \mathrm{pos}_{\mathcal{D}}(t), t|_\pi \notin \mathtt{NF}_{\mathcal{P}}\}$.

*Proof of Theorem 8.1.12.* In the following, we will often implicitly use that for an annotated term $t \in \mathcal{T}^\sharp$, we have $\flat(t) \in \mathtt{ANF}_{\mathcal{P}}$ iff $t \in \mathtt{ANF}_{\mathcal{DP}(\mathcal{P})}$ since a rewrite rule and its corresponding canonical annotated dependency pair have the same left-hand side. *Proof*

► *Proof* ($\mathrm{edh}_{\xrightarrow{\mathtt{i}}_{\mathcal{P}}}(t) \leq \mathrm{edh}_{\langle \mathcal{DP}(\mathcal{P}), \mathcal{DP}(\mathcal{P})\rangle}(t)$, *Soundness*): Let $\mathfrak{T} = (N, E)$ be an $\xrightarrow{\mathtt{i}}_{\mathcal{P}}$-RT

that starts with $t$ for some term $t \in \mathcal{T}_{\mathcal{B}}$. We create a $\mathcal{DP}(\mathcal{P})$-CT $\mathfrak{T}' = (N, E)$ that starts with $t^\sharp$ and mirrors every rewrite step by using the ADP corresponding to the used rewrite rule. Moreover, we ensure that every rewrite step is an **at**-step, and hence, we have $\mathrm{edl}(\mathfrak{T}) = \mathrm{edl}_{\langle \mathcal{DP}(\mathcal{P}), \mathcal{DP}(\mathcal{P})^\sharp \rangle}(\mathfrak{T}')$.
We construct the terms $t'_x$ for the nodes of the $\mathcal{DP}(\mathcal{P})$-CT inductively such that for all inner nodes $x \in N \setminus \mathrm{Leaf}$ with children nodes $xE = \{y_1, \dots, y_k\}$ we have $t'_x \overset{\mathrm{i}}{\hookrightarrow}_{\mathcal{DP}(\mathcal{P})} \{{}^{p_{y_1}/p_x}t'_{y_1}, \dots, {}^{p_{y_k}/p_x}t'_{y_k}\}$ and use Case **at**. Let $X \subseteq N$ be the set of nodes $x$ where we have already defined the corresponding term $t'_x$. During our construction, we ensure that the following property holds for all $x \in X$:

$$t_x \overset{\flat}{=} t'_x \wedge \mathrm{pos}_{\mathcal{D}\wedge\neg\mathtt{NF}_{\mathcal{P}}}(t_x) \subseteq \mathrm{pos}_{\mathcal{D}^\sharp}(t'_x). \tag{8.10}$$

This means that the corresponding term $t_x$ for the node $x$ in $\mathfrak{T}$ has the same structure as the term $t'_x$ in $\mathfrak{T}'$, and additionally, all the redexes in $t_x$ are annotated in $t'_x$.
The root of $\mathfrak{T}'$ is labeled by ${}^{1}\sharp_{\mathcal{D}}(t)$, where $\sharp_{\mathcal{D}}(t) = t^\sharp$ since $t$ is basic. Here, we obviously have $t \overset{\flat}{=} \sharp_{\mathcal{D}}(t)$ and $\mathrm{pos}_{\mathcal{D}\wedge\neg\mathtt{NF}_{\mathcal{P}}}(t) \subseteq \mathrm{pos}_{\mathcal{D}}(t) = \mathrm{pos}_{\mathcal{D}^\sharp}(\sharp_{\mathcal{D}}(t))$. As long as there is still an inner node $x \in X$ such that its successors are not contained in $X$, we do the following. Let $xE = \{y_1, \dots, y_k\}$ be the set of its successors. We need to define the corresponding terms $t'_{y_1}, \dots, t'_{y_k}$ for the nodes $y_1, \dots, y_k$. Since $x$ is not a leaf, we have $t_x \overset{\mathrm{i}}{\to}_{\mathcal{P}} \{{}^{p_{y_1}/p_x}t_{y_1}, \dots, {}^{p_{y_k}/p_x}t_{y_k}\}$. This means that there is a rule $\ell \to \{{}^{p_1}r_1, \dots, {}^{p_k}r_k\} \in \mathcal{P}$, a position $\pi$, and a substitution $\sigma$ such that $t_x|_\pi = \ell\sigma \in \mathtt{ANF}_{\mathcal{P}}$. Furthermore, we have $t_{y_j} = t_x[r_j\sigma]_\pi$ for all $1 \le j \le k$. So the node $y_j$ in $\mathfrak{T}$ is labeled by ${}^{p_x \cdot p_j}t_x[r_j\sigma]_\pi$ for all $1 \le j \le k$.
The corresponding ADP for the rule is $\ell \overset{\mathrm{t}}{\hookrightarrow} \{{}^{p_1}\sharp_{\mathcal{D}}(r_1), \dots, {}^{p_k}\sharp_{\mathcal{D}}(r_k)\}$. Furthermore, $\pi \in \mathrm{pos}_{\mathcal{D}\wedge\neg\mathtt{NF}_{\mathcal{P}}}(t_x) \subseteq \mathrm{pos}_{\mathcal{D}^\sharp}(t'_x)$ and $t_x \overset{\flat}{=} t'_x$ by the induction hypothesis. Hence, we can rewrite $t'_x$ with $\ell \overset{\mathrm{t}}{\hookrightarrow} \{{}^{p_1}\sharp_{\mathcal{D}}(r_1), \dots, {}^{p_k}\sharp_{\mathcal{D}}(r_k)\}$, using the position $\pi$ and the substitution $\sigma$, and Case **at** applies. We get $t'_x \overset{\mathrm{i}}{\hookrightarrow}_{\mathcal{DP}(\mathcal{P})} \{{}^{p_1}t'_{y_1}, \dots, {}^{p_k}t'_{y_k}\}$ with $t'_{y_j} = t'_x[\sharp_{\mathcal{D}}(r_j)\sigma]_\pi$. This means that we have $t_{y_j} \overset{\flat}{=} t'_{y_j}$.
It remains to prove $\mathrm{pos}_{\mathcal{D}\wedge\neg\mathtt{NF}_{\mathcal{P}}}(t_{y_j}) \subseteq \mathrm{pos}_{\mathcal{D}^\sharp}(t'_{y_j})$ for all $1 \le j \le k$. For all $\tau \in \mathrm{pos}_{\mathcal{D}\wedge\neg\mathtt{NF}_{\mathcal{P}}}(t_{y_j}) = \mathrm{pos}_{\mathcal{D}\wedge\neg\mathtt{NF}_{\mathcal{P}}}(t_x[r_j\sigma]_\pi)$ that are orthogonal to or strictly above $\pi$, we have $\tau \in \mathrm{pos}_{\mathcal{D}\wedge\neg\mathtt{NF}_{\mathcal{P}}}(t_x) \subseteq \mathrm{pos}_{\mathcal{D}^\sharp}(t'_x)$ by the induction hypothesis, and all annotations orthogonal to or strictly above $\pi$ remain in $t'_{y_j}$ as they were in $t'_x$. For all positions $\tau \in \mathrm{pos}_{\mathcal{D}\wedge\neg\mathtt{NF}_{\mathcal{P}}}(t_{y_j}) = \mathrm{pos}_{\mathcal{D}\wedge\neg\mathtt{NF}_{\mathcal{P}}}(t_x[r_j\sigma]_\pi)$ that are at or below $\pi$, we know that, due to innermost evaluation, at least the defined root symbol of a term that is not in normal form must be inside $r_j$, and thus $\tau \in \mathrm{pos}_{\mathcal{D}^\sharp}(t'_{y_j})$, as all defined symbols of $r_j$ are annotated in $t'_{y_j} = t'_x[\sharp_{\mathcal{D}}(r_j)\sigma]_\pi$.

▶ *Proof*($\mathrm{edh}^{\mathrm{i}}_{\to_{\mathcal{P}}}(t) \ge \mathrm{edh}_{\langle \mathcal{DP}(\mathcal{P}), \mathcal{DP}(\mathcal{P})^\sharp \rangle}(t)$, *Completeness*): Let $\mathfrak{T} = (N, E)$ be a $\mathcal{DP}(\mathcal{P})$-CT that starts with $t^\sharp$ for some term $t \in \mathcal{T}_{\mathcal{B}}$. We create an $\overset{\mathrm{i}}{\to}_{\mathcal{P}}$-RT $\mathfrak{T}' = (N, E)$ that starts with $t$ and mirrors every rewrite step by using the corresponding original probabilistic rule for the used ADP. Hence, we get $\mathrm{edl}_{\langle \mathcal{DP}(\mathcal{P}), \mathcal{DP}(\mathcal{P})^\sharp \rangle}(\mathfrak{T}) \le \mathrm{edl}(\mathfrak{T}')$, since $\mathrm{edl}(\mathfrak{T}')$ counts all steps of $\mathfrak{T}'$.
For every node $x \in N$, the label of $x$ in $\mathfrak{T}'$ is ${}^{\mathfrak{T}}_{x}\flat(t_x)$, where $t_x$ is the term for the node $x$ in $\mathfrak{T}$, i.e., we remove all annotations. We only have to show that $\mathfrak{T}'$ is indeed a valid RT, i.e., that the edge relation represents valid rewrite steps with $\overset{\mathrm{i}}{\to}_{\mathcal{P}}$, but this follows directly from the fact that if we remove all annotations in the ADP rewrite relation, then we get the ordinary probabilistic term rewrite relation again. *QED*

Like the original DP framework of [GTSF06] and the ADP framework for AST of

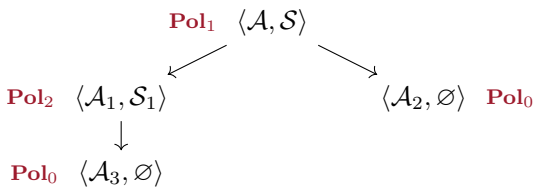



Figure 8.4: Example of a solved proof tree.

Chapter 7, our ADP framework is a *divide-and-conquer* approach which applies *processors* to simplify ADP problems until all sub-problems are solved. As in [NEG13], a processor also returns a complexity $c \in \mathfrak{C}$ that is derived during the application of the processor.

Definition 8.1.13 – ADP Processor (for Complexity).

*An (ADP) processor* Proc *is a function* $\text{Proc}(\langle \mathcal{A}, \mathcal{S} \rangle) = (c, \{\langle \mathcal{A}_1, \mathcal{S}_1 \rangle, \ldots, \langle \mathcal{A}_n, \mathcal{S}_n \rangle\})$ *mapping an ADP problem* $\langle \mathcal{A}, \mathcal{S} \rangle$ *to a complexity* $c \in \mathfrak{C}$ *and a set of ADP problems.*

The intuition for Proc is that in addition to the complexities of $\langle \mathcal{A}_i, \mathcal{S}_i \rangle$, the complexity $c$ is also used to obtain an upper bound on the complexity of $\langle \mathcal{A}, \mathcal{S} \rangle$. During the analysis with our ADP framework, we construct a *proof tree* that contains all sub-problems and complexities resulting from the application of processors.

Definition 8.1.14 – Proof Tree.

*A proof tree is a labeled, finite tree* $(N, E, L_{\mathcal{A}}, L_{\mathcal{C}})$ *with a labeling* $L_{\mathcal{A}}$ *that maps each node to an ADP problem and a second labeling* $L_{\mathcal{C}}$ *that maps each node to a complexity from* $\mathfrak{C}$*. Each edge represents an application of a processor, i.e., if* $v$ *is an inner node with* $vE = \{w_1, \ldots, w_n\}$*, then* $\text{Proc}(L_{\mathcal{A}}(v)) = (L_{\mathcal{C}}(v), \{L_{\mathcal{A}}(w_1), \ldots, L_{\mathcal{A}}(w_n)\})$ *for some processor* Proc*, and we require that the complexity of all leaves* $v$ *is* $L_{\mathcal{C}}(v) = \mathbf{Pol}_0$ *if the corresponding ADP problem* $L_{\mathcal{A}}(v)$ *is solved, and* $L_{\mathcal{C}}(v) = \mathbf{Inf}$ *otherwise. We call a proof tree solved if all ADP problems in its leaves are solved.*

Figure 8.4 shows an example of a solved proof tree where the complexities given by the labeling $L_{\mathcal{C}}$ are depicted next to the nodes. So here, a processor with $\text{Proc}(\langle \mathcal{A}, \mathcal{S} \rangle) = (\mathbf{Pol}_1, \{\langle \mathcal{A}_1, \mathcal{S}_1 \rangle, \langle \mathcal{A}_2, \varnothing \rangle\})$ was used for the step from the root to its two children.

To ensure that the maximum of all complexities in a proof tree is an upper bound on the complexity of the ADP problem at the root, we require that proof trees are *well formed*, i.e., that for every node $v$ with $L_{\mathcal{A}}(v) = \langle \mathcal{A}, \mathcal{S} \rangle$, the expected complexity of $\langle \mathcal{A}, \mathcal{S} \rangle$ is bounded by the $L_{\mathcal{C}}$-labels of the subtree starting at $v$ and the $L_{\mathcal{C}}$-labels on the path from the root to $v$. However, since no processor has been applied to the leaves of the proof tree (yet), for leaves we use the actual expected complexity instead of the $L_{\mathcal{C}}$-label. Moreover, for well-formed proof trees we require that the ADPs from $\mathcal{A} \setminus \mathcal{S}$ have already been taken into account in the path from the root to $v$. This will be exploited, e.g., in the *knowledge propagation processor* of Section 8.2.3.

**Definition 8.1.15** – Well-Formed Proof Tree.
*Let $\sqcup$ be the maximum operator on complexities, i.e., for $c, d \in \mathfrak{C}$, let $c \sqcup d = d$ if $c \sqsubseteq d$ and $c \sqcup d = c$ otherwise (so, e.g., $\mathbf{Pol}_2 \sqcup \mathbf{Pol}_1 = \mathbf{Pol}_2$).*
*A proof tree $(N, E, L_{\mathcal{A}}, L_{\mathcal{C}})$ is well formed if for every node $v$ with $L_{\mathcal{A}}(v) = \langle \mathcal{A}, \mathcal{S} \rangle$ and path $v_1, \ldots, v_k = v$ from the root $v_1$ to $v$, we have*

$$\iota_{\langle \mathcal{A}, \mathcal{S} \rangle} \sqsubseteq L_{\mathcal{C}}(v_1) \sqcup \cdots \sqcup L_{\mathcal{C}}(v_{k-1}) \sqcup \max\{L'_{\mathcal{C}}(w) \mid (v, w) \in E^*\} \text{ and}$$
$$\iota_{\langle \mathcal{A}, \mathcal{A} \setminus \mathcal{S} \rangle} \sqsubseteq L_{\mathcal{C}}(v_1) \sqcup \cdots \sqcup L_{\mathcal{C}}(v_{k-1}).$$

*Here, $E^*$ is the reflexive-transitive closure of the edge relation, and for $k = 1$, the empty maximum $L_{\mathcal{C}}(v_1) \sqcup \cdots \sqcup L_{\mathcal{C}}(v_{k-1})$ is $\mathbf{Pol}_0$. Moreover, let $L'_{\mathcal{C}}(v) = L_{\mathcal{C}}(v)$ for inner nodes $v$ and $L'_{\mathcal{C}}(v) = \iota_{L_{\mathcal{A}}(v)}$ for leaves $v$.*

The following corollary shows that for well-formed proof trees, the complexity of the ADP problem at the root is indeed bounded by the maximum of all complexities at the nodes.

**Corollary 8.1.16** – Complexity Bound from Well-Formed Proof Tree.
*Let $\mathfrak{P} = (N, E, L_{\mathcal{A}}, L_{\mathcal{C}})$ be a well-formed proof tree with $L_{\mathcal{A}}(v_1) = \langle \mathcal{A}, \mathcal{S} \rangle$ for the root $v_1$ of $\mathfrak{P}$. Then,*

$$\iota_{\langle \mathcal{A}, \mathcal{S} \rangle} \sqsubseteq \max\{L_{\mathcal{C}}(v) \mid v \in N\}.$$

*Proof.* We apply the first condition for well-formed proof trees (Definition 8.1.15) to the root $v_1$ itself. As the prefix $L_{\mathcal{C}}(v_1) \sqcup \cdots \sqcup L_{\mathcal{C}}(v_{k-1})$ is empty, we get $\iota_{\langle \mathcal{A}, \mathcal{S} \rangle} \sqsubseteq \max\{L'_{\mathcal{C}}(w) \mid w \in N\}$.
It remains to show $\max\{L'_{\mathcal{C}}(w) \mid w \in N\} \sqsubseteq \max\{L_{\mathcal{C}}(w) \mid w \in N\}$, for which it suffices to prove $L'_{\mathcal{C}}(w) \sqsubseteq L_{\mathcal{C}}(w)$ for all $w \in N$. For every inner node $w$, we have $L'_{\mathcal{C}}(w) = L_{\mathcal{C}}(w)$ by definition of $L'_{\mathcal{C}}$, and thus, $L'_{\mathcal{C}}(w) \sqsubseteq L_{\mathcal{C}}(w)$. For every leaf node $w$, we have $L'_{\mathcal{C}}(w) = \iota_{L_{\mathcal{A}}(w)}$. If $L_{\mathcal{A}}(w) = \langle \mathcal{A}_w, \mathcal{S}_w \rangle$ is solved, then $\iota_{L_{\mathcal{A}}(w)} = \mathbf{Pol}_0$. As $L_{\mathcal{C}}(w) = \mathbf{Pol}_0$ for solved leaves by Definition 8.1.14, we get $L'_{\mathcal{C}}(w) = \iota_{L_{\mathcal{A}}(w)} = \mathbf{Pol}_0 = L_{\mathcal{C}}(w)$. Otherwise, $L_{\mathcal{A}}(w)$ is not solved, so that $L_{\mathcal{C}}(w) = \mathbf{Inf}$ by Definition 8.1.14, and hence $L'_{\mathcal{C}}(w) = \iota_{L_{\mathcal{A}}(w)} \sqsubseteq \mathbf{Inf} = L_{\mathcal{C}}(w)$. In all cases we have $L'_{\mathcal{C}}(w) \sqsubseteq L_{\mathcal{C}}(w)$, which concludes the proof. *QED*

Now we can define when a processor is *sound*.

**Definition 8.1.17** – Soundness of Processors.
*A processor $\mathrm{Proc}(\langle \mathcal{A}, \mathcal{S} \rangle) = (c, \{\langle \mathcal{A}_1, \mathcal{S}_1 \rangle, \ldots, \langle \mathcal{A}_n, \mathcal{S}_n \rangle\})$ is sound if for all well-formed proof trees $(N, E, L_{\mathcal{A}}, L_{\mathcal{C}})$ and all nodes $v \in N$, we have: if $L_{\mathcal{A}}(v) = \langle \mathcal{A}, \mathcal{S} \rangle$ and $v_1, \ldots, v_k = v$ is the path from the root node $v_1$ to $v$, then*

$$\iota_{\langle \mathcal{A}, \mathcal{S} \rangle} \sqsubseteq L_{\mathcal{C}}(v_1) \sqcup \ldots \sqcup L_{\mathcal{C}}(v_{k-1}) \sqcup c \sqcup \iota_{\langle \mathcal{A}_1, \mathcal{S}_1 \rangle} \sqcup \ldots \sqcup \iota_{\langle \mathcal{A}_n, \mathcal{S}_n \rangle}, \quad (8.11)$$
$$\iota_{\langle \mathcal{A}_i, \mathcal{A}_i \setminus \mathcal{S}_i \rangle} \sqsubseteq L_{\mathcal{C}}(v_1) \sqcup \ldots \sqcup L_{\mathcal{C}}(v_{k-1}) \sqcup c \qquad \text{for all } 1 \leq i \leq n. \quad (8.12)$$

So (8.11) requires that the complexity of the considered ADPs $\mathcal{S}$ must be bounded by the maximum of all "previous" complexities $L_{\mathcal{C}}(v_1), \ldots, L_{\mathcal{C}}(v_{k-1})$, the newly derived complexity $c$, and the complexity of the remaining ADP problems $\langle \mathcal{A}_1, \mathcal{S}_1 \rangle, \ldots, \langle \mathcal{A}_n, \mathcal{S}_n \rangle$. Moreover, (8.12) ensures that in the remaining ADP problems $\langle \mathcal{A}_i, \mathcal{S}_i \rangle$, the complexity of the "non-considered" ADPs $\mathcal{A}_i \setminus \mathcal{S}_i$ is bounded by the maximum of all previous complexities and the newly derived complexity $c$. This ensures that well-formedness of

proof trees is preserved when extending them by applying sound processors.

Lemma 8.1.18 – Sound Processors Preserve Well-Formedness.
*Let $\mathfrak{P} = (N, E, L_{\mathcal{A}}, L_{\mathcal{C}})$ be a well-formed proof tree with a leaf $v$ where $L_{\mathcal{A}}(v)$ is not solved, and let* Proc *be a sound processor such that* $\text{Proc}(L_{\mathcal{A}}(v)) = (c, \{\langle \mathcal{A}_1, \mathcal{S}_1\rangle, \dots, \langle \mathcal{A}_n, \mathcal{S}_n\rangle\})$ *with $n \geq 1$. Let $\mathfrak{P}'$ result from $\mathfrak{P}$ by adding fresh nodes $w_1, \dots, w_n$ and edges $(v, w_1), \dots, (v, w_n)$, where the labeling is extended such that $L_{\mathcal{A}}(w_i) = \langle \mathcal{A}_i, \mathcal{S}_i\rangle$ for all $1 \leq i \leq n$, $L_{\mathcal{C}}(w_i) = \mathbf{Pol}_0$ if $\mathcal{S}_i = \varnothing$ and $L_{\mathcal{C}}(w_i) = \mathbf{Inf}$ otherwise, and the label of $v$ is changed to $L_{\mathcal{C}}(v) = c$. Then $\mathfrak{P}'$ is also a well-formed proof tree.*

*Proof.* $\mathfrak{P}'$ is a proof tree, since the edges from the new inner node $v$ represent the application of Proc and the new leaves $w_1, \dots, w_n$ are labeled as required by Definition 8.1.14. For the first condition required for well-formed proof trees, note that in $\mathfrak{P}$, we had $L'_{\mathcal{C}}(v) = \iota_{L_{\mathcal{A}}(v)}$, because $v$ was a leaf. Since $v$ is not a leaf anymore in $\mathfrak{P}'$, here we have $L'_{\mathcal{C}}(v) = L_{\mathcal{C}}(v) = c$. However, $v$ now has the children $w_1, \dots, w_n$. Let $v_1, \dots, v_k = v$ be the path from the root to $v$. Hence, we obtain

$$\begin{aligned} & L_{\mathcal{C}}(v_1) \sqcup \dots \sqcup L_{\mathcal{C}}(v_{k-1}) \sqcup \iota_{L_{\mathcal{A}}(v)} \\ \sqsubseteq\ & L_{\mathcal{C}}(v_1) \sqcup \dots \sqcup L_{\mathcal{C}}(v_{k-1}) \sqcup c \sqcup \iota_{\langle \mathcal{A}_1, \mathcal{S}_1\rangle} \sqcup \dots \sqcup \iota_{\langle \mathcal{A}_n, \mathcal{S}_n\rangle} \qquad (\dagger) \\ =\ & L_{\mathcal{C}}(v_1) \sqcup \dots \sqcup L_{\mathcal{C}}(v_{k-1}) \sqcup L_{\mathcal{C}}(v_k) \sqcup L'_{\mathcal{C}}(w_1) \sqcup \dots \sqcup L'_{\mathcal{C}}(w_n). \end{aligned}$$

Here, (†) holds due to the first condition (8.11) of Definition 8.1.17. Thus, $\mathfrak{P}'$ also satisfies the first condition for well-formed proof trees.
Now we consider the second condition required for well-formed proof trees. Since $\mathfrak{P}$ is well formed and Proc is sound, the second condition (8.12) of Definition 8.1.17 implies that $\iota_{\langle \mathcal{A}_i, \mathcal{A}_i \setminus \mathcal{S}_i\rangle} \sqsubseteq L_{\mathcal{C}}(v_1) \sqcup \dots \sqcup L_{\mathcal{C}}(v_{k-1}) \sqcup L_{\mathcal{C}}(v_k)$ with $L_{\mathcal{C}}(v_k) = c$ holds for all $1 \leq i \leq n$, which is the second condition for the new leaves $w_i$. For all other nodes, the path from the root has not changed. Hence, $\mathfrak{P}'$ is well formed as well. *QED*

To determine an upper bound on the expected runtime complexity $\iota_{\mathcal{P}}$ of a finite PTRS $\mathcal{P}$, our ADP framework starts with the canonical ADP problem $\langle \mathcal{DP}(\mathcal{P}), \mathcal{DP}(\mathcal{P})\rangle$ and applies sound processors repeatedly until all problems are solved. The initial proof tree that only consists of its root $v_1$ with $L_{\mathcal{A}}(v_1) = \langle \mathcal{DP}(\mathcal{P}), \mathcal{DP}(\mathcal{P})\rangle$ is well formed: the first condition of Definition 8.1.15 holds since $v_1$ is a leaf, and the second one holds since $\iota_{\langle \mathcal{DP}(\mathcal{P}), \varnothing\rangle} = \mathbf{Pol}_0$. By Lemma 8.1.18, all proof trees that we construct in this way are well formed. Then by Theorem 8.1.12 and Corollary 8.1.16, the expected runtime complexity $\iota_{\mathcal{P}}$ is bounded by the maximum of all complexities occurring in the corresponding proof tree.

Corollary 8.1.19 – Soundness of the ADP Framework for Expected Runtime.
*Let $\mathcal{P}$ be a finite PTRS and $\mathfrak{P} = (N, E, L_{\mathcal{A}}, L_{\mathcal{C}})$ be a well-formed solved proof tree where $L_{\mathcal{A}}(v_1) = \langle \mathcal{DP}(\mathcal{P}), \mathcal{DP}(\mathcal{P})\rangle$ for the root $v_1$ of $\mathfrak{P}$. Then we have*

$$\iota_{\mathcal{P}} \sqsubseteq \max\{L_{\mathcal{C}}(v) \mid v \in N\}.$$

*Proof.* Since the root $v_1$ of the well-formed proof tree $\mathfrak{P}$ is labeled with the canonical ADP problem $L_{\mathcal{A}}(v_1) = \langle \mathcal{DP}(\mathcal{P}), \mathcal{DP}(\mathcal{P})\rangle$, Corollary 8.1.16 yields

$$\iota_{\langle \mathcal{DP}(\mathcal{P}), \mathcal{DP}(\mathcal{P})\rangle} \sqsubseteq \max\{L_{\mathcal{C}}(v) \mid v \in N\}.$$

Moreover, by the chain criterion (Theorem 8.1.12), we have $\iota_{\mathcal{P}} = \iota_{\langle \mathcal{DP}(\mathcal{P}), \mathcal{DP}(\mathcal{P}) \rangle}$, and thus,

$$\iota_{\mathcal{P}} = \iota_{\langle \mathcal{DP}(\mathcal{P}), \mathcal{DP}(\mathcal{P}) \rangle} \sqsubseteq \max\{L_{\mathcal{C}}(v) \mid v \in N\}.$$

QED

Remark 8.1.20

While our framework is inspired by the DT framework of [NEG13] for complexity analysis of TRSs, our adaptation to PTRSs differs from [NEG13] in several aspects. Apart from using ADPs instead of dependency tuples (which results in a "tight" chain criterion instead of an over-approximation), we use proof trees (instead of just proof chains, which allows us to use processors that return several sub-problems), and we introduce the novel concept of *well-formed* proof trees and require that sound processors preserve well-formedness. This will allow us to define a *knowledge propagation processor* in Section 8.2.3 which takes the knowledge provided by well-formed proof trees into account. In contrast to [NEG13], we obtain such a processor without extending our ADP problems by an additional component $\mathcal{K}$ that contains those ADPs which were already taken into account in the proof up to now, since in our setting we would always have $\mathcal{K} = \mathcal{A} \setminus \mathcal{S}$.

SECTION 8.2

## Structural Processors

In this section, we introduce processors based on structural properties of the ADP problem. Compared to the structural processors for `AST`, we have one additional processor without counterpart in the ADP framework for `AST`, namely the *knowledge propagation processor*, which we adapt from [NEG13]. Moreover, the dependency graph processor directly integrates the idea of the usable terms processor, so there is no additional usable terms processor. In total, we present three processors, but several of the remaining processors shown in Chapter 7 can be adjusted to the complexity analysis setting as well.

1. The *usable rules processor* sets the flag to `f` for all rules that can never be used to evaluate the arguments of an annotated symbol when their variables are instantiated with normal forms, so that these rules no longer have to be counted.
2. The *dependency graph processor* builds a control flow graph of which function calls can follow each other and decomposes the problem along its SCC-prefixes, where each SCC is analyzed together with nodes that can reach it, since for complexity one may not consider SCCs in isolation.
3. The *knowledge propagation processor* removes an ADP $\alpha$ from the set $\mathcal{S}$ once all its predecessors in the dependency graph have already been accounted for, as the complexity of $\alpha$ is then already subsumed by the knowledge collected in the proof tree.

Another possible processor is a *probability removal processor* similar to the one for `AST` (Section 7.4.4): if all ADPs are non-probabilistic, then one can switch to the non-probabilistic *dependency tuple* framework for complexity analysis of [NEG13].

The soundness proofs of these processors use the following observation for chain trees that start with $t^\sharp$ for some $t \in \mathcal{T}_{\mathcal{B}}$. An annotation at a position $\tau$ is created by an **at**- or

**af**-step at a position above or equal to $\tau$ and stays at $\tau$ until it is used or removed, where steps strictly below $\tau$ that do not remove it use ADPs with the flag $\mathtt{t}$. So for every node $w \neq \mathfrak{r}$ that performs an **at**- or **af**-step at a position $\tau$, its *creator* is the last proper ancestor $v$ of $w$ that rewrites at a position above or equal to $\tau$. Then $v$ rewrites with an ADP $\ell \xhookrightarrow{m} \{p_1 r_1, \dots, p_k r_k\}$ and a substitution $\sigma$ at a position $\pi$ with $\tau = \pi.\rho$ for some $\rho \in \mathrm{pos}_{\mathcal{D}^\sharp}(r_j)$, where $w$ is below the child of $v$ that corresponds to $r_j$. Moreover, $\ell\sigma \in \mathtt{ANF}_{\mathcal{A}}$, and $\flat(t_w|_\tau)$ results from $\flat(r_j|_\rho)\sigma$ by innermost steps strictly below the root with ADPs with the flag $\mathtt{t}$. Since no node created by $v$ at a position is an ancestor of another one created by $v$ at the same position, the probabilities of all nodes whose creator is $v$ add up to at most $d \cdot p_v$ if every right-hand side contains at most $d$ annotated symbols.

### 8.2.1 Usable Rules Processor

We start with a processor that considers the *usable rules*. Usable rules over-approximate the set of those rules that can be used to evaluate the arguments of annotated function symbols if their variables are instantiated by normal forms (as required in innermost evaluations). Essentially, the usable rules of a term $t$ consist of all rules for the defined symbols $f$ occurring in $t$ and all rules that are usable for the terms in the right-hand sides of $f$-rules.

Definition 8.2.1 – Usable Rules.
*For every $f \in \Sigma^\sharp$ and set of ADPs $\mathcal{A}$, let* $\mathrm{Rules}_{\mathcal{A}}(f) = \{\ell \xhookrightarrow{\mathtt{t}} \mu \in \mathcal{A} \mid \mathrm{root}(\ell) = f\}$. *Moreover, for every $t \in \mathcal{T}^\sharp$, the usable rules $\mathcal{U}_{\mathcal{A}}(t)$ of $t$ w.r.t. $\mathcal{A}$ are defined as*

$$\mathcal{U}_{\mathcal{A}}(t) = \varnothing, \qquad \text{if } t \in \mathcal{V} \text{ or } \mathcal{A} = \varnothing$$

$$\mathcal{U}_{\mathcal{A}}(f(t_1, \dots, t_n)) = \mathrm{Rules}_{\mathcal{A}}(f) \cup \bigcup_{j=1}^{n} \mathcal{U}_{\mathcal{A}'}(t_j) \cup \bigcup_{\substack{\ell \xhookrightarrow{\mathtt{t}} \mu \in \mathrm{Rules}_{\mathcal{A}}(f),\\ r \in \mathrm{Supp}(\mu)}} \mathcal{U}_{\mathcal{A}'}(\flat(r)), \qquad \text{otherwise,}$$

*where $\mathcal{A}' = \mathcal{A} \setminus \mathrm{Rules}_{\mathcal{A}}(f)$. The usable rules of $\mathcal{A}$ are*

$$\mathcal{U}(\mathcal{A}) = \bigcup_{\substack{\ell \xhookrightarrow{m} \mu \in \mathcal{A},\\ r \in \mathrm{Supp}(\mu), t \trianglelefteq^\sharp r}} \mathcal{U}_{\mathcal{A}}(t^\sharp).$$

Similar to the usable rules processor for $\mathtt{AST}$ in Section 7.4.3, our usable rules processor sets the flag of all non-usable rules in $\mathcal{A}$ to $\mathtt{f}$ to indicate that they cannot be used to evaluate arguments of annotated functions that are rewritten afterwards. The rules in $\mathcal{A}$'s subset $\mathcal{S}$ are changed analogously (since the purpose of $\mathcal{S}$ is only to indicate which ADPs must still be counted for complexity).

Theorem 8.2.2 – Usable Rules Processor.
For an ADP problem $\langle \mathcal{A}, \mathcal{S} \rangle$, let

$$\mathcal{A}' = \mathcal{U}(\mathcal{A}) \cup \{\ell \xhookrightarrow{\mathtt{f}} \mu \mid \ell \xhookrightarrow{m} \mu \in \mathcal{A} \setminus \mathcal{U}(\mathcal{A})\},$$
$$\mathcal{S}' = (\mathcal{S} \cap \mathcal{U}(\mathcal{A})) \cup \{\ell \xhookrightarrow{\mathtt{f}} \mu \mid \ell \xhookrightarrow{m} \mu \in \mathcal{S} \setminus \mathcal{U}(\mathcal{A})\}.$$

Then, $\mathrm{Proc}_{\mathtt{UR}}(\langle \mathcal{A}, \mathcal{S} \rangle) = (\mathbf{Pol}_0, \{\langle \mathcal{A}', \mathcal{S}' \rangle\})$ is sound.

*Proof.* Let $\mathfrak{P}$ be a well-formed proof tree with $L_{\mathcal{A}}(v) = \langle \mathcal{A}, \mathcal{S} \rangle$ and let $v_1, \ldots, v_k = v$ be the path from the root node $v_1$ to $v$. For every $\alpha \in \mathcal{A}$, let $\alpha' = \alpha$ if $\alpha \in \mathcal{U}(\mathcal{A})$, and otherwise let $\alpha'$ result from $\alpha$ by changing its flag to $\mathsf{f}$. So $\mathcal{A}' = \{\alpha' \mid \alpha \in \mathcal{A}\}$, $\mathcal{S}' = \{\alpha' \mid \alpha \in \mathcal{S}\}$, and $\mathtt{ANF}_{\mathcal{A}} = \mathtt{ANF}_{\mathcal{A}'}$, since $\alpha$ and $\alpha'$ have the same left-hand side.
For (8.11), it suffices to show $\iota_{\langle \mathcal{A}, \mathcal{S} \rangle} \sqsubseteq \iota_{\langle \mathcal{A}', \mathcal{S}' \rangle}$. Let $\mathfrak{T}$ be an $\mathcal{A}$-CT whose root is labeled with $t^\sharp$ for some $t \in \mathcal{T}_{\mathcal{B}}$, and let $\mathfrak{T}'$ be the $\mathcal{A}'$-CT that performs every step of $\mathfrak{T}$ with $\alpha'$ instead of $\alpha$ (at the same position and with the same substitution). Let $w$ be a node that performs an **at**- or **af**-step at a position $\tau$ in $\mathfrak{T}$, and let $v$ be its creator. As observed at the beginning of this section, the steps between $v$ and $w$ strictly below $\tau$ are innermost steps with ADPs with the flag $\mathsf{t}$ in a reduction of $\flat(r_j|_\rho)\sigma$, where $\sigma$ instantiates all variables by normal forms. As in the proof of Theorem 7.4.14 (see also [GTSF06]), such steps only use ADPs from $\mathcal{U}(\mathcal{A})$, which keep their flag $\mathsf{t}$ in $\mathcal{A}'$. So by induction on the depth of $w$, the annotation at $\tau$ also exists in $\mathfrak{T}'$, i.e., every **at**- or **af**-step of $\mathfrak{T}$ is also one of $\mathfrak{T}'$. As $\alpha \in \mathcal{S}$ implies $\alpha' \in \mathcal{S}'$, we obtain $\mathrm{edl}_{\langle \mathcal{A}, \mathcal{S} \rangle}(\mathfrak{T}) \leq \mathrm{edl}_{\langle \mathcal{A}', \mathcal{S}' \rangle}(\mathfrak{T}')$.
For (8.12), we show $\iota_{\langle \mathcal{A}', \mathcal{A}' \setminus \mathcal{S}' \rangle} \sqsubseteq \iota_{\langle \mathcal{A}, \mathcal{A} \setminus \mathcal{S} \rangle}$, which implies (8.12) by the well-formedness of $\mathfrak{P}$. Every $\mathcal{A}'$-CT $\mathfrak{T}'$ can be viewed as an $\mathcal{A}$-CT $\mathfrak{T}$ by using some $\alpha \in \mathcal{A}$ with $\alpha' = \gamma$ instead of every ADP $\gamma$ used in $\mathfrak{T}'$. Since $\alpha$ only has the flag $\mathsf{f}$ if $\gamma$ has it, $\mathfrak{T}$ keeps all annotations of $\mathfrak{T}'$, and $\gamma \notin \mathcal{S}'$ implies $\alpha \notin \mathcal{S}$. Hence, $\mathrm{edl}_{\langle \mathcal{A}', \mathcal{A}' \setminus \mathcal{S}' \rangle}(\mathfrak{T}') \leq \mathrm{edl}_{\langle \mathcal{A}, \mathcal{A} \setminus \mathcal{S} \rangle}(\mathfrak{T})$. *QED*

Example 8.2.3 (Usable Rules Processor). Consider the ADP problem from Example 8.1.1, namely $\langle \mathcal{DP}(\mathcal{P}_{\mathsf{quot}}), \mathcal{DP}(\mathcal{P}_{\mathsf{quot}}) \rangle$. There is only one term $r$ in the right-hand sides with an annotated subterm $t \trianglelefteq^\sharp r$ where $t^\sharp$ has a defined symbol below the annotated root, viz. $t^\sharp = \mathsf{q}^\sharp(\mathsf{geo}(x), y, y)$. Thus, only the $\mathsf{geo}$-ADP (8.2) is usable and we can set the flag of all other ADPs to $\mathsf{f}$. Hence, we get $\mathrm{Proc}_{\mathtt{UR}}(\langle \mathcal{DP}(\mathcal{P}_{\mathsf{quot}}), \mathcal{DP}(\mathcal{P}_{\mathsf{quot}}) \rangle) = (\mathbf{Pol}_0, \{\langle \mathcal{A}_1, \mathcal{A}_1 \rangle\})$ where $\mathcal{A}_1 = \{(8.13)\text{–}(8.17)\}$ with

$$\mathsf{start}(x, y) \overset{\mathsf{f}}{\hookrightarrow} \{{}^1\mathsf{q}^\sharp(\mathsf{geo}^\sharp(x), y, y)\} \quad (8.13)$$

$$\mathsf{geo}(x) \overset{\mathsf{t}}{\hookrightarrow} \{{}^{1/2}\mathsf{geo}^\sharp(\mathsf{s}(x)),\ {}^{1/2}x\} \quad (8.14)$$

$$\mathsf{q}(\mathsf{s}(x), \mathsf{s}(y), z) \overset{\mathsf{f}}{\hookrightarrow} \{{}^1\mathsf{q}^\sharp(x, y, z)\} \quad (8.15)$$

$$\mathsf{q}(x, 0, \mathsf{s}(z)) \overset{\mathsf{f}}{\hookrightarrow} \{{}^1\mathsf{s}(\mathsf{q}^\sharp(x, \mathsf{s}(z), \mathsf{s}(z)))\} \quad (8.16)$$

$$\mathsf{q}(0, \mathsf{s}(y), \mathsf{s}(z)) \overset{\mathsf{f}}{\hookrightarrow} \{{}^1 0\}. \quad (8.17)$$

Having fewer rules with the flag $\mathsf{t}$ has advantages, e.g., for the *dependency graph* and the *reduction pair processor* in Sections 8.2.2 and 8.3.1.

Counterexample 8.2.4 (Basic Start Terms). The restriction to basic start terms is not only required to infer polynomial upper bounds from CPIs (see Corollary 3.4.36), but it is also essential for the soundness of the usable rules processor. To see this, let $\mathcal{P}'_{\mathsf{exp}}$ contain all rules of $\mathcal{P}_{\mathsf{exp}}$ from Example 8.0.2 except $\mathsf{start} \to \{{}^1\mathsf{f}(\mathsf{geo}(0))\}$. Note that $\iota_{\mathcal{P}'_{\mathsf{exp}}} = \mathbf{Exp}$, since a reduction starting in $\mathsf{f}(\mathsf{s}^k(0))$ takes $2^{k+1} - 2$ steps and the $\mathsf{geo}$-rule only needs 2 steps in expectation. If we do not require basic start terms, then we can start an evaluation with the term $\mathsf{f}(\mathsf{geo}(0))$, i.e., then $\mathcal{P}'_{\mathsf{exp}}$ is not $\mathtt{SAST}$ (not even for innermost evaluation). The canonical ADPs $\mathcal{DP}(\mathcal{P}'_{\mathsf{exp}})$ are the same as for $\mathcal{P}_{\mathsf{exp}}$ in Example 8.1.2 just without the $\mathsf{start}$-ADP. Thus, $\mathcal{DP}(\mathcal{P}'_{\mathsf{exp}})$ has no usable rules and $\mathrm{Proc}_{\mathtt{UR}}$ sets the flag of all ADPs to $\mathsf{f}$. When starting with a term like $\mathsf{f}^\sharp(\mathsf{geo}(0))$, then the first application of the $\mathsf{geo}$-ADP now removes the annotation of the $\mathsf{f}^\sharp$-symbol above it, as the $\mathsf{geo}$-ADP now has the flag $\mathsf{f}$. So for the resulting ADP problem, the $\mathsf{f}$-ADPs are never applied at annotated positions in such chain trees and thus, they all have finite expected derivation length. Hence, we would now falsely infer that $\mathcal{P}'_{\mathsf{exp}}$ is $\mathtt{SAST}$ w.r.t. arbitrary start terms.

### 8.2.2 Dependency Graph Processor

The *dependency graph* is a control flow graph that indicates which function calls can occur after each other. This does not depend on the probabilities, and we can consider each function call on its own. Hence, we can use the ordinary dependency graph of the corresponding (non-probabilistic) dependency pairs. To also detect the predecessors of ADPs $\ell \xhookrightarrow{m} \mu$ without annotations, we add a dependency pair $\ell^\sharp \to \bot$ for a fresh symbol $\bot$ in that case. By using ordinary dependency pairs for the control flow graph, we integrate the idea of the usable terms processor of Chapter 7 (Section 7.4.2) directly into the dependency graph processor and do not need to define an additional usable terms processor for analyzing SAST.

Definition 8.2.5 – Non-Probabilistic Variant, Dependency Pairs.

*For a set of ADPs $\mathcal{A}$, let* $\mathrm{np}(\mathcal{A}) = \{\ell \to \flat(r_j) \mid \ell \xhookrightarrow{\mathtt{t}} \{p_1 r_1, \dots, p_k r_k\} \in \mathcal{A}, 1 \le j \le k\}$ *denote its non-probabilistic rule variant. So* $\mathrm{np}(\mathcal{A})$ *is an ordinary TRS which only considers the ADPs with the flag* t*. For any ADP* $\alpha = \ell \xhookrightarrow{m} \{p_1 r_1, \dots, p_k r_k\}$*, let* $\mathrm{dp}(\alpha) = \{\ell^\sharp \to t^\sharp \mid 1 \le j \le k, t \trianglelefteq_\sharp r_j\}$*. Moreover, let* $\mathrm{dp}^\bot(\alpha) = \{\ell^\sharp \to \bot\}$ *if* $\mathrm{dp}(\alpha) = \varnothing$*, and* $\mathrm{dp}^\bot(\alpha) = \mathrm{dp}(\alpha)$*, otherwise. For $\mathcal{A}$, let* $\mathrm{dp}(\mathcal{A}) = \bigcup_{\alpha \in \mathcal{A}} \mathrm{dp}^\bot(\alpha)$ *denote its non-probabilistic DP variant, which is a set containing dependency pairs as in [AG00, GTSF06] and rules of the form* $\ell^\sharp \to \bot$.

So each dependency pair from $\mathrm{dp}(\alpha)$ corresponds to a single annotation on the right-hand side of the ADP $\alpha$. In the dependency graph, the edges indicate whether one DP can follow another when the instantiated arguments are evaluated with $\mathrm{np}(\mathcal{A})$.

Definition 8.2.6 – Dependency Graph.

*The $\mathcal{A}$-dependency graph has the nodes* $\mathrm{dp}(\mathcal{A})$ *and there is an edge from a DP* $\ell_1^\sharp \to t_1^\sharp$ *to a node with the left-hand side* $\ell_2^\sharp$ *(i.e., to a DP* $\ell_2^\sharp \to t_2^\sharp$ *or to* $\ell_2^\sharp \to \bot$*) if there are substitutions* $\sigma_1, \sigma_2$ *such that* $t_1^\sharp\sigma_1 \xrightarrow{\mathtt{i}}{}^*_{\mathrm{np}(\mathcal{A})} \ell_2^\sharp\sigma_2$ *and both* $\ell_1^\sharp\sigma_1$ *and* $\ell_2^\sharp\sigma_2$ *are in argument normal form, i.e.,* $\ell_1^\sharp\sigma_1, \ell_2^\sharp\sigma_2 \in \mathtt{ANF}_{\mathcal{A}}$.

Like the innermost dependency graph of Definition 7.4.5, this graph only considers innermost reductions, but its nodes are dependency pairs instead of ADPs. While the dependency graph is not computable in general, several techniques have been developed to compute over-approximations of the graph automatically, e.g., [AG00, GTSF06, HM05], including the estimation via Cap and Ren from Chapter 7.

Example 8.2.7 (Dependency Graph). We continue with $\langle \mathcal{A}_1, \mathcal{A}_1 \rangle$ from Example 8.2.3, where $\mathcal{A}_1 = \{(8.13)\text{–}(8.17)\}$. The nodes $\mathrm{dp}(\mathcal{A}_1)$ contain the DPs

$$\mathsf{start}^\sharp(x,y) \to \mathsf{q}^\sharp(\mathsf{geo}(x),y,y) \quad (8.18) \qquad \mathsf{q}^\sharp(\mathsf{s}(x),\mathsf{s}(y),z) \to \mathsf{q}^\sharp(x,y,z) \quad (8.21)$$

$$\mathsf{start}^\sharp(x,y) \to \mathsf{geo}^\sharp(x) \quad (8.19) \qquad \mathsf{q}^\sharp(x,\mathsf{0},\mathsf{s}(z)) \to \mathsf{q}^\sharp(x,\mathsf{s}(z),\mathsf{s}(z)) \quad (8.22)$$

$$\mathsf{geo}^\sharp(x) \to \mathsf{geo}^\sharp(\mathsf{s}(x)) \quad (8.20) \qquad \mathsf{q}^\sharp(\mathsf{0},\mathsf{s}(y),\mathsf{s}(z)) \to \bot. \quad (8.23)$$

The $\mathcal{A}_1$-dependency graph is depicted in Figure 8.5.

The idea of the *dependency graph processor* for termination analysis is to analyze each strongly connected component (SCC)[1] of the dependency graph separately. However

[1] Remember, a set $\mathcal{G}$ of DPs is an *SCC* if it is a maximal cycle, i.e., a maximal set where for any $\alpha, \alpha'$ in $\mathcal{G}$ there is a non-empty path from $\alpha$ to $\alpha'$ only traversing nodes from $\mathcal{G}$.

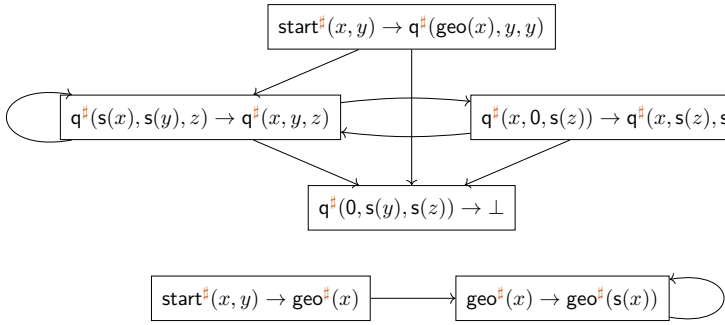



Figure 8.5: $\mathcal{A}_1$-dependency graph.

(already in the non-probabilistic setting, e.g., [NEG13]), this is not possible when analyzing complexity. There are examples where all SCCs have linear complexity but the full system has quadratic complexity, or where all individual SCCs have a finite complexity but the full system is not SAST (see Counterexample 8.2.11).

The problem is that when considering an SCC individually, then we lose the information about how often and with which instantiations of the variables this SCC is "called". For that reason, we now present a dependency graph processor (adapted from [NEG13]) which regards each SCC *together with its "prefix"*, i.e., together with all nodes of the dependency graph that can reach the SCC. As usual, we say that a node *reaches* an SCC if there is a path from the node to the SCC in the dependency graph, where the path has length $\geq 0$, i.e., each node of the SCC also reaches it. Similarly, a node $\alpha$ reaches a node $\beta$ if there is a path of length $\geq 0$ from $\alpha$ to $\beta$, i.e., every node reaches itself. However, prefixes which are independent of each other can be regarded separately, i.e., we only regard SCC-prefixes $\mathcal{J}$ where for all nodes $\alpha, \beta \in \mathcal{J}$, $\alpha$ reaches $\beta$ or $\beta$ reaches $\alpha$.

Definition 8.2.8 – SCC-Prefix.
*Let $\mathcal{A}$ be a set of ADPs. Then $\mathcal{J}$ is an SCC-prefix of the $\mathcal{A}$-dependency graph if there exists an SCC $\mathcal{G} \subseteq \mathcal{J}$ where $\mathcal{J} \subseteq \mathrm{dp}(\mathcal{A})$ is a maximal set such that all DPs of $\mathcal{J}$ reach $\mathcal{G}$ and for all $\alpha, \beta \in \mathcal{J}$, $\alpha$ reaches $\beta$ or $\beta$ reaches $\alpha$. Here, a node reaches another node (or a set of nodes) if there is a path of length $\geq 0$ from it to this node (or to a node of the set). The set of all SCC-prefixes is denoted by $\boldsymbol{\mathcal{J}}^{\mathcal{A}}_{\mathrm{pref}}$.*

For example, the $\mathcal{A}_1$-dependency graph of Figure 8.5 has two SCCs $\{(8.20)\}$ and $\{(8.21), (8.22)\}$, and two SCC-prefixes $\boldsymbol{\mathcal{J}}^{\mathcal{A}_1}_{\mathrm{pref}} = \{\mathcal{J}_1, \mathcal{J}_2\}$, namely $\mathcal{J}_1 = \{(8.19), (8.20)\}$ and $\mathcal{J}_2 = \{(8.18), (8.21), (8.22)\}$. In this example, $\mathcal{J}_1$ and $\mathcal{J}_2$ represent two completely independent parts of the dependency graph.

The dependency graph processor now handles each SCC-prefix $\mathcal{J}$ separately. To consider only the effects of the DPs $\mathcal{J}$ in the ADPs of $\mathcal{A}$, we replace every ADP $\alpha$ by the variant $\alpha|_{\mathcal{J}}$ where only those symbols are annotated that correspond to the DPs from $\mathcal{J}$. For example, for $\mathcal{J}_1 = \{(8.19), (8.20)\}$ and the ADP

$$\alpha = \quad \mathsf{start}(x, y) \xhookrightarrow{\mathsf{f}} \{{}^{1}\mathsf{q}^{\sharp}(\mathsf{geo}^{\sharp}(x), y, y)\}, \tag{8.13}$$

in $\alpha|_{\mathcal{J}_1}$ we only annotate $\mathsf{geo}^{\sharp}$ but not $\mathsf{q}^{\sharp}$ in the right-hand side, due to the DP (8.19) which results from its subterm $\mathsf{geo}^{\sharp}(x)$, i.e., $\alpha|_{\mathcal{J}_1} = \mathsf{start}(x, y) \xhookrightarrow{\mathsf{f}} \{{}^{1}\mathsf{q}(\mathsf{geo}^{\sharp}(x), y, y)\}$.

**Theorem 8.2.9** – Dependency Graph Processor.

Let $\langle \mathcal{A}, \mathcal{S} \rangle$ be an ADP problem and let $\mathcal{J} \in \mathcal{J}^{\mathcal{A}}_{\text{pref}}$ be an SCC-prefix of the $\mathcal{A}$-dependency graph. For any ADP $\alpha = \ell \xhookrightarrow{m} \{{}^{p_1}r_1, \dots, {}^{p_k}r_k\} \in \mathcal{A}$ let $\alpha|_{\mathcal{J}} = \ell \xhookrightarrow{m} \{{}^{p_1}\sharp_{\Phi_1}(r_1), \dots, {}^{p_k}\sharp_{\Phi_k}(r_k)\}$ where for $1 \leq j \leq k$, we have $\pi \in \Phi_j$ iff there exists an $\ell^\sharp \to t^\sharp \in \mathcal{J}$ such that $t \trianglelefteq^{\pi}_{\sharp} r_j$ (i.e., $\pi \in \text{pos}_{\mathcal{D}^\sharp}(r_j)$ and $t = \flat(r_j|_\pi)$). Similarly, let $\mathcal{A}|_{\mathcal{J}} = \{\alpha|_{\mathcal{J}} \mid \alpha \in \mathcal{A}\}$ and $\mathcal{S}|_{\mathcal{J}} = \{\alpha|_{\mathcal{J}} \mid \alpha \in \mathcal{S}\}$.
Then $\text{Proc}_{\texttt{DG}}(\langle \mathcal{A}, \mathcal{S} \rangle) = (\mathbf{Pol}_0, \{\langle \mathcal{A}|_{\mathcal{J}}, \mathcal{S}|_{\mathcal{J}} \rangle \mid \mathcal{J} \in \mathcal{J}^{\mathcal{A}}_{\text{pref}}\})$ is sound if $\mathcal{J}^{\mathcal{A}}_{\text{pref}} \neq \varnothing$, and $\text{Proc}_{\texttt{DG}}(\langle \mathcal{A}, \mathcal{S} \rangle) = (\mathbf{Pol}_0, \{\langle \mathcal{A}, \varnothing \rangle\})$ is sound if $\mathcal{J}^{\mathcal{A}}_{\text{pref}} = \varnothing$.

Note that if a DP from $\mathcal{J}$ could originate from several subterms $r_j|_\pi$ in several ADPs $\alpha$, then we annotate all their positions in $\mathcal{A}|_{\mathcal{J}}$. A slightly more powerful variant of the processor could be obtained by storing for every DP of $\mathcal{J}$ from which subterm of which ADP it originates, and only annotating this position. We did not do this in Theorem 8.2.9 to ease the presentation.

*Proof.* Let $\mathfrak{P}$ be a well-formed proof tree with $L_{\mathcal{A}}(v) = \langle \mathcal{A}, \mathcal{S} \rangle$, let $v_1, \dots, v_k = v$ be the path from the root node $v_1$ to $v$, and let $c_{(v_1,\dots,v_k)} = L_{\mathcal{C}}(v_1) \sqcup \dots \sqcup L_{\mathcal{C}}(v_{k-1})$. Let $\mathcal{J}^{\mathcal{A}}_{\text{pref}} = \{\mathcal{J}_1, \dots, \mathcal{J}_n\}$ with $n \geq 0$, let $N = |\text{dp}(\mathcal{A})|$, let $d$ be the maximal number of annotated symbols in any term on a right-hand side of an ADP from $\mathcal{A}$, and let $D = \max(d, 1)$. *Proof*

▶ *Proof*((8.11)): Let $\mathfrak{T}$ be an $\mathcal{A}$-CT whose root is labeled with $t^\sharp$ for some $t \in \mathcal{T}_{\mathcal{B}}$, and for $1 \leq i \leq n$, let $\mathfrak{T}_i$ be the $\mathcal{A}|_{\mathcal{J}_i}$-CT that performs every step of $\mathfrak{T}$ with $\alpha|_{\mathcal{J}_i}$ instead of $\alpha$ (at the same position and with the same substitution). This is possible, since $\alpha$ and $\alpha|_{\mathcal{J}_i}$ only differ in the annotations of their right-hand sides, and by induction, every node has the same term in $\mathfrak{T}$ and $\mathfrak{T}_i$ up to annotations, where every annotated position in $\mathfrak{T}_i$ is also annotated in $\mathfrak{T}$. Let Junk be the set of all nodes that perform an **at**- or **af**-step in $\mathfrak{T}$, but in none of the trees $\mathfrak{T}_1, \dots, \mathfrak{T}_n$. Then

$$\text{edl}_{\langle \mathcal{A}, \mathcal{S} \rangle}(\mathfrak{T}) \leq \sum_{1 \leq i \leq n} \text{edl}_{\langle \mathcal{A}|_{\mathcal{J}_i}, \mathcal{S}|_{\mathcal{J}_i} \rangle}(\mathfrak{T}_i) + \sum_{w \in \text{Junk}} p_w. \tag{8.24}$$

To bound the nodes in Junk, consider a node $w$ that performs an **at**- or **af**-step in $\mathfrak{T}$. Repeatedly taking the creator yields a *creation chain* $\mathfrak{r} = w_0, w_1, \dots, w_h = w$, where $w_{g-1}$ is the creator of $w_g$ for all $1 \leq g \leq h$. If $w_{g-1}$ uses the ADP $\ell \xhookrightarrow{m} \{{}^{p_1}r_1, \dots, {}^{p_k}r_k\}$ and creates the annotation of $w_g$ at the position $\rho$ of $r_j$, let $\delta_g = \ell^\sharp \to (\flat(r_j|_\rho))^\sharp \in \text{dp}(\mathcal{A})$.

(a) There is an edge from $\delta_g$ to every node of $\text{dp}^{\perp}(\alpha')$, where $\alpha'$ is the ADP used at $w_g$ (in particular, to $\delta_{g+1}$): if $w_{g-1}$ and $w_g$ use the substitutions $\sigma$ and $\sigma'$ and $\alpha'$ has the left-hand side $\ell'$, then $(\flat(r_j|_\rho))^\sharp\sigma \xrightarrow{\text{i}}{}^*_{\text{np}(\mathcal{A})} \ell'^\sharp\sigma'$ by the observation at the beginning of this section, and $\ell^\sharp\sigma, \ell'^\sharp\sigma' \in \texttt{ANF}_{\mathcal{A}}$, since all steps are innermost.

(b) If $\delta_1, \dots, \delta_g \in \mathcal{J}_i$, then $w_g$ also performs an **at**- or **af**-step in $\mathfrak{T}_i$: by induction on $g$, $w_{g-1}$ creates the annotation of $w_g$ in $\mathfrak{T}_i$ as well, since $\delta_g \in \mathcal{J}_i$, and it is not removed afterwards, since the flags of the ADPs are the same in $\mathcal{A}|_{\mathcal{J}_i}$.

Now let $w \in \text{Junk}$, and let $g^*$ be the largest index such that $\delta_{g^*}$ reaches an SCC (or $g^* = 0$ if there is none). By (a), $\delta_1, \dots, \delta_{g^*}$ are contained in an SCC-prefix $\mathcal{J}_i$, so by (b), the *anchor* $w_{g^*}$ is the root or performs an **at**- or **af**-step in $\mathfrak{T}_i$. Moreover, $\delta_{g^*+1}, \dots, \delta_h$ reach no SCC, so they are pairwise different and we have $h - g^* \leq N$.

As the nodes created by a node $c$ have the probability at most $d \cdot p_c$ in total, we obtain

$$\sum_{w \in \mathrm{Junk}} p_w \;\leq\; (N+1) \cdot D^N \cdot \Big(1 + \sum_{1 \leq i \leq n} \mathrm{edl}_{\langle \mathcal{A}|_{\mathcal{J}_i}, \mathcal{A}|_{\mathcal{J}_i} \rangle}(\mathfrak{T}_i)\Big). \tag{8.25}$$

Since every **at**- or **af**-step of $\mathfrak{T}_i$ is one of $\mathfrak{T}$, we have $\mathrm{edl}_{\langle \mathcal{A}|_{\mathcal{J}_i}, \mathcal{A}|_{\mathcal{J}_i} \rangle}(\mathfrak{T}_i) \leq \mathrm{edl}_{\langle \mathcal{A}|_{\mathcal{J}_i}, \mathcal{S}|_{\mathcal{J}_i} \rangle}(\mathfrak{T}_i) + \mathrm{edl}_{\langle \mathcal{A}, \mathcal{A} \setminus \mathcal{S} \rangle}(\mathfrak{T})$. So by (8.24) and (8.25), $\mathrm{edl}_{\langle \mathcal{A}, \mathcal{S} \rangle}(\mathfrak{T})$ is bounded by a constant (independent of $\mathfrak{T}$) times $1 + \sum_{1 \leq i \leq n} \mathrm{edl}_{\langle \mathcal{A}|_{\mathcal{J}_i}, \mathcal{S}|_{\mathcal{J}_i} \rangle}(\mathfrak{T}_i) + \mathrm{edl}_{\langle \mathcal{A}, \mathcal{A} \setminus \mathcal{S} \rangle}(\mathfrak{T})$. Hence, by the facts on $\iota$ after Definition 8.1.15, we get $\iota_{\langle \mathcal{A}, \mathcal{S} \rangle} \sqsubseteq \iota_{\langle \mathcal{A}|_{\mathcal{J}_1}, \mathcal{S}|_{\mathcal{J}_1} \rangle} \sqcup \ldots \sqcup \iota_{\langle \mathcal{A}|_{\mathcal{J}_n}, \mathcal{S}|_{\mathcal{J}_n} \rangle} \sqcup \iota_{\langle \mathcal{A}, \mathcal{A} \setminus \mathcal{S} \rangle}$, and $\iota_{\langle \mathcal{A}, \mathcal{A} \setminus \mathcal{S} \rangle} \sqsubseteq c_{(v_1, \ldots, v_k)}$ by well-formedness of $\mathfrak{P}$.

▶ *Proof* ((8.12)): If $n = 0$, then for $\mathcal{S} = \mathcal{A}$, the above reasoning yields $\mathrm{edl}_{\langle \mathcal{A}, \mathcal{A} \rangle}(\mathfrak{T}) \leq (N+1) \cdot D^N$, i.e., $\iota_{\langle \mathcal{A}, \mathcal{A} \rangle} = \mathbf{Pol}_0$. Otherwise, every $\mathcal{A}|_{\mathcal{J}_i}$-CT can be viewed as an $\mathcal{A}$-CT by using some $\alpha \in \mathcal{A}$ with $\alpha|_{\mathcal{J}_i} = \gamma$ instead of every used ADP $\gamma$, i.e., the same ADP with (possibly) more annotations, where $\gamma \notin \mathcal{S}|_{\mathcal{J}_i}$ implies $\alpha \notin \mathcal{S}$. So $\iota_{\langle \mathcal{A}|_{\mathcal{J}_i}, \mathcal{A}|_{\mathcal{J}_i} \setminus \mathcal{S}|_{\mathcal{J}_i} \rangle} \sqsubseteq \iota_{\langle \mathcal{A}, \mathcal{A} \setminus \mathcal{S} \rangle} \sqsubseteq c_{(v_1, \ldots, v_k)}$.
Note that the proof only uses the edges in (a), so it also works for every over-approximation of the $\mathcal{A}$-dependency graph. *QED*

Example 8.2.10 (Dependency Graph Processor). Due to the two SCC-prefixes $\mathcal{J}_1 = \{(8.19), (8.20)\}$ and $\mathcal{J}_2 = \{(8.18), (8.21), (8.22)\}$, the dependency graph processor transforms the ADP problem $\langle \mathcal{A}_1, \mathcal{A}_1 \rangle$ from Example 8.2.3 into $\langle \mathcal{A}_1|_{\mathcal{J}_1}, \mathcal{A}_1|_{\mathcal{J}_1} \rangle$ (corresponding to the $\mathsf{geo}^\sharp$-SCC-prefix) with

$$\mathsf{start}(x,y) \overset{\mathsf{f}}{\hookrightarrow} \{{}^1\mathsf{q}(\mathsf{geo}^\sharp(x), y, y)\} \tag{8.26}$$
$$\mathsf{geo}(x) \overset{\mathsf{t}}{\hookrightarrow} \{{}^{1/2}\mathsf{geo}^\sharp(\mathsf{s}(x)),\ {}^{1/2}x\} \tag{8.27}$$
$$\mathsf{q}(\mathsf{s}(x), \mathsf{s}(y), z) \overset{\mathsf{f}}{\hookrightarrow} \{{}^1\mathsf{q}(x,y,z)\} \tag{8.28}$$
$$\mathsf{q}(x, 0, \mathsf{s}(z)) \overset{\mathsf{f}}{\hookrightarrow} \{{}^1\mathsf{s}(\mathsf{q}(x, \mathsf{s}(z), \mathsf{s}(z)))\} \tag{8.29}$$
$$\mathsf{q}(0, \mathsf{s}(y), \mathsf{s}(z)) \overset{\mathsf{f}}{\hookrightarrow} \{{}^1 0\}, \tag{8.30}$$

and $\langle \mathcal{A}_1|_{\mathcal{J}_2}, \mathcal{A}_1|_{\mathcal{J}_2} \rangle$ (corresponding to the $\mathsf{q}^\sharp$-SCC-prefix) with

$$\mathsf{start}(x,y) \overset{\mathsf{f}}{\hookrightarrow} \{{}^1\mathsf{q}^\sharp(\mathsf{geo}(x), y, y)\} \tag{8.31}$$
$$\mathsf{geo}(x) \overset{\mathsf{t}}{\hookrightarrow} \{{}^{1/2}\mathsf{geo}(\mathsf{s}(x)),\ {}^{1/2}x\} \tag{8.32}$$
$$\mathsf{q}(\mathsf{s}(x), \mathsf{s}(y), z) \overset{\mathsf{f}}{\hookrightarrow} \{{}^1\mathsf{q}^\sharp(x,y,z)\} \tag{8.33}$$
$$\mathsf{q}(x, 0, \mathsf{s}(z)) \overset{\mathsf{f}}{\hookrightarrow} \{{}^1\mathsf{s}(\mathsf{q}^\sharp(x, \mathsf{s}(z), \mathsf{s}(z)))\} \tag{8.34}$$
$$\mathsf{q}(0, \mathsf{s}(y), \mathsf{s}(z)) \overset{\mathsf{f}}{\hookrightarrow} \{{}^1 0\}. \tag{8.35}$$

As stated before, the dependency graph processor of this section combines several ideas from the literature in a single processor. For example, our dependency graph processor subsumes the effect of the "leaf removal processor" of [NEG13] on annotations. Leaves of the dependency graph like (8.23) are not part of any SCC. Hence, they are never contained in SCC-prefixes and thus, the annotations that only correspond to such leaves are always removed. For a similar reason, $\mathrm{Proc}_{\mathtt{DG}}$ from Theorem 8.2.9 subsumes the "rhs simplification processor" of [NEG13] and the related "usable terms processor" of Section 7.4.2, both of which share the same underlying idea.

In contrast to the dependency graph presented in Chapter 7, it does not suffice to only consider strongly connected components of the dependency graph. In particular, Counterexample 8.2.11 shows that only considering SCCs without prefixes would be unsound for analyzing complexity and proving `SAST`.

Counterexample 8.2.11 (Analyzing SCCs Without Their Prefixes Is Unsound). Recall $\mathcal{P}_{\mathsf{exp}}$ from Example 8.0.2 which is not `SAST` and $\mathcal{DP}(\mathcal{P}_{\mathsf{exp}})$ from Example 8.1.2. The $\mathcal{DP}(\mathcal{P}_{\mathsf{exp}})$-dependency graph is depicted in Figure 8.6. When only considering the SCCs $\mathcal{G}_{\mathsf{geo}^\sharp}$ (containing the $\mathsf{geo}^\sharp$-DP) and $\mathcal{G}_{\mathsf{f}^\sharp}$ (containing the three $\mathsf{f}^\sharp$-DPs), then we could falsely "prove" `SAST` since $\iota_{\langle \mathcal{DP}(\mathcal{P}_{\mathsf{exp}})|_{\mathcal{G}_{\mathsf{geo}^\sharp}}, \mathcal{DP}(\mathcal{P}_{\mathsf{exp}})|_{\mathcal{G}_{\mathsf{geo}^\sharp}} \rangle} = \mathbf{Pol}_0$ and

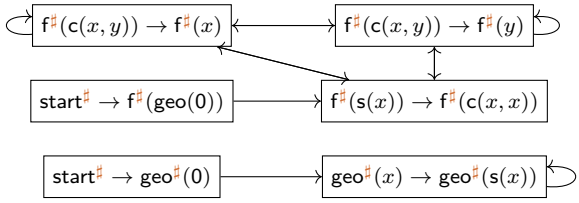



Figure 8.6: $\mathcal{DP}(\mathcal{P}_{\mathsf{exp}})$-dependency graph.

$\iota_{\langle \mathcal{DP}(\mathcal{P}_{\mathsf{exp}})|_{\mathcal{G}_{\mathsf{f}^\sharp}}, \mathcal{DP}(\mathcal{P}_{\mathsf{exp}})|_{\mathcal{G}_{\mathsf{f}^\sharp}} \rangle} = \mathbf{Exp}$. The problem is that for the $\mathsf{f}^\sharp$-SCC $\mathcal{G}_{\mathsf{f}^\sharp}$, one also has to consider the DP $\mathsf{start}^\sharp \to \mathsf{f}^\sharp(\mathsf{geo}(0))$ which determines with which instantiations of the variables the $\mathsf{f}^\sharp$-SCC is called (i.e., it ensures that the $\mathsf{geo}$-rules become usable). Indeed, we have $\iota_{\langle \mathcal{DP}(\mathcal{P}_{\mathsf{exp}}), \mathcal{DP}(\mathcal{P}_{\mathsf{exp}}) \rangle} = \mathbf{Inf}$.

### 8.2.3 Knowledge Propagation Processor

The dependency graph can not only be used to decompose an ADP problem $\langle \mathcal{A}, \mathcal{S} \rangle$ according to the SCC-prefixes via the dependency graph processor, but it can also be used to remove an ADP $\alpha$ from $\mathcal{S}$ if all "*predecessors*" of $\alpha$ have already been taken into account. More precisely, let $\mathrm{Pre}(\alpha) \subseteq \mathcal{A}$ contain all ADPs that can "generate" a redex for a step with $\alpha$ at an annotated position, i.e., $\mathrm{Pre}(\alpha)$ consists of all ADPs $\beta \in \mathcal{A}$ such that there is an edge from some DP in $\mathrm{dp}^\bot(\beta)$ to some DP in $\mathrm{dp}^\bot(\alpha)$ in the $\mathcal{A}$-dependency graph. Note that $\mathrm{dp}^\bot(\alpha) \neq \varnothing$ for all ADPs $\alpha$. If $d$ is the maximal number of annotated symbols in any term on a right-hand side of an ADP from $\mathcal{A}$, then in any $\mathcal{A}$-chain tree $\mathfrak{T}$, the probabilities of $\alpha$-steps can be over-approximated as follows. Except for the very first step, every **at**- or **af**-step with $\alpha$ is preceded by a step with some ADP $\beta$ from $\mathrm{Pre}(\alpha)$. Every term in $\beta$'s right-hand side can trigger at most $d$ $\alpha$-steps. If the $\beta$-step had probability $p$, then adding all probabilities for these $\alpha$-steps yields at most $d \cdot p$. Since the very first step of the tree might also be an $\alpha$-step, one obtains

$$\begin{aligned} \mathrm{edl}_{\langle \mathcal{A}, \{\alpha\} \rangle}(\mathfrak{T}) &\leq 1 + \textstyle\sum_{v \in N^{\mathfrak{T}} \setminus \mathrm{Leaf}^{\mathfrak{T}},\, \mathrm{Step}(v) \in \mathrm{Pre}(\alpha) \times \{\mathbf{at}, \mathbf{af}\}} d \cdot p_v \\ &= 1 + d \cdot \mathrm{edl}_{\langle \mathcal{A}, \mathrm{Pre}(\alpha) \rangle}(\mathfrak{T}). \end{aligned}$$

This in turn implies $\mathrm{eirc}_{\langle \mathcal{A}, \{\alpha\} \rangle}(n) \leq 1 + d \cdot \mathrm{eirc}_{\langle \mathcal{A}, \mathrm{Pre}(\alpha) \rangle}(n)$ for all $n \in \mathbb{N}$, and thus, $\iota_{\langle \mathcal{A}, \{\alpha\} \rangle} \sqsubseteq \iota_{\langle \mathcal{A}, \mathrm{Pre}(\alpha) \rangle}$.

Hence, if $\alpha \in \mathcal{S}$ and $\mathrm{Pre}(\alpha) \cap \mathcal{S} = \varnothing$ (i.e., $\mathrm{Pre}(\alpha) \subseteq \mathcal{A} \setminus \mathcal{S}$), then in any well-formed proof tree with a node $v$ where $L_{\mathcal{A}}(v) = \langle \mathcal{A}, \mathcal{S} \rangle$, the ADPs from $\mathrm{Pre}(\alpha)$ have already been taken into account in the path $v_1, \ldots, v_k = v$ from the root node $v_1$ to $v$, i.e.,

$$\iota_{\langle \mathcal{A}, \{\alpha\} \rangle} \sqsubseteq \iota_{\langle \mathcal{A}, \mathrm{Pre}(\alpha) \rangle} \sqsubseteq \iota_{\langle \mathcal{A}, \mathcal{A} \setminus \mathcal{S} \rangle} \sqsubseteq L_{\mathcal{C}}(v_1) \sqcup \cdots \sqcup L_{\mathcal{C}}(v_{k-1}).$$

As the proof tree already contains knowledge about $\mathrm{Pre}(\alpha)$'s complexity, the *knowledge propagation processor* removes $\alpha$ from $\mathcal{S}$. This processor is adapted from the knowledge propagation processor of [NEG13], which stores the ADPs whose complexity has already been taken into account in an additional component $\mathcal{K}$ (see Remark 8.1.20).

Theorem 8.2.12 – Knowledge Propagation Processor.

Let $\langle \mathcal{A}, \mathcal{S} \rangle$ be an ADP problem, let $\alpha \in \mathcal{S}$ and $\mathrm{Pre}(\alpha) \cap \mathcal{S} = \varnothing$, where $\mathrm{Pre}(\alpha)$ consists of all ADPs $\beta \in \mathcal{A}$ such that there is an edge from some DP in $\mathrm{dp}^{\perp}(\beta)$ to some DP in $\mathrm{dp}^{\perp}(\alpha)$ in the $\mathcal{A}$-dependency graph. Then the following processor is sound:

$$\mathrm{Proc}_{\mathtt{KP}}(\langle \mathcal{A}, \mathcal{S} \rangle) = (\mathbf{Pol}_0, \{\langle \mathcal{A}, \mathcal{S} \setminus \{\alpha\} \rangle\}).$$

*Proof.* Let $\mathfrak{P}$ be a well-formed proof tree with $L_{\mathcal{A}}(v) = \langle \mathcal{A}, \mathcal{S} \rangle$, let $v_1, \ldots, v_k = v$ be the path from the root node $v_1$ to $v$, and let $c_{(v_1,\ldots,v_k)} = L_{\mathcal{C}}(v_1) \sqcup \ldots \sqcup L_{\mathcal{C}}(v_{k-1})$. We first show that (8.11) holds, i.e.,

$$\iota_{\langle \mathcal{A}, \mathcal{S} \rangle} \sqsubseteq c_{(v_1,\ldots,v_k)} \sqcup \mathbf{Pol}_0 \sqcup \iota_{\langle \mathcal{A}, \mathcal{S} \setminus \{\alpha\} \rangle}.$$

Let $\mathfrak{T} = (N, E)$ be an $\mathcal{A}$-CT whose root is labeled with $t^{\sharp}$ for some $t \in \mathcal{T}_{\mathcal{B}}$. Let $d$ be the maximal number of annotated symbols in any term on a right-hand side of an ADP from $\mathcal{A}$. We show

$$\begin{aligned} \mathrm{edl}_{\langle \mathcal{A}, \{\alpha\} \rangle}(\mathfrak{T}) &\leq 1 + \textstyle\sum_{v \in N \setminus \mathrm{Leaf}^{\mathfrak{T}},\, \mathrm{Step}(v) \in \mathrm{Pre}(\alpha) \times \{\mathbf{at}, \mathbf{af}\}} d \cdot p_v \\ &= 1 + d \cdot \mathrm{edl}_{\langle \mathcal{A}, \mathrm{Pre}(\alpha) \rangle}(\mathfrak{T}). \end{aligned}$$

The root contributes at most $p_{\mathfrak{r}} = 1$. Every other node $w$ that performs an **at**- or **af**-step with $\alpha$ has a creator $v$, which performs an **at**- or **af**-step with some ADP $\beta$. As shown in the proof of Theorem 8.2.9, there is an edge from the DP of $w$, which is in $\mathrm{dp}(\beta)$, to every node of $\mathrm{dp}^{\perp}(\alpha)$, i.e., $\beta \in \mathrm{Pre}(\alpha)$. As observed at the beginning of this section, the probabilities of all nodes whose creator is $v$ add up to at most $d \cdot p_v$, which proves the inequality. This implies $\mathrm{eirc}_{\langle \mathcal{A}, \{\alpha\} \rangle}(n) \leq 1 + d \cdot \mathrm{eirc}_{\langle \mathcal{A}, \mathrm{Pre}(\alpha) \rangle}(n)$ for all $n \in \mathbb{N}$ and thus, $\iota_{\langle \mathcal{A}, \{\alpha\} \rangle} \sqsubseteq \iota_{\langle \mathcal{A}, \mathrm{Pre}(\alpha) \rangle}$. Hence, (8.11) holds, because of well-formedness of $\mathfrak{P}$, i.e.,

$$\begin{aligned} \iota_{\langle \mathcal{A}, \mathcal{S} \rangle} &= \iota_{\langle \mathcal{A}, \mathcal{S} \setminus \{\alpha\} \rangle} \sqcup \iota_{\langle \mathcal{A}, \{\alpha\} \rangle} \sqsubseteq \iota_{\langle \mathcal{A}, \mathcal{S} \setminus \{\alpha\} \rangle} \sqcup \iota_{\langle \mathcal{A}, \mathrm{Pre}(\alpha) \rangle} \\ &\sqsubseteq \iota_{\langle \mathcal{A}, \mathcal{S} \setminus \{\alpha\} \rangle} \sqcup \iota_{\langle \mathcal{A}, \mathcal{A} \setminus \mathcal{S} \rangle} \sqsubseteq \iota_{\langle \mathcal{A}, \mathcal{S} \setminus \{\alpha\} \rangle} \sqcup c_{(v_1,\ldots,v_k)}. \end{aligned}$$

Now we show (8.12), i.e.,

$$\iota_{\langle \mathcal{A}, \mathcal{A} \setminus (\mathcal{S} \setminus \{\alpha\}) \rangle} \sqsubseteq c_{(v_1,\ldots,v_k)} \sqcup \mathbf{Pol}_0.$$

This holds again because $\mathfrak{P}$ is well formed:

$$\begin{aligned} \iota_{\langle \mathcal{A}, \mathcal{A} \setminus (\mathcal{S} \setminus \{\alpha\}) \rangle} &= \iota_{\langle \mathcal{A}, (\mathcal{A} \setminus \mathcal{S}) \cup \{\alpha\} \rangle} = \iota_{\langle \mathcal{A}, \mathcal{A} \setminus \mathcal{S} \rangle} \sqcup \iota_{\langle \mathcal{A}, \{\alpha\} \rangle} \\ &\sqsubseteq c_{(v_1,\ldots,v_k)} \sqcup c_{(v_1,\ldots,v_k)} = c_{(v_1,\ldots,v_k)}. \end{aligned}$$

*Proof* *QED*

Example 8.2.13 (Knowledge Propagation Processor). Applying the reduction pair processor to $\langle \mathcal{A}_1|_{\mathcal{J}_2}, \mathcal{A}_1|_{\mathcal{J}_2} \rangle$ from Example 8.2.10 results in the ADP problem $\langle \mathcal{A}_1|_{\mathcal{J}_2}, \{(8.34)\} \rangle$ (see Example 8.3.3 in Section 8.3.1 below). We have $\mathrm{dp}^{\perp}((8.34)) = \{(8.22)\}$, and the only DP with an edge to (8.22) in the $\mathcal{A}_1|_{\mathcal{J}_2}$-dependency graph is (8.21) (the edges between the $\mathtt{q}^{\sharp}$-DPs are the same as in Figure 8.5), where $(8.21) \in \mathrm{dp}^{\perp}((8.33))$. Thus, $\mathrm{Pre}((8.34)) = \{(8.33)\}$, i.e., in particular $(8.34) \notin \mathrm{Pre}((8.34))$. Hence, we can apply the knowledge propagation processor and obtain $\mathrm{Proc}_{\mathtt{KP}}(\langle \mathcal{A}_1|_{\mathcal{J}_2}, \{(8.34)\} \rangle) = (\mathbf{Pol}_0, \{\langle \mathcal{A}_1|_{\mathcal{J}_2}, \varnothing \rangle\})$, again containing a solved ADP problem.

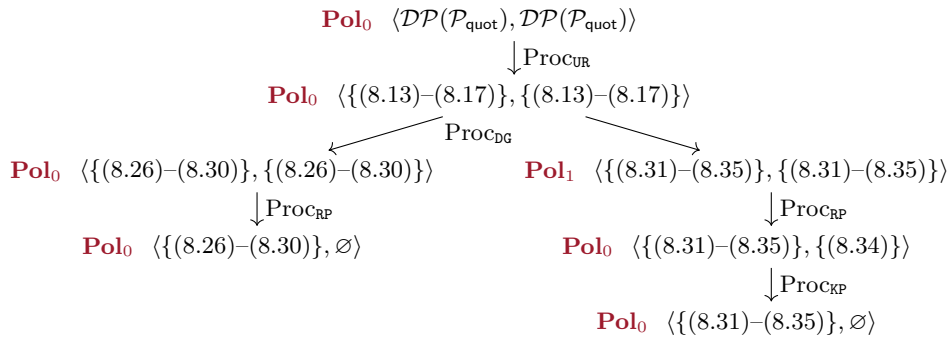



Figure 8.7: Solved proof tree for $\mathcal{P}_{\mathsf{quot}}$.

The solved proof tree is shown in Figure 8.7. Thus, we infer that the expected innermost runtime complexity of $\mathcal{P}_{\mathsf{quot}}$ is at most linear, i.e., $\iota_{\mathcal{P}_{\mathsf{quot}}} \sqsubseteq \mathbf{Pol}_1$, and in particular, innermost evaluation of $\mathcal{P}_{\mathsf{quot}}$ is `SAST` w.r.t. basic start terms.

Note that in Chapter 7 we did not have to add such a processor, because a node in the dependency graph whose predecessors have all already been successfully analyzed is no longer part of any SCC, so its annotations are removed by the dependency graph processor. While the ADP framework for `AST` removes annotations from ADPs that have been successfully analyzed, the ADP framework for `SAST` keeps the annotations and only removes the ADP from $\mathcal{S}$, keeping the SCCs in the dependency graph intact.

SECTION 8.3

## Processors Based on Algebras

In this section, we present an adaptation of the *reduction pair processor* that lifts the direct application of algebras to ADP problems. As for proving `AST` in Section 7.5.1, we phrase the processor for arbitrary collapsible barycentric $\Sigma^\sharp$-algebras (see Definitions 3.4.30 and 3.4.31), so that the semimodule operations $\oplus$ and $\odot$ replace the addition and multiplication used in [ADLY20]. The resulting theorem yields an abstract upper bound on the (innermost) runtime complexity. We then instantiate the algebra with polynomial interpretations, which allows us to infer a concrete complexity bound.

### 8.3.1 Reduction Pair Processor

Now we lift the direct application of algebras explained in Section 3.4.5 to an ADP processor for complexity analysis of PTRSs, which allows us to apply a collapsible barycentric $\Sigma^\sharp$-algebra $(\mathfrak{I}, \succ)$ in a modular way. As in the classical DP approach [AG00, GTSF06], here it suffices if $\mathfrak{I}$ is *weakly* monotonic, i.e., if $x \succcurlyeq y$ implies $f_\mathfrak{I}(\dots, x, \dots) \succcurlyeq f_\mathfrak{I}(\dots, y, \dots)$ for all $f \in \Sigma^\sharp$ and $x, y \in A$, where $A$ is the carrier of the algebra. Moreover, as in Section 7.5.1, to ensure "weak monotonicity w.r.t. expected values" we require concavity of all functions $f_\mathfrak{I}$. As discussed in Section 3.4.5, natural polynomial interpretations are such algebras, provided that all polynomials are multilinear (see Definition 3.4.32). The same holds for matrix interpretations as in [EWZ06]. As before, the expected value of a distribution $\mu = \{{}^{p_1}t_1, \dots, {}^{p_k}t_k\}$ of

carrier elements is $\mathbb{E}(\mu) = \bigoplus_{i=1}^{k} p_i \odot t_i$.

The *reduction pair processor* imposes the following requirements on $\mathfrak{I}$.

(1) All ADPs with the flag $\mathtt{t}$ must be weakly decreasing in expectation when removing all annotations. Due to weak monotonicity, this ensures that evaluating the arguments of a function call (i.e., applying the rule in a context) also decreases weakly in expectation.

(2) All ADPs must be weakly decreasing when comparing the annotated left-hand side $\mathfrak{I}(\ell^\sharp)$ with the expected value of the annotated subterms of the right-hand side $\{{}^{p_1}r_1, \dots, {}^{p_k}r_k\}$. To measure the value of a term $r_j$, here we consider all its subterms $t \trianglelefteq_\sharp r_j$ at annotated positions and sum up the interpretations of all such $t^\sharp$ w.r.t. the semimodule addition $\oplus$, i.e., we consider $\mathfrak{I}^\sharp_{sum}(r_j) = \bigoplus_{t \trianglelefteq_\sharp r_j} \mathfrak{I}(t^\sharp)$, where, as in Section 7.5.1, the sum contains one summand for every annotated position of $r_j$. Regarding this sum instead of the interpretation $\mathfrak{I}(r_j)$ of the whole term $r_j$ is the reason why we only need *weak* monotonicity, as in Section 7.5.1.

(3) The ADPs that shall be removed from $\mathcal{S}$ must even be strictly decreasing with a uniform gap $\epsilon > 0$ w.r.t. $\succ_\epsilon$ when comparing these values.

Then, the processor removes all strictly decreasing ADPs from the component $\mathcal{S}$ of the ADP problem. However, the ADPs are still kept in $\mathcal{A}$, because they may still be used in reductions. The processor returns an abstract complexity bound $c = \iota(g_\mathfrak{I})$ that is determined by the growth of the (embedded) interpretation of basic terms that are annotated at the root. When $\mathfrak{I}$ is a polynomial interpretation, this abstract bound becomes a concrete polynomial, exponential, or doubly exponential bound, as we show in Corollary 8.3.2.

Theorem 8.3.1 – Reduction Pair Processor.

Let $(\mathfrak{I}, \succ)$ be a weakly monotonic, collapsible barycentric $\Sigma^\sharp$-algebra with embedding $\mathrm{emb} : A \to \mathbb{R}_{\geq 0}$, and let $\epsilon > 0$. Let $\langle \mathcal{A}, \mathcal{S} \rangle$ be an ADP problem where $\mathcal{A} = \mathcal{A}_{\succcurlyeq} \uplus \mathcal{A}_{\succ}$ and $\mathcal{A}_{\succ} \cap \mathcal{S} \neq \varnothing$ such that:

$$
\begin{array}{lllll}
(1) & \forall \ell \xhookrightarrow{\mathtt{t}} \{{}^{p_1}r_1, \dots, {}^{p_k}r_k\} \in \mathcal{A}. & \mathfrak{I}(\ell) & \succcurlyeq & \bigoplus_{1 \leq j \leq k} p_j \odot \mathfrak{I}(\flat(r_j)), \\
(2) & \forall \ell \xhookrightarrow{m} \{{}^{p_1}r_1, \dots, {}^{p_k}r_k\} \in \mathcal{A}_{\succcurlyeq}. & \mathfrak{I}(\ell^\sharp) & \succcurlyeq & \bigoplus_{1 \leq j \leq k} p_j \odot \mathfrak{I}^\sharp_{sum}(r_j), \\
(3) & \forall \ell \xhookrightarrow{m} \{{}^{p_1}r_1, \dots, {}^{p_k}r_k\} \in \mathcal{A}_{\succ}. & \mathfrak{I}(\ell^\sharp) & \succ_\epsilon & \bigoplus_{1 \leq j \leq k} p_j \odot \mathfrak{I}^\sharp_{sum}(r_j),
\end{array}
$$

where $\mathfrak{I}^\sharp_{sum}(r_j) = \bigoplus_{t \trianglelefteq_\sharp r_j} \mathfrak{I}(t^\sharp)$.
Then $\mathrm{Proc}_{\mathtt{RP}}(\langle \mathcal{A}, \mathcal{S} \rangle) = (c, \{\langle \mathcal{A}, \mathcal{S} \setminus \mathcal{A}_{\succ} \rangle\})$ is sound, where the complexity $c \in \mathfrak{C}$ is the abstract bound $c = \iota(g_\mathfrak{I})$ with $g_\mathfrak{I}(n) = \sup\{\mathrm{emb}(\mathfrak{I}^\sharp_0(t^\sharp)) \mid t \in \mathcal{T}_\mathcal{B} \text{ and } |t| \leq n\}$, and $\mathfrak{I}^\sharp_0$ behaves like $\mathfrak{I}^\sharp_{sum}$ but maps every variable to $\mathbf{0}$.

*Proof.* As in Section 7.5.1, the conditions (1), (2), (3) from Theorem 8.3.1 can be lifted to rewrite steps with $\xhookrightarrow{\mathtt{i}}_\mathcal{A}$ instead of just rules and, therefore, to edges of a CT. More precisely, the weak Cases 1 and 2 are lifted analogously to Cases (c) and (d) in the proof of Lemma 7.5.2 (using weak monotonicity and concavity of the $f_\mathfrak{I}$). The strict Case 3 combines the annotation bookkeeping of Cases (b) and (d) of Lemma 7.5.2 with the barycentric shape of Requirement (3), where the concluding weak steps are absorbed by $\succ_\epsilon$ (see the remark below Definition 3.4.31). Concretely, one obtains the following for every step $s \xhookrightarrow{\mathtt{i}}_\mathcal{A} \{{}^{p_1}t_1, \dots, {}^{p_k}t_k\}$. *Proof*

1. For every $s \overset{\mathtt{i}}{\hookrightarrow}_{\mathcal{A}} \mu$ with **nt** or **nf**, we have
$$\mathfrak{I}^{\sharp}_{sum}(s) \succcurlyeq \bigoplus_{1\le j\le k} p_j \odot \mathfrak{I}^{\sharp}_{sum}(t_j).$$
2. For every $s \overset{\mathtt{i}}{\hookrightarrow}_{\mathcal{A}} \mu$ with **at** or **af** and an ADP from $\mathcal{A}_{\succcurlyeq}$, we have
$$\mathfrak{I}^{\sharp}_{sum}(s) \succcurlyeq \bigoplus_{1\le j\le k} p_j \odot \mathfrak{I}^{\sharp}_{sum}(t_j).$$
3. For every $s \overset{\mathtt{i}}{\hookrightarrow}_{\mathcal{A}} \mu$ with **at** or **af** and an ADP from $\mathcal{A}_{\succ}$, we have
$$\mathfrak{I}^{\sharp}_{sum}(s) \succ_{\epsilon} \bigoplus_{1\le j\le k} p_j \odot \mathfrak{I}^{\sharp}_{sum}(t_j).$$

Remember that $\mathfrak{I}^{\sharp}_0$ behaves exactly as $\mathfrak{I}^{\sharp}_{sum}$ but in addition, it maps every variable to $\mathbf{0}$. Let $\mathrm{emb}' = {}^1\!/_{\epsilon} \cdot \mathrm{emb}$, which is again monotone and concave, so that $a \succ_{\epsilon} b$ implies $\mathrm{emb}'(a) \ge \mathrm{emb}'(b) + 1$. We measure every annotated term $s$ by $\mathcal{V}al(s) = \mathrm{emb}'(\mathfrak{I}^{\sharp}_0(s)) \in \mathbb{R}_{\ge 0}$, so that $g'_{\mathfrak{I}}(n) = \sup\{\mathcal{V}al(t^{\sharp}) \mid t \in \mathcal{T}_{\mathcal{B}} \text{ and } |t| \le n\} = {}^1\!/_{\epsilon} \cdot g_{\mathfrak{I}}(n)$. Since the constant factor ${}^1\!/_{\epsilon}$ does not affect the complexity, we have $\iota(g'_{\mathfrak{I}}) = \iota(g_{\mathfrak{I}}) = c$. We now embed these inequalities into $\mathbb{R}_{\ge 0}$ via $\mathrm{emb}'$, where we instantiate every variable by $\mathbf{0}$ (which is admissible since the inequalities hold for all instantiations; for the strict Case 3, both the $\succ$- and the emb-gap component of $\succ_{\epsilon}$ are preserved). Since $\mathrm{emb}'$ is monotone and concave, i.e., $\mathrm{emb}'(\bigoplus_{1\le j\le k} p_j \odot a_j) \ge \sum_{1\le j\le k} p_j \cdot \mathrm{emb}'(a_j)$, Cases 1 and 2 yield the *weak* decrease $\mathcal{V}al(s) \ge \sum_{1\le j\le k} p_j \cdot \mathcal{V}al(t_j)$. For Case 3, the strict decrease $\mathfrak{I}^{\sharp}_0(s) \succ_{\epsilon} \bigoplus_{1\le j\le k} p_j \odot \mathfrak{I}^{\sharp}_0(t_j)$ provides, by the gap of $\succ_{\epsilon}$ (rescaled to 1), that $\mathrm{emb}'(\mathfrak{I}^{\sharp}_0(s)) \ge \mathrm{emb}'(\bigoplus_{1\le j\le k} p_j \odot \mathfrak{I}^{\sharp}_0(t_j)) + 1$; together with concavity of $\mathrm{emb}'$ we obtain the *strict* decrease

$$\mathcal{V}al(s) \ge 1 + \textstyle\sum_{1\le j\le k} p_j \cdot \mathcal{V}al(t_j).$$

This is exactly where collapsibility is needed: the $\epsilon$-decrease guarantees a uniform minimal drop of $\epsilon > 0$ (rescaled to 1) in the embedding for every $\mathcal{A}_{\succ}$-step, even though the underlying carrier $A$ may not have such a uniform minimal decrease.
We only have to prove that the complexity of $\mathcal{A}$ when counting only $\mathcal{A}_{\succ}$-ADPs is bounded by $c$, i.e., $\iota_{\langle \mathcal{A}, \mathcal{A}_{\succ}\rangle} \sqsubseteq c$.
Afterwards, we can conclude that (8.11) holds, i.e.,

$$\iota_{\langle \mathcal{A}, \mathcal{S}\rangle} \sqsubseteq c \sqcup \iota_{\langle \mathcal{A}, \mathcal{S}\setminus\mathcal{A}_{\succ}\rangle},$$

because

$$\iota_{\langle \mathcal{A}, \mathcal{S}\rangle} = \iota_{\langle \mathcal{A}, \mathcal{S}\cap\mathcal{A}_{\succ}\rangle} \sqcup \iota_{\langle \mathcal{A}, \mathcal{S}\setminus\mathcal{A}_{\succ}\rangle} \sqsubseteq \iota_{\langle \mathcal{A}, \mathcal{A}_{\succ}\rangle} \sqcup \iota_{\langle \mathcal{A}, \mathcal{S}\setminus\mathcal{A}_{\succ}\rangle} \sqsubseteq c \sqcup \iota_{\langle \mathcal{A}, \mathcal{S}\setminus\mathcal{A}_{\succ}\rangle}.$$

Similarly, one can also conclude that (8.12) holds. Let $\mathfrak{P}$ be a well-formed proof tree with $L_{\mathcal{A}}(v) = \langle \mathcal{A}, \mathcal{S}\rangle$, let $v_1, \ldots, v_k = v$ be the path from the root node $v_1$ to $v$, and let $c_{(v_1,\ldots,v_k)} = L_{\mathcal{C}}(v_1) \sqcup \cdots \sqcup L_{\mathcal{C}}(v_{k-1})$. Then we have

$$\iota_{\langle \mathcal{A}, \mathcal{A}\setminus(\mathcal{S}\setminus\mathcal{A}_{\succ})\rangle} \sqsubseteq c_{(v_1,\ldots,v_k)} \sqcup c$$

because $\iota_{\langle \mathcal{A}, \mathcal{A}\setminus(\mathcal{S}\setminus\mathcal{A}_{\succ})\rangle} \sqsubseteq \iota_{\langle \mathcal{A}, (\mathcal{A}\setminus\mathcal{S})\cup\mathcal{A}_{\succ}\rangle}$ and

$$\iota_{\langle \mathcal{A}, (\mathcal{A}\setminus\mathcal{S})\cup\mathcal{A}_{\succ}\rangle} = \iota_{\langle \mathcal{A}, \mathcal{A}\setminus\mathcal{S}\rangle} \sqcup \iota_{\langle \mathcal{A}, \mathcal{A}_{\succ}\rangle} \sqsubseteq c_{(v_1,\ldots,v_k)} \sqcup c.$$

So it remains to show that $\iota_{\langle \mathcal{A}, \mathcal{A}_\succ \rangle} \sqsubseteq c$ holds. Consider a basic term $t$ and an $\mathcal{A}$-CT $\mathfrak{T} = (N, E)$ with $t^\sharp$ at the root. We want to show that $\mathcal{V}al(t^\sharp) = \mathrm{emb}'(\mathfrak{I}_0^\sharp(t^\sharp))$ is a bound on the expected derivation length $\mathrm{edl}_{\langle \mathcal{A}, \mathcal{A}_\succ \rangle}(\mathfrak{T})$.
We have $\mathcal{V}al(t^\sharp)$ as the initial value at the root. Whenever we perform a rewrite step with **nt** or **nf** or with an ADP from $\mathcal{A}_{\succcurlyeq}$, we weakly decrease the value in expectation. And whenever we perform a rewrite step with **at** or **af** and an ADP from $\mathcal{A}_\succ$ at a node $v$, we strictly decrease the value by at least $1 \cdot p_v = p_v$ in expectation, as shown above. This behavior is illustrated in Figure 8.8. To make this precise, for every $h \in \mathbb{N}$, let $N_{<h}$ be the set of all inner nodes $v$ of depth $< h$ with $\mathrm{Step}(v) \in \mathcal{A}_\succ \times \{\mathbf{at}, \mathbf{af}\}$, and let $L_h$ be the set of all nodes of depth $h$ together with all leaves of depth $< h$. By induction on $h$, we obtain

$$\mathcal{V}al(t^\sharp) \;\geq\; \textstyle\sum_{v \in N_{<h}} p_v + \sum_{x \in L_h} p_x \cdot \mathcal{V}al(t_x),$$

since for every inner node $x$ with the children $y_1, \ldots, y_k$, we have $p_{y_j} = p_x \cdot p_j$, and hence the above (weak or strict) decrease yields $p_x \cdot \mathcal{V}al(t_x) \geq \sum_{1 \leq j \leq k} p_{y_j} \cdot \mathcal{V}al(t_{y_j})$, and even $p_x \cdot \mathcal{V}al(t_x) \geq p_x + \sum_{1 \leq j \leq k} p_{y_j} \cdot \mathcal{V}al(t_{y_j})$ if $\mathrm{Step}(x) \in \mathcal{A}_\succ \times \{\mathbf{at}, \mathbf{af}\}$. As $\mathcal{V}al(t_x) \geq 0$ for all nodes $x$, this implies $\sum_{v \in N_{<h}} p_v \leq \mathcal{V}al(t^\sharp)$ for all $h \in \mathbb{N}$, and therefore

$$\mathrm{edl}_{\langle \mathcal{A}, \mathcal{A}_\succ \rangle}(\mathfrak{T}) \;=\; \sum_{\substack{v \in N \setminus \mathrm{Leaf}^{\mathfrak{T}}, \\ \mathrm{Step}(v) \in \mathcal{A}_\succ \times \{\mathbf{at}, \mathbf{af}\}}} p_v \;=\; \sup_{h \in \mathbb{N}} \sum_{v \in N_{<h}} p_v \;\leq\; \mathcal{V}al(t^\sharp). \tag{8.36}$$

Hence, we obtain $\mathrm{edl}_{\langle \mathcal{A}, \mathcal{A}_\succ \rangle}(\mathfrak{T}) \leq \mathcal{V}al(t^\sharp) \leq g'_{\mathfrak{I}}(|t|)$ and thus, $\iota_{\langle \mathcal{A}, \mathcal{A}_\succ \rangle} \sqsubseteq \iota(g'_{\mathfrak{I}}) = c$. *QED*

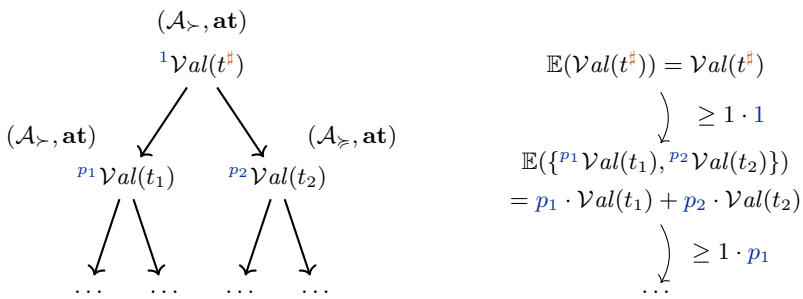



Figure 8.8: Expected decrease of $\mathcal{V}al(t^\sharp) = \mathrm{emb}(\mathfrak{I}_0^\sharp(t^\sharp))$ in a chain tree.

In contrast to the reduction pair processor for proving AST from Section 7.5.1, our processor cannot remove any annotation from $\mathcal{A}$. The reason is that the ADPs from $\mathcal{A}_\succ$ may still be needed to reach all annotated terms that are relevant for the complexity of $\mathcal{A}_{\succcurlyeq}$. (This problem already occurs when analyzing complexity in the non-probabilistic setting, see, e.g., [NEG13, Ex. 25].) But we can remove strictly decreasing ADPs from $\mathcal{S}$ and, therefore, do not have to count them anymore for the complexity. The complexity of the removed ADPs is accounted for by the abstract bound $c = \iota(g_{\mathfrak{I}})$.

To search for a suitable algebra fully automatically and to turn the abstract bound $c = \iota(g_{\mathfrak{I}})$ into a concrete complexity class, we instantiate Theorem 8.3.1 with polynomial interpretations. As in Corollaries 3.4.34 and 3.4.36, every natural, weakly monotonic, and multilinear polynomial interpretation Pol is a weakly monotonic barycentric

collapsible $\Sigma^\sharp$-algebra with carrier $\mathbb{R}_{\geq 0}$ and the monotone, affine identity embedding $\mathrm{emb} = id : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ (for which the $\epsilon$-decrease $\succ_\epsilon$ is the standard gap order $>_\epsilon$), so that $\oplus$ and $\odot$ become the standard addition $+$ and multiplication $\cdot$, and $\succcurlyeq$ and $\succ$ become $\geq$ and $>$. For such interpretations, $g_{\mathrm{Pol}}(n) = \sup\{\mathrm{Pol}_0^\sharp(t^\sharp) \mid t \in \mathcal{T}_\mathcal{B} \text{ and } |t| \leq n\}$ is the function that maps $n$ to the maximal interpretation of any basic term of size $\leq n$ which is annotated at the root. For CPIs, the degrees of the polynomials used for the annotated symbols determine a concrete complexity class for $\iota(g_{\mathrm{Pol}})$.

Corollary 8.3.2 – Reduction Pair Processor via Polynomial Interpretations.

*Let* $\mathrm{Pol} : \Sigma^\sharp \to \mathbb{N}[\mathcal{V}]$ *be a weakly monotonic, multilinear polynomial interpretation, let* $\epsilon > 0$, *and let* $\mathrm{Pol}^\sharp_{sum}(r_j) = \sum_{t \trianglelefteq_\sharp r_j} \mathrm{Pol}(t^\sharp)$. *Let* $\langle \mathcal{A}, \mathcal{S} \rangle$ *be an ADP problem where* $\mathcal{A} = \mathcal{A}_\geq \uplus \mathcal{A}_>$ *and* $\mathcal{A}_> \cap \mathcal{S} \neq \varnothing$ *such that:*

$$
\begin{array}{lllll}
(1) & \forall \ell \xhookrightarrow{t} \{p_1 r_1, \ldots, p_k r_k\} \in \mathcal{A}. & \mathrm{Pol}(\ell) & \geq & \sum_{1 \leq j \leq k} p_j \cdot \mathrm{Pol}(\flat(r_j)), \\
(2) & \forall \ell \xhookrightarrow{m} \{p_1 r_1, \ldots, p_k r_k\} \in \mathcal{A}_\geq. & \mathrm{Pol}(\ell^\sharp) & \geq & \sum_{1 \leq j \leq k} p_j \cdot \mathrm{Pol}^\sharp_{sum}(r_j), \\
(3) & \forall \ell \xhookrightarrow{m} \{p_1 r_1, \ldots, p_k r_k\} \in \mathcal{A}_>. & \mathrm{Pol}(\ell^\sharp) & >_\epsilon & \sum_{1 \leq j \leq k} p_j \cdot \mathrm{Pol}^\sharp_{sum}(r_j).
\end{array}
$$

*Then* $\mathrm{Proc}_{\mathtt{RP}}(\langle \mathcal{A}, \mathcal{S} \rangle) = (c, \{\langle \mathcal{A}, \mathcal{S} \setminus \mathcal{A}_> \rangle\})$ *is sound, where the complexity* $c \in \mathfrak{C}$ *is determined as follows: If* Pol *is a CPI and for all* $f^\sharp \in \mathcal{D}^\sharp$, *the polynomial* $f^\sharp_{\mathrm{Pol}}$ *has at most degree* $a$, *then* $c = \mathbf{Pol}_a$. *If* Pol *is not a CPI, then* $c = \mathbf{Exp}$ *if all constructors are interpreted by linear polynomials, and otherwise* $c = \mathbf{2\text{-}Exp}$.

*Proof.* $(\mathrm{Pol}, >)$ is a weakly monotonic, collapsible barycentric $\Sigma^\sharp$-algebra with the identity embedding emb, so that $\mathfrak{I}^\sharp_{sum}$ becomes $\mathrm{Pol}^\sharp_{sum}$. Under this instantiation, Constraints (1)–(3) of the corollary are exactly Constraints (1)–(3) of Theorem 8.3.1, since $\bigoplus_{1 \leq j \leq k} p_j \odot \mathrm{Pol}^\sharp_{sum}(r_j) = \sum_{1 \leq j \leq k} p_j \cdot \mathrm{Pol}^\sharp_{sum}(r_j)$ and analogously for $\mathrm{Pol}(\flat(\cdot))$. Hence, by Theorem 8.3.1, $\mathrm{Proc}_{\mathtt{RP}}(\langle \mathcal{A}, \mathcal{S} \rangle) = (\iota(g_{\mathrm{Pol}}), \{\langle \mathcal{A}, \mathcal{S} \setminus \mathcal{A}_> \rangle\})$ is sound. Since emb is the identity, we have $g_{\mathrm{Pol}}(n) = \sup\{\mathrm{Pol}_0^\sharp(t^\sharp) \mid t \in \mathcal{T}_\mathcal{B} \text{ and } |t| \leq n\}$, and it remains to show $\iota(g_{\mathrm{Pol}}) \sqsubseteq c$. This follows from the analysis of the size of $\mathrm{Pol}_0(t)$ for basic terms $t$ in Section 3.2.5 (which is also used for Corollary 3.4.36). Instead of basic terms, here we only need to consider terms that have an annotation at the root, and constructor terms as proper subterms. Thus, if Pol is a CPI whose annotated symbols are interpreted by polynomials of degree at most $a$, then $g_{\mathrm{Pol}}(n)$ grows at most polynomially with degree $a$, i.e., $\iota(g_{\mathrm{Pol}}) \sqsubseteq \mathbf{Pol}_a$. If Pol is not a CPI, then $g_{\mathrm{Pol}}(n)$ grows at most exponentially if all constructors are interpreted by linear polynomials ($\iota(g_{\mathrm{Pol}}) \sqsubseteq \mathbf{Exp}$), and at most doubly exponentially otherwise ($\iota(g_{\mathrm{Pol}}) \sqsubseteq \mathbf{2\text{-}Exp}$), see Section 3.2.5. *Proof* *QED*

Using the reduction pair processor, we can continue the complexity analysis of $\mathcal{P}_{\mathsf{quot}}$.

Example 8.3.3 (Reduction Pair Processor). Consider $\langle \mathcal{A}_1|_{\mathcal{J}_1}, \mathcal{A}_1|_{\mathcal{J}_1} \rangle$ from Example 8.2.10, and a polynomial interpretation Pol with $\mathsf{start}^\sharp_{\mathrm{Pol}}(x, y) = 2$, $\mathsf{geo}^\sharp_{\mathrm{Pol}}(x) = \mathsf{q}^\sharp_{\mathrm{Pol}}(x, y, z) = 1$, $\mathsf{s}_{\mathrm{Pol}}(x) = \mathsf{geo}_{\mathrm{Pol}}(x) = x+1$, and $f_{\mathrm{Pol}} = 0$ for all other symbols $f$. Then $\mathsf{geo}(x) \xhookrightarrow{\mathsf{t}} \{{}^{1}/_{2}\mathsf{geo}^\sharp(\mathsf{s}(x)), {}^{1}/_{2}x\}$ (8.27) is weakly decreasing in expectation when disregarding annotations, since $\mathrm{Pol}(\mathsf{geo}(x)) = x+1 = {}^{1}/_{2} \cdot \mathrm{Pol}(\mathsf{geo}(\mathsf{s}(x))) + {}^{1}/_{2} \cdot \mathrm{Pol}(x)$. Moreover, when regarding annotations, then all ADPs are strictly decreasing: for (8.27) we have $\mathrm{Pol}(\mathsf{geo}^\sharp(x)) = 1 > {}^{1}/_{2} = {}^{1}/_{2} \cdot \mathrm{Pol}(\mathsf{geo}^\sharp(\mathsf{s}(x)))$, for $\mathsf{start}(x, y) \xhookrightarrow{\mathsf{f}} \{{}^{1}\mathsf{q}(\mathsf{geo}^\sharp(x), y, y)\}$ (8.26) we have $\mathrm{Pol}(\mathsf{start}^\sharp(x, y)) = 2 > 1 = \mathrm{Pol}(\mathsf{geo}^\sharp(x))$, and for (8.28)–(8.30) we have $\mathrm{Pol}(\mathsf{q}^\sharp(\ldots)) = 1 > 0$ (as their right-hand sides do not contain annotations), so Con-

straint (3) holds with $\epsilon = 1/2$. Since Pol is a CPI which interprets all annotated symbols as constants, we obtain $\mathrm{Proc}_{\mathtt{RP}}(\langle \mathcal{A}_1|_{\mathcal{J}_1}, \mathcal{A}_1|_{\mathcal{J}_1}\rangle) = (\mathbf{Pol}_0, \{\langle \mathcal{A}_1|_{\mathcal{J}_1}, \varnothing\rangle\})$, containing a solved ADP problem.

For the other ADP problem $\langle \mathcal{A}_1|_{\mathcal{J}_2}, \mathcal{A}_1|_{\mathcal{J}_2}\rangle$ from Example 8.2.10, we use a polynomial interpretation with $\mathsf{start}^\sharp_{\mathrm{Pol}}(x,y) = x+3$, $\mathsf{geo}^\sharp_{\mathrm{Pol}}(x) = 1$, $\mathsf{q}^\sharp_{\mathrm{Pol}}(x,y,z) = \mathsf{s}_{\mathrm{Pol}}(x) = \mathsf{geo}_{\mathrm{Pol}}(x) = x+1$, and $f_{\mathrm{Pol}} = 0$ for all other symbols $f$. Then (8.32) is again weakly decreasing when disregarding annotations. When regarding the annotations, then the ADP (8.34) is weakly decreasing (since $\mathrm{Pol}(\mathsf{q}^\sharp(x,\mathsf{0},\mathsf{s}(z))) = x+1 = \mathrm{Pol}(\mathsf{q}^\sharp(x,\mathsf{s}(z),\mathsf{s}(z)))$), and all other ADPs are strictly decreasing (with a uniform gap of 1, so Constraint (3) holds with $\epsilon = 1$). Since Pol is a CPI where $\mathsf{start}^\sharp$ and $\mathsf{q}^\sharp$ are interpreted as linear polynomials, we get $\mathrm{Proc}_{\mathtt{RP}}(\langle \mathcal{A}_1|_{\mathcal{J}_2}, \mathcal{A}_1|_{\mathcal{J}_2}\rangle) = (\mathbf{Pol}_1, \{\langle \mathcal{A}_1|_{\mathcal{J}_2}, \{(8.34)\}\rangle\})$. However, there is no polynomial interpretation which orients (8.34) strictly and the other ADPs weakly. Thus, we need another processor to solve the remaining problem.

SECTION 8.4

## Transformational Processors

Finally, we show how to adapt transformational processors (see Section 7.6) to our novel ADP framework. For brevity, we only consider the *narrowing processor*.

We briefly recapitulate the core idea behind the narrowing processor. Let $\mathcal{A} = \mathcal{A}' \uplus \{\ell \xhookrightarrow{m} \{{}^{p_1}r_1, \ldots, {}^{p_k}r_k\}\}$ be a set of ADPs and let $t \trianglelefteq_\sharp r_j$ for some $1 \le j \le k$. If we have to perform rewrite steps on (an instance of) $t$ in order to enable the next application of an ADP at an annotated position, then the idea of the narrowing processor is to perform the first step of this reduction already on the ADP $\ell \xhookrightarrow{m} \{{}^{p_1}r_1, \ldots, {}^{p_k}r_k\}$ via narrowing. So whenever there is a $t \trianglelefteq_\sharp r_j$ and a non-variable position $\tau$ in $t$ such that $t|_\tau$ unifies with the left-hand side $\ell'$ of some (variable-renamed) ADP $\ell' \xhookrightarrow{m'} \{{}^{p'_1}r'_1, \ldots, {}^{p'_{k'}}r'_{k'}\} \in \mathcal{A}$ using an mgu $\delta$ such that $\ell\delta, \ell'\delta \in \mathtt{ANF}_{\mathcal{A}}$, then $\delta$ is a *narrowing substitution* of $t$. Here, we only consider narrowing substitutions up to variable renaming, so every term has only finitely many narrowing substitutions. This is analogous to the narrowing substitutions defined in [NEG13] for DTs.

As shown in Section 7.6.4, in the probabilistic setting, the narrowing processor can only be used in a weaker version. Hence, it was renamed there to the *rule overlap instantiation processor*. While we can apply the narrowing substitution to the ADP, we cannot perform any rewrite steps. To be precise, if $\delta_1, \ldots, \delta_d$ are all narrowing substitutions of $t$, then we can replace $\ell \xhookrightarrow{m} \{{}^{p_1}r_1, \ldots, {}^{p_k}r_k\}$ by $\ell\delta_e \xhookrightarrow{m} \{{}^{p_1}r_1\delta_e, \ldots, {}^{p_k}r_k\delta_e\}$, for all $1 \le e \le d$, but we cannot perform any rewrite steps on $t\delta_e$ directly.

Moreover, there could be another subterm $t' \trianglelefteq_\sharp r_j$ (with $t' \neq t$) which was involved in a chain tree (i.e., $t'^\sharp\sigma \xrightarrow{\mathtt{i}}{}^*_{\mathrm{np}(\mathcal{A})} \tilde{\ell}\tilde{\sigma}$ for some substitutions $\sigma, \tilde{\sigma}$ and a left-hand side $\tilde{\ell}$ of an ADP), but this reduction is no longer possible when replacing $t'$ by the instantiations $t'\delta_1, \ldots, t'\delta_d$. We say that $t'$ is *covered* by $\delta_1, \ldots, \delta_d$ if for each narrowing substitution $\rho$ of $t'$, there is a $\delta_e$ with $1 \le e \le d$ such that $\delta_e$ is more general than $\rho$ on the variables of $\ell$, i.e., $x\rho = x\delta_e\rho'$ for all $x \in \mathcal{V}(\ell)$ and some substitution $\rho'$. So the rule overlap instantiation processor has to add another ADP $\ell \xhookrightarrow{m} \{{}^{p_1}\sharp_{\mathrm{capt}_1(\delta_1,\ldots,\delta_d)}(r_1), \ldots, {}^{p_k}\sharp_{\mathrm{capt}_k(\delta_1,\ldots,\delta_d)}(r_k)\}$, where $\mathrm{capt}_i(\delta_1, \ldots, \delta_d)$ contains all positions of subterms $t' \trianglelefteq_\sharp r_i$ which are not covered by the narrowing substitutions

$\delta_1, \dots, \delta_d$ of $t$.

Compared to Section 7.6.4, the rule overlap instantiation processor works in the same way for SAST as for innermost AST (Theorem 7.6.19). We only have to make sure that the instantiation is performed on both $\mathcal{A}$ and $\mathcal{S}$ if $\alpha \in \mathcal{S}$.

**Theorem 8.4.1** – Rule Overlap Instantiation Processor.
Let $\langle \mathcal{A}, \mathcal{S} \rangle$ be an ADP problem with $\mathcal{A} = \mathcal{A}' \uplus \{\alpha\}$ for $\alpha = \ell \xhookrightarrow{m} \{{}^{p_1}r_1, \dots, {}^{p_k}r_k\}$, let $1 \leq j \leq k$, and let $t \trianglelefteq_\sharp r_j$. Let $\delta_1, \dots, \delta_d$ be all narrowing substitutions of $t$ (up to variable renaming, where $d \geq 0$). Then $\mathrm{Proc}_{\mathtt{ROI}}(\langle \mathcal{A}, \mathcal{S} \rangle) = (\mathbf{Pol}_0, \{\langle \mathcal{A}' \cup M, \widetilde{\mathcal{S}} \rangle\})$ is sound, where

$$M = \{\ell\delta_e \xhookrightarrow{m} \{{}^{p_1}r_1\delta_e, \dots, {}^{p_k}r_k\delta_e\} \mid 1 \leq e \leq d\} \cup \{\ell \xhookrightarrow{m} \{{}^{p_1}\sharp_{\mathrm{capt}_1(\delta_1,\dots,\delta_d)}(r_1), \dots, {}^{p_k}\sharp_{\mathrm{capt}_k(\delta_1,\dots,\delta_d)}(r_k)\}\},$$

$$\widetilde{\mathcal{S}} = \begin{cases} (\mathcal{S} \setminus \{\alpha\}) \cup M, & \text{if } \alpha \in \mathcal{S}, \\ \mathcal{S}, & \text{otherwise.} \end{cases}$$

*Proof.* We first show that (8.11) holds, i.e., we show $\iota_{\langle \mathcal{A}, \mathcal{S} \rangle} \sqsubseteq \iota_{\langle \mathcal{A}' \cup M, \widetilde{\mathcal{S}} \rangle}$. Every $\mathcal{A}$-CT $\mathfrak{T}$ gives rise to an $(\mathcal{A}' \cup M)$-CT $\mathfrak{T}'$, since every rewrite step with $\alpha$ can also be done with an ADP from $M$. Moreover, if $\alpha \in \mathcal{S}$, then $M \subseteq \widetilde{\mathcal{S}}$, so every rewrite step that counts for the expected derivation length of $\mathfrak{T}$ w.r.t. $\langle \mathcal{A}, \mathcal{S} \rangle$ still counts for the expected derivation length of $\mathfrak{T}'$ w.r.t. $\langle \mathcal{A}' \cup M, \widetilde{\mathcal{S}} \rangle$. To be precise, we get $\mathrm{edl}_{\langle \mathcal{A}, \mathcal{S} \rangle}(\mathfrak{T}) \leq \mathrm{edl}_{\langle \mathcal{A}' \cup M, \widetilde{\mathcal{S}} \rangle}(\mathfrak{T}')$. (Note that we only have "$\leq$" and not "$=$" since we may add ADPs to $\widetilde{\mathcal{S}}$ that did not count before, but finding a tree of equal or greater expected derivation length suffices.) As this holds for every $\mathcal{A}$-CT, we obtain $\iota_{\langle \mathcal{A}, \mathcal{S} \rangle} \sqsubseteq \iota_{\langle \mathcal{A}' \cup M, \widetilde{\mathcal{S}} \rangle}$, and (8.11) holds.

Now we explain the precise construction. Let $\mathfrak{T} = (N, E)$ be an $\mathcal{A}$-CT and let $\overline{\mathcal{A}'} = \mathcal{A}' \cup M$. Since every left-hand side of an ADP in $\overline{\mathcal{A}'}$ is an instance of a left-hand side of an ADP in $\mathcal{A}$ and vice versa, we have $\mathtt{NF}_{\mathcal{A}} = \mathtt{NF}_{\overline{\mathcal{A}'}}$ and $\mathtt{ANF}_{\mathcal{A}} = \mathtt{ANF}_{\overline{\mathcal{A}'}}$. We will create an $\overline{\mathcal{A}'}$-CT $\mathfrak{T}' = (N, E)$. As mentioned, the core idea of this construction is that every rewrite step with $\alpha$ can also be done with an ADP from $M$. If we use $\ell \xhookrightarrow{m} \{{}^{p_1}\sharp_{\mathrm{capt}_1(\delta_1,\dots,\delta_d)}(r_1), \dots, {}^{p_k}\sharp_{\mathrm{capt}_k(\delta_1,\dots,\delta_d)}(r_k)\} \in M$, we may create fewer annotations than we did when using the old ADP $\alpha$. However, we will show that these missing annotations are always at the roots of normal forms, so they can never be used for a later rewrite step at an annotated position. We construct the terms $t'_x$ for the nodes of the $\overline{\mathcal{A}'}$-CT $\mathfrak{T}'$ inductively such that for all nodes $x \in N \setminus \mathrm{Leaf}$ with $xE = \{y_1, \dots, y_h\}$ we have $t'_x \xhookrightarrow{\mathtt{i}}_{\overline{\mathcal{A}'}} \{{}^{p_{y_1}/p_x}t'_{y_1}, \dots, {}^{p_{y_h}/p_x}t'_{y_h}\}$. Let $X \subseteq N$ be the set of nodes $x$ where we have already defined the corresponding term $t'_x$. During our construction, we ensure that the following property holds for all $x \in X$:

$$\flat(t_x) = \flat(t'_x) \wedge \mathrm{pos}_{\mathcal{D}^\sharp}(t_x) \cap \mathrm{pos}_{\mathcal{D} \wedge \neg \mathtt{NF}_{\mathcal{A}}}(\flat(t_x)) \subseteq \mathrm{pos}_{\mathcal{D}^\sharp}(t'_x). \tag{8.37}$$

Thus, all annotations of root symbols of subterms that are not in normal form in $t_x$ are still annotated in $t'_x$. In particular, every **at**- or **af**-step in $\mathfrak{T}$ is also an **at**- or **af**-step in $\mathfrak{T}'$, since its redex is not in normal form.

For the construction, we start with the same term at the root. Here, (8.37) obviously holds. As long as there is still an inner node $x \in X$ such that its successors are not contained in $X$, we do the following. Let $xE = \{y_1, \dots, y_h\}$ be the set of its successors. We need to define the corresponding terms $t'_{y_1}, \dots, t'_{y_h}$ for the nodes $y_1, \dots, y_h$. Since

*Proof*

$x$ is not a leaf and $\mathfrak{T}$ is an $\mathcal{A}$-CT, we have $t_x \overset{\mathtt{i}}{\hookrightarrow}_{\mathcal{A}} \{{}^{p_{y_1}/p_x}t_{y_1}, \ldots, {}^{p_{y_h}/p_x}t_{y_h}\}$. We have the following three cases.

(A) If it is a step with $\overset{\mathtt{i}}{\hookrightarrow}_{\mathcal{A}}$ using an ADP that is different from $\alpha$ in $\mathfrak{T}$, then we perform a rewrite step with the same ADP, the same redex, and the same substitution in $\mathfrak{T}'$. Then, it is easy to see that (8.37) holds for the resulting terms.

(B) If it is a step with $\overset{\mathtt{i}}{\hookrightarrow}_{\mathcal{A}}$ using $\alpha$ at a position $\pi \notin \mathrm{pos}_{\mathcal{D}^\sharp}(t_x)$ in $\mathfrak{T}$, then we perform a rewrite step with the new ADP $\ell \overset{m}{\hookrightarrow} \{{}^{p_1}\sharp_{\mathrm{capt}_1(\delta_1,\ldots,\delta_d)}(r_1), \ldots, {}^{p_k}\sharp_{\mathrm{capt}_k(\delta_1,\ldots,\delta_d)}(r_k)\}$, the same redex, same substitution, and same position in $\mathfrak{T}'$. Since the new ADP has the same underlying terms as $\alpha$, it is easy to see that (8.37) holds for the resulting terms. Note that the ADP that we use contains fewer annotations than the original ADP, but since $\pi \notin \mathrm{pos}_{\mathcal{D}^\sharp}(t_x)$, the step in $\mathfrak{T}$ does not introduce any annotations of the right-hand side anyway.

(C) If it is a step with $\overset{\mathtt{i}}{\hookrightarrow}_{\mathcal{A}}$ using $\alpha$ at a position $\pi \in \mathrm{pos}_{\mathcal{D}^\sharp}(t_x)$ in $\mathfrak{T}$, then we perform a rewrite step with an ADP from $M$ at the same position in $\mathfrak{T}'$, where the choice of this ADP only depends on the substitution of the step in $\mathfrak{T}$.

It remains to consider Case (C) in detail. Here, we have $t_x \overset{\mathtt{i}}{\hookrightarrow}_{\mathcal{A}} \{{}^{p_{y_1}/p_x}t_{y_1}, \ldots, {}^{p_{y_k}/p_x}t_{y_k}\}$ using the ADP $\alpha$, the position $\pi \in \mathrm{pos}_{\mathcal{D}^\sharp}(t_x)$, and a substitution $\sigma$ such that $\flat(t_x|_\pi) = \ell\sigma \in \mathtt{ANF}_{\mathcal{A}}$. Since $t_x|_\pi$ is not in normal form, we have $\pi \in \mathrm{pos}_{\mathcal{D}^\sharp}(t_x) \cap \mathrm{pos}_{\mathcal{D}\wedge\neg\mathtt{NF}_{\mathcal{A}}}(\flat(t_x)) \subseteq \mathrm{pos}_{\mathcal{D}^\sharp}(t'_x)$ by (8.37).

(C-i) If there are a $1 \leq e \leq d$ and a substitution $\gamma$ such that $x\sigma = x\delta_e\gamma$ for all $x \in \mathcal{V}(\ell)$, then we use the ADP $\ell\delta_e \overset{m}{\hookrightarrow} \{{}^{p_1}r_1\delta_e, \ldots, {}^{p_k}r_k\delta_e\} \in M$ with the position $\pi$ and the substitution $\gamma$. We have $\flat(t'_x|_\pi) = \flat(t_x|_\pi) = \ell\delta_e\gamma = \ell\sigma \in \mathtt{ANF}_{\overline{\mathcal{A}'}}$, and since $\mathcal{V}(r_j) \subseteq \mathcal{V}(\ell)$, we have $r_j\delta_e\gamma = r_j\sigma$ for all $1 \leq j \leq k$. So the step in $\mathfrak{T}'$ introduces the same annotations as the step in $\mathfrak{T}$, and (8.37) is satisfied for the resulting terms.

(C-ii) Otherwise, we use the ADP $\ell \overset{m}{\hookrightarrow} \{{}^{p_1}\sharp_{\mathrm{capt}_1(\delta_1,\ldots,\delta_d)}(r_1), \ldots, {}^{p_k}\sharp_{\mathrm{capt}_k(\delta_1,\ldots,\delta_d)}(r_k)\}$ with the position $\pi$ and the substitution $\sigma$. Compared to the step in $\mathfrak{T}$, the annotations of the resulting terms only differ at the positions $\pi.\rho$ of subterms $t' \trianglelefteq^{\rho}_{\sharp} r_j$ that are covered by $\delta_1, \ldots, \delta_d$. For every such $t'$, we show $t'\sigma \in \mathtt{NF}_{\mathcal{A}}$, which implies that (8.37) is satisfied for the resulting terms. Assume that $t'\sigma \notin \mathtt{NF}_{\mathcal{A}}$. Since $\ell\sigma \in \mathtt{ANF}_{\mathcal{A}}$ and $\ell \notin \mathcal{V}$, we have $x\sigma \in \mathtt{NF}_{\mathcal{A}}$ for all $x \in \mathcal{V}(\ell)$. Hence, $t'\sigma$ contains a redex whose proper subterms are normal forms at a non-variable position $q$ of $t'$, i.e., $t'|_q\sigma = \ell'\sigma' \in \mathtt{ANF}_{\mathcal{A}}$ for the (variable-renamed) left-hand side $\ell'$ of some ADP of $\mathcal{A}$ and some substitution $\sigma'$. So $t'|_q$ and $\ell'$ are unifiable with some mgu $\theta$, and there is a substitution $\gamma'$ with $x\sigma = x\theta\gamma'$ for all $x \in \mathcal{V}(\ell)$ and $x\sigma' = x\theta\gamma'$ for all $x \in \mathcal{V}(\ell')$ (the variables of $\ell$ that do not occur in $t'|_q$ are neither changed by $\theta$ nor do they occur in its range). As $\ell\theta\gamma' = \ell\sigma$ and $\ell'\theta\gamma' = \ell'\sigma'$ are in $\mathtt{ANF}_{\mathcal{A}}$, we also have $\ell\theta, \ell'\theta \in \mathtt{ANF}_{\mathcal{A}}$, since every redex in a proper subterm of $\ell\theta$ or $\ell'\theta$ would remain a redex after applying $\gamma'$. Thus, $\theta$ is a narrowing substitution of $t'$. Since $t'$ is covered by $\delta_1, \ldots, \delta_d$, there are a $1 \leq e \leq d$ and a substitution $\rho'$ with $x\theta = x\delta_e\rho'$ for all $x \in \mathcal{V}(\ell)$, and hence, $x\sigma = x\delta_e\rho'\gamma'$ for all $x \in \mathcal{V}(\ell)$. But then Case (C-i) applies, which is a contradiction.

Now we show condition (8.12). Let $\mathfrak{P}$ be a well-formed proof tree with $L_{\mathcal{A}}(v) = \langle \mathcal{A}, \mathcal{S} \rangle$, let $v_1, \ldots, v_k = v$ be the path from the root node $v_1$ to $v$, and let $c_{(v_1,\ldots,v_k)} = L_{\mathcal{C}}(v_1) \sqcup \ldots \sqcup L_{\mathcal{C}}(v_{k-1})$. We have $\iota_{\langle \mathcal{A}' \cup M, (\mathcal{A}' \cup M) \setminus \widetilde{\mathcal{S}} \rangle} \sqsubseteq c_{(v_1,\ldots,v_k)}$, because every $(\mathcal{A}' \cup M)$-CT $\mathfrak{T}$ gives rise to an $\mathcal{A}$-CT $\mathfrak{T}'$ with at least the same **at**- and **af**-steps. We can replace each usage of an ADP $\ell\delta_e \xhookrightarrow{m} \{{}^{p_1}r_1\delta_e, \ldots, {}^{p_k}r_k\delta_e\}$ with the more general ADP $\alpha$, and each ADP $\ell \xhookrightarrow{m} \{{}^{p_1}\sharp_{\mathrm{capt}_1(\delta_1,\ldots,\delta_d)}(r_1), \ldots, {}^{p_k}\sharp_{\mathrm{capt}_k(\delta_1,\ldots,\delta_d)}(r_k)\}$ can be replaced by $\alpha$ as well, leading to more annotations than before.
If an **at**- or **af**-step is performed with an ADP $\beta \in (\mathcal{A}' \cup M) \setminus \widetilde{\mathcal{S}}$ in $\mathfrak{T}$, we have the following cases.

- If $\beta \in \mathcal{A}' \setminus \widetilde{\mathcal{S}}$, and $\alpha \in \mathcal{S}$, then
$$\beta \in \mathcal{A}' \setminus \widetilde{\mathcal{S}} = \mathcal{A}' \setminus ((\mathcal{S} \setminus \{\alpha\}) \cup M) = \mathcal{A}' \setminus (\mathcal{S} \cup M) \subseteq \mathcal{A} \setminus \mathcal{S}.$$
- If $\beta \in \mathcal{A}' \setminus \widetilde{\mathcal{S}}$, and $\alpha \notin \mathcal{S}$, then
$$\beta \in \mathcal{A}' \setminus \widetilde{\mathcal{S}} = \mathcal{A}' \setminus \mathcal{S} \subseteq \mathcal{A} \setminus \mathcal{S}.$$
- If $\beta \in M \setminus \widetilde{\mathcal{S}}$, then $\alpha \notin \mathcal{S}$, and we use $\alpha$ in $\mathfrak{T}'$.

In every case, if the step counts towards the expected derivation length w.r.t. $(\mathcal{A}' \cup M) \setminus \widetilde{\mathcal{S}}$ in $\mathfrak{T}$, it will count for the expected derivation length w.r.t. $\mathcal{A} \setminus \mathcal{S}$ in $\mathfrak{T}'$ as well. Therefore, we obtain $\mathrm{edl}_{\langle \mathcal{A}' \cup M, (\mathcal{A}' \cup M) \setminus \widetilde{\mathcal{S}} \rangle}(\mathfrak{T}) \leq \mathrm{edl}_{\langle \mathcal{A}, \mathcal{A} \setminus \mathcal{S} \rangle}(\mathfrak{T}')$. As this holds for every $(\mathcal{A}' \cup M)$-CT, we get $\iota_{\langle \mathcal{A}' \cup M, (\mathcal{A}' \cup M) \setminus \widetilde{\mathcal{S}} \rangle} \sqsubseteq \iota_{\langle \mathcal{A}, \mathcal{A} \setminus \mathcal{S} \rangle} \sqsubseteq c_{(v_1,\ldots,v_k)}$, by well-formedness of $\mathfrak{P}$, and (8.12) holds. *QED*

We illustrate the rule overlap instantiation processor for complexity with the following example.

Example 8.4.2 (Rule Overlap Instantiation Processor). Consider $\mathcal{P}_{\mathsf{roi}}$ from Example 7.6.20.

$$\mathsf{f}(\mathsf{d}(x)) \to \{{}^{3/4}\mathsf{e}(\mathsf{f}(\mathsf{g}(x)), \mathsf{f}(\mathsf{h}(x))), {}^{1/4}\mathsf{a}\} \qquad \begin{array}{l}\mathsf{g}(\mathsf{a}) \to \{{}^{1}\mathsf{d}(\mathsf{a})\}\\ \mathsf{h}(\mathsf{b}) \to \{{}^{1}\mathsf{d}(\mathsf{b})\}.\end{array}$$

$\mathcal{P}_{\mathsf{roi}}$ has constant runtime complexity, i.e., $\iota_{\mathcal{P}_{\mathsf{roi}}} = \mathbf{Pol}_0$, because for every instantiation, at most one of the two recursive $\mathsf{f}$-calls in the right-hand side of the $\mathsf{f}$-rule can be evaluated. The reason is that we can either use the $\mathsf{g}$-rule if $x$ is instantiated with $\mathsf{a}$, or we can apply the $\mathsf{h}$-rule if $x$ is instantiated with $\mathsf{b}$, but not both.

With the rule overlap instantiation processor, our ADP framework can determine this constant complexity automatically. Using the dependency graph processor on the canonical ADP problem $\langle \mathcal{DP}(\mathcal{P}_{\mathsf{roi}}), \mathcal{DP}(\mathcal{P}_{\mathsf{roi}}) \rangle$ yields $\langle \mathcal{A}_{\mathsf{roi}}, \mathcal{A}_{\mathsf{roi}} \rangle$, where $\mathcal{A}_{\mathsf{roi}}$ consists of

$$\mathsf{f}(\mathsf{d}(x)) \xhookrightarrow{\mathsf{t}} \{{}^{3/4}\mathsf{e}(\mathsf{f}^{\sharp}(\mathsf{g}(x)), \mathsf{f}^{\sharp}(\mathsf{h}(x))), {}^{1/4}\mathsf{a}\} \qquad \begin{array}{l}\mathsf{g}(\mathsf{a}) \xhookrightarrow{\mathsf{t}} \{{}^{1}\mathsf{d}(\mathsf{a})\}\\ \mathsf{h}(\mathsf{b}) \xhookrightarrow{\mathsf{t}} \{{}^{1}\mathsf{d}(\mathsf{b})\}.\end{array}$$

We apply $\mathrm{Proc}_{\mathtt{ROI}}$ using the term $t = \mathsf{f}(\mathsf{g}(x))$ whose only narrowing substitution is $\delta = \{x/\mathsf{a}\}$. For the other subterm $\mathsf{f}^{\sharp}(\mathsf{h}(x))$ with annotated root, $\mathsf{f}(\mathsf{h}(x))$ is not covered by $\delta$. Hence, we generate an additional ADP where this second subterm is annotated.

Table 8.1: Overview of the processors in the probabilistic ADP framework for SAST.

| Processor | Definition of Proc | Explanation |
|---|---|---|
| Chain Criterion (Section 8.1) | $\mathcal{P} \mapsto \langle \mathcal{DP}(\mathcal{P}), \mathcal{DP}(\mathcal{P}) \rangle$ | Reduces bounding the expected runtime of a PTRS to bounding the expected runtime of its canonical ADP problem, with a tight bound ($\iota_{\mathcal{P}} = \iota_{\langle \mathcal{DP}(\mathcal{P}), \mathcal{DP}(\mathcal{P}) \rangle}$). |
| Usable Rules Processor (Section 8.2.1) | $\mathrm{Proc}_{\mathtt{UR}}(\langle \mathcal{A}, \mathcal{S} \rangle)$ $= (\mathbf{Pol}_0, \{\langle \mathcal{A}', \mathcal{S}' \rangle\})$ | Detects rules that are irrelevant below annotations and changes their flag to f. |
| Dependency Graph Processor (Section 8.2.2) | $\mathrm{Proc}_{\mathtt{DG}}(\langle \mathcal{A}, \mathcal{S} \rangle)$ $= (\mathbf{Pol}_0, \{\langle \mathcal{A}\vert_{\mathcal{J}}, \mathcal{S}\vert_{\mathcal{J}} \rangle \mid$ $\mathcal{J}$ is an SCC-prefix of $\mathcal{A}\})$ | Decomposes the problem into SCC-prefix-based sub-problems and restricts annotations to the chosen SCC-prefix (it returns $\{\langle \mathcal{A}, \varnothing \rangle\}$ if there is no SCC-prefix). |
| Knowledge Propagation Processor (Section 8.2.3) | $\mathrm{Proc}_{\mathtt{KP}}(\langle \mathcal{A}, \mathcal{S} \rangle)$ $= (\mathbf{Pol}_0, \{\langle \mathcal{A}, \mathcal{S} \setminus \{\alpha\} \rangle\})$ | Removes $\alpha$ from $\mathcal{S}$ if all predecessors of $\alpha$ in the dependency graph are already outside $\mathcal{S}$, exploiting complexity knowledge in the proof tree. |
| Reduction Pair Processor (Section 8.3.1) | $\mathrm{Proc}_{\mathtt{RP}}(\langle \mathcal{A}, \mathcal{S} \rangle)$ $= (c, \{\langle \mathcal{A}, \mathcal{S} \setminus \mathcal{A}_{\succ} \rangle\})$ | Removes the strictly decreasing ADPs from $\mathcal{S}$ using a polynomial interpretation, and derives a complexity bound $c$. |
| Rule Overlap Instantiation Processor (Section 8.4) | $\mathrm{Proc}_{\mathtt{ROI}}(\langle \mathcal{A}, \mathcal{S} \rangle)$ $= (\mathbf{Pol}_0, \{\langle \mathcal{A}' \cup M, \widetilde{\mathcal{S}} \rangle\})$ | Instantiates one ADP by all narrowing substitutions caused by rule overlaps, but unlike classical narrowing it does not already execute the rewrite step. |

Thus, we replace the former f-ADP by the following two new ADPs.

$$\mathsf{f}(\mathsf{d}(\mathsf{a})) \xhookrightarrow{\mathsf{t}} \{{}^{3}\!/_{4}\mathsf{e}(\,\mathsf{f}^{\sharp}(\mathsf{g}(\mathsf{a})),\, \mathsf{f}^{\sharp}(\mathsf{h}(\mathsf{a}))\,),\ {}^{1}\!/_{4}\mathsf{a}\}$$
$$\mathsf{f}(\mathsf{d}(x)) \xhookrightarrow{\mathsf{t}} \{{}^{3}\!/_{4}\mathsf{e}(\,\mathsf{f}(\mathsf{g}(x)),\, \mathsf{f}^{\sharp}(\mathsf{h}(x))\,),\ {}^{1}\!/_{4}\mathsf{a}\}.$$

Now one can remove the annotation of $\mathsf{f}^{\sharp}(\mathsf{h}(\mathsf{a}))$ from the first ADP by the dependency graph processor, since the normal form $\mathsf{h}(\mathsf{a})$ can never be rewritten to a term of the form $\mathsf{d}(\ldots)$. Then we apply the reduction pair processor with the interpretation that maps $\mathsf{f}^{\sharp}_{\mathrm{Pol}}(x) = \mathsf{g}^{\sharp}_{\mathrm{Pol}}(x) = \mathsf{h}^{\sharp}_{\mathrm{Pol}}(x) = 1$ and $f_{\mathrm{Pol}} = 0$ for all other symbols $f$. Then all ADPs are strictly decreasing with $\epsilon = 1/4$, so they are all removed from $\mathcal{S}$ and the resulting ADP problems are solved. Since this interpretation is a CPI that maps all annotated symbols to constants, the derived complexity is $\mathbf{Pol}_0$, which proves $\iota_{\langle \mathcal{A}_{\mathsf{roi}}, \mathcal{A}_{\mathsf{roi}} \rangle} = \iota_{\mathcal{P}_{\mathsf{roi}}} = \mathbf{Pol}_0$. Using an interpretation that maps all annotated symbols to constants would not be possible without the rule overlap instantiation processor.

SECTION 8.5

## All Processors at One Glance

Table 8.1 shows all processors of the ADP framework for SAST that we presented in this chapter at one glance. For each processor, we give its name, its formal definition Proc, and a short explanation of its underlying idea.

SECTION 8.6

## Evaluation

As in Section 4.5, we implemented all techniques of this chapter in AProVE and applied them to all 199 PTRSs from the *Termination Problem Database* [TPDB]. All experiments were performed on a machine with 16 GB of RAM and a timeout of 30 seconds for each example.

Since a *finite* bound on the expected innermost runtime complexity proves SAST of innermost evaluation w.r.t. basic start terms (see Corollary 8.1.19 and the remark after it, as well as Chapter 4), we evaluate this property directly through the expected complexity ADP framework of this chapter: AProVE analyzes the expected (innermost, basic) runtime complexity, and every finite upper bound counts as a proof of SAST w.r.t. basic start terms. As in Chapter 7, proofs are obtained *exclusively* via the ADP framework of this chapter, without any direct interpretations. Recall that the framework is only applicable to innermost evaluation with basic start terms. In this setting, AProVE derives a finite expected complexity bound, i.e., proves SAST w.r.t. basic start terms, for 66 PTRSs ($66/199 \approx 33.2\%$).

To assess the contribution of each processor, we also ran the experiment with each processor of the expected-complexity ADP framework disabled in turn (*leave-one-out*). This also includes the probability removal processor mentioned at the beginning of Section 8.2. Table 8.2 lists, for each configuration, the full distribution of the derived upper bounds together with the resulting number of finite bounds, i.e., of proofs of SAST w.r.t. basic start terms.

| Disabled processor | $\mathbf{Pol}_0$ | $\mathbf{Pol}_1$ | $\mathbf{Pol}_2$ | **Exp** | **2-Exp** | finite bound |
|---|---|---|---|---|---|---|
| – (full framework) | 33 | 21 | 7 | 5 | 0 | 66 (33.2%) |
| reduction pair | 2 | 1 | 1 | 1 | 0 | 5 (2.5%) |
| usable rules | 28 | 22 | 7 | 4 | 0 | 61 (30.7%) |
| knowledge propagation | 31 | 21 | 7 | 5 | 0 | 64 (32.2%) |
| probability removal | 33 | 21 | 6 | 6 | 0 | 66 (33.2%) |
| dependency graph | 28 | 22 | 7 | 6 | 0 | 63 (31.7%) |

Table 8.2: Evaluation of the processors of the ADP framework for expected complexity, where the last column counts the finite bounds, i.e., the proofs of SAST of innermost evaluation w.r.t. basic start terms.

As for AST, the reduction pair processor is by far the most important technique: without it, only 5 of the 66 proofs remain, since it is the only processor that can orient and remove annotated dependency pairs via interpretations and thereby derive a complexity bound. The usable rules processor accounts for five further proofs, the knowledge propagation processor for two, and the dependency graph processor for three further proofs. Disabling the probability removal processor does not change the number of proofs: one proof is lost, but for another PTRS, where the full framework runs into the timeout, AProVE then finds an exponential bound. The bound distribution reveals a further, subtler effect: several processors also *improve the quality* of the bounds. Disabling the usable rules processor turns constant bounds into linear ones (the $\mathbf{Pol}_0$ count drops from 33 to 28 while $\mathbf{Pol}_1$ rises from 21 to 22), and disabling the dependency graph processor shows the same effect and additionally yields the exponential bound

for the PTRS mentioned above (the **Exp** count rises from 5 to 6). As before, all processors nevertheless remain useful in practice, since they are often much cheaper than the reduction pair processor, speed up the overall proof search, and tighten the resulting complexity bounds.

SECTION 8.7

## Related Work & Open Problems

The related work for dependency pairs in general has already been discussed in Section 7.9 including [NEG13] which our whole chapter is based on. Moreover, in Section 10.6 we discuss further related work regarding techniques to analyze complexity of other formalisms of probabilistic programs.

Here, we want to highlight the most important related work on analyzing expected runtimes and expected costs for other probabilistic programming formalisms. For imperative probabilistic programs, the ert-calculus [KKMO16, KKMO18] provides a weakest-precondition-style calculus to reason about expected runtimes, which was extended to induction-based proofs of lower bounds on expected runtimes in [HKGK20], and to amortized expected runtimes in [BKK+23]. Automated tools that infer bounds on expected costs or expected runtimes of imperative probabilistic programs include Absynth [NCH18], eco − imp [AMS20], which performs a modular analysis, and KoAT [MHG21] for probabilistic integer programs, developed at our chair. Moreover, exact expected runtimes can be computed for constant probability programs [GGH19], and expected runtimes of probabilistic programs with unbounded recursion are analyzed in [BKKV15]. For probabilistic functional programs, which are closest to term rewriting, there exist type-based approaches for expected cost analysis [ADLG19, WKH20] and automated amortized cost analysis for probabilistic data structures [LMZ22].

The open problems also remain the same as in Section 7.9. However, since complexity analysis is an optimization problem while termination analysis is only a decision problem, the development of an optimal strategy becomes even more intriguing and involved. In addition, computing expected runtimes exactly is even harder than deciding the corresponding termination properties [KKM19].

# 9 Disproving Termination

Almost all the techniques we have introduced so far are designed to prove termination of PTRSs, i.e., to prove AST or SAST, and upper bounds on the worst-case complexity. However, it is equally important to develop techniques to disprove termination of PTRSs, i.e., to disprove AST or SAST. While proving termination ensures the absence of bugs, disproving termination can be used to find bugs in PTRSs.

When analyzing non-termination, the goal is to find a *finite representation* of an *infinite run*. One example of such a finite representation is a *loop*. A loop in a TRS $\mathcal{R}$ is a sequence $t \to_{\mathcal{R}}^{+} C[t\sigma]$ for some term $t$, context $C$, and substitution $\sigma$. Indeed, such a loop gives rise to an infinite rewrite sequence $t \to_{\mathcal{R}}^{n} C[t\sigma] \to_{\mathcal{R}}^{n} C[C[t\sigma]\sigma] \to_{\mathcal{R}}^{n} \cdots$ for some $n \in \mathbb{N}_{>0}$. In fact, most non-termination techniques for TRSs search for such loops heuristically.

The goal of the AST and SAST disproving analysis remains the same: find a finite structure that represents an infinite execution that terminates with probability $< 1$ or that has an infinite expected derivation length, respectively. Note that we actually disprove PAST instead of SAST, because we try to find a single reduction tree with infinite expected derivation length. So while for proving we always tried to find a finite bound on the expected derivation length, disproving focuses on single reduction trees. Obviously, a proof for non-PAST yields a proof for non-SAST, as SAST implies PAST.

We show in Counterexample 9.1.1 that searching for the existence of loops does not suffice in the probabilistic setting. Instead, we introduce a general approach to disprove termination of PTRSs by embedding random walks into rewrite sequences of PTRSs (Theorem 9.1.8). Thus, our finite representations of infinite runs are random walks.

First, we embed random walks via *occurrences* of *loops* (Theorems 9.2.6, 9.2.11 and 9.2.15), which were previously used to disprove termination of TRSs, see, e.g., [GTS05]. Second, we embed random walks via looping *pattern terms* (Theorems 9.3.16, 9.3.30 and 9.3.35), which were previously used to prove non-looping non-termination of TRSs [EEG12]. While the original techniques are *qualitative* (*find the existence of a loop*), the probabilistic setting requires *quantitative* analyses (*find the number of possible loops*). We develop a dynamic programming algorithm (Algorithm 8) to solve the counting problems that arise in this setting, e.g., to determine how many instances of a term $t$ occur in a term $s$ at pairwise non-overlapping positions. These counting problems can be seen as a specific case of the vertex-disjoint graph-packing problem.

### Contributions

We present the following contributions in this chapter. The results marked with !⋆! are novel, while the remaining contributions can be found in the following peer-reviewed publication: ⋆ [KNSG26].

- ⋆ The general proof idea to embed random walks into RTs to disprove `AST` or to disprove `SAST` (Theorem 9.1.8).
- ⋆ A framework to count term or pattern occurrences using a dynamic programming algorithm (Algorithm 8) for computing the maximal number of non-overlapping occurrences (Definition 9.2.2).
- ⋆ Embedding random walks based on non-overlapping linear term occurrences (Theorem 9.2.6).
- ⋆ Embedding random walks based on non-overlapping term occurrences in non-variable-decreasing trees (Theorem 9.2.11).
- ⋆ Embedding random walks based on orthogonal term occurrences (Theorem 9.2.15).
- !⋆! Embedding random walks based on orthogonal pattern occurrences (Theorem 9.3.16), using a weaker requirement compared to [KNSG26].
- !⋆! Embedding random walks based on non-overlapping linear pattern occurrences (Theorem 9.3.30).
- !⋆! Embedding random walks based on non-overlapping pattern occurrences in non-variable-decreasing pattern trees (Theorem 9.3.35).

## Structure

We present the core principle behind embedding RTs with a known termination probability below 1 into a $\to_{\mathcal{P}}$-RT to disprove `AST` of $\mathcal{P}$ in Section 9.1. Here, we also show that the same idea can be used to disprove `PAST` (and thus `SAST`): we embed an RT with an infinite expected derivation length into a $\to_{\mathcal{P}}$-RT. Then, we explain how to automatically find the $\to_{\mathcal{P}}$-RTs and the random walks to disprove `AST` or `PAST`. More precisely, in Section 9.2 we show how to embed random walks based on loops and develop a dynamic programming algorithm to solve the arising counting problems for occurrences of terms. Finally, in Section 9.3, we show how to embed random walks based on looping pattern terms. We end this chapter by performing an experimental evaluation in Section 9.4, and discussing related work and giving important open problems for future work in Section 9.5.

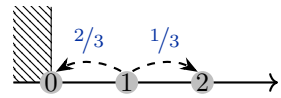
SECTION 9.1

## Disproving `AST` and `PAST` via Loops and Embeddings

The most common approach to disprove termination of a TRS $\mathcal{R}$ is by finding *loops*. A loop is a sequence $t \to_{\mathcal{R}}^{+} C[t\sigma]$ for some term $t$, context $C$, and substitution $\sigma$. Indeed, such a loop gives rise to an infinite rewrite sequence $t \to_{\mathcal{R}}^{n} C[t\sigma] \to_{\mathcal{R}}^{n} C[C[t\sigma]\sigma] \to_{\mathcal{R}}^{n} \cdots$

for some $n \in \mathbb{N}_{>0}$. However, to disprove AST or PAST of a PTRS $\mathcal{P}$, it is not sufficient to find a loop in the *non-probabilistic variant* $\mathrm{np}(\mathcal{P}) = \{\ell \to r \mid \ell \to \mu \in \mathcal{P}, r \in \mathrm{Supp}(\mu)\}$.

Counterexample 9.1.1. Consider the PTRS $\mathcal{P}_{13}$ with the rule $\mathtt{g}(x) \to \{{}^{2/3}x, {}^{1/3}\mathtt{g}(\mathtt{g}(x))\}$ modeling a negatively drifting random walk on the number of g's, see Figure 9.1. Here, $\mathrm{np}(\mathcal{P}_{13})$ contains the two rules $\mathtt{g}(x) \to x$ and $\mathtt{g}(x) \to \mathtt{g}(\mathtt{g}(x))$. The latter gives rise to the loop $\mathtt{g}(x) \to C[\mathtt{g}(x)\sigma]$ with $C = \mathtt{g}(\Box)$ and the identity substitution $\sigma$. However, $\mathcal{P}_{13}$ is PAST, as it models a negatively drifting random walk.




Count $\|t\| = |t|_{\mathtt{g}}$

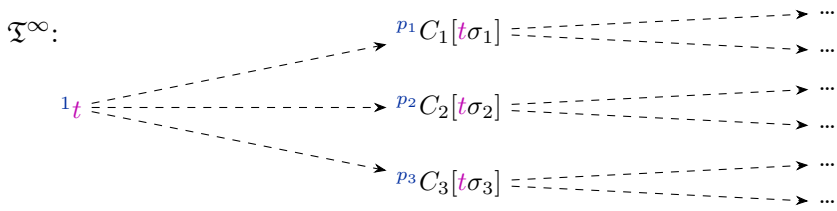
Figure 9.1: $\mathcal{P}_{13}$ as an RWP.

Instead of $\mathrm{np}(\mathcal{P})$, one has to consider $\to_{\mathcal{P}}$-RTs. If there is a $\to_{\mathcal{P}}$-RT with root $t$, at least one rewrite step, and an instance of $t$ in every leaf, then $\mathcal{P}$ is not AST and thus, also neither PAST nor SAST.

The simplest case is the one described above, where a looping term occurs in every leaf of a $\to_{\mathcal{P}}$-RT.

Theorem 9.1.2 – Disproving AST and PAST via Loops.
Let $\mathcal{P}$ be a PTRS and $\mathfrak{T}$ be a $\to_{\mathcal{P}}$-RT with $\mathrm{h}(\mathfrak{T}) > 0$ such that $\mathrm{root}(\mathfrak{T}) = t$ and for every $v \in \mathrm{Leaf}(\mathfrak{T})$ there is a context $C_v$ and a substitution $\sigma_v$ such that $t_v = C_v[t\sigma_v]$. Then neither $\mathtt{AST}(\to_{\mathcal{P}})$ nor $\mathtt{PAST}(\to_{\mathcal{P}})$ holds.

*Proof.* We construct an infinite $\to_{\mathcal{P}}$-RT $\mathfrak{T}^\infty$ based on $\mathfrak{T}$ where we always rewrite one of the occurrences of $t$ in a leaf according to the rules used to generate $\mathfrak{T}$. Since $\mathfrak{T}$ has an occurrence of $t$ in every leaf, the resulting tree $\mathfrak{T}^\infty$ has no leaves, and therefore $\mathcal{P}$ is neither AST nor PAST.




We only have to show that if we rewrite a term $s = C[t\sigma]$ for some context $C$ and some substitution $\sigma$ using the same rules as in $\mathfrak{T}$, then we obtain a tree $\mathfrak{T}_s$ where all leaves contain an occurrence of $t$ again. Let $\tau$ be the position of the hole in $C$. If we rewrite at position $\pi$ in $\mathfrak{T}$, then we rewrite at position $\tau.\pi$ in the tree $\mathfrak{T}_s$. For every leaf $v \in \mathrm{Leaf}(\mathfrak{T})$ there is a corresponding leaf $v' \in \mathrm{Leaf}(\mathfrak{T}_s)$ with $t_{v'} = C[t_v\sigma]$. Since there exists an occurrence of $t$ within $t_v$, i.e., $t_v = C'[t\sigma']$, there is an occurrence of $t$ within $t_{v'} = C[t_v\sigma] = C[C'[t\sigma']\sigma]$ as well. *QED*

Example 9.1.3. Consider $\mathcal{P}$ with the only rule $\mathtt{f}(\mathtt{g}(x)) \to \{{}^{2/3}\mathtt{g}(\mathtt{f}(\mathtt{g}(x))), {}^{1/3}\mathtt{f}(\mathtt{g}(\mathtt{f}(x)))\}$. Since the $\to_{\mathcal{P}}$-RT corresponding to the only rewrite rule (depicted in Figure 9.2) contains the looping term $\mathtt{f}(\mathtt{g}(x))$ at the root and in each leaf, we know by Theorem 9.1.2 that neither $\mathtt{AST}(\to_{\mathcal{P}})$ nor $\mathtt{PAST}(\to_{\mathcal{P}})$ holds.

However, Theorem 9.1.2 can only be used to disprove AST and PAST for a very restricted

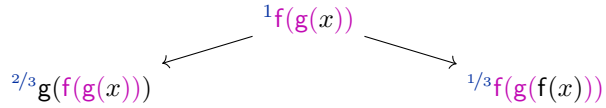



Figure 9.2: $\to_{\mathcal{P}}$-RT with looping term $\mathsf{f}(\mathsf{g}(x))$.

class of PTRSs. To handle more complicated examples, instead of requiring that a looping term $t$ occurs in every leaf of a $\to_{\mathcal{P}}$-RT, one has to find a PARS $(A, \to)$ and a $\to$-RT $\mathfrak{T}$ where it is known that $|\mathfrak{T}| < 1$ (to disprove $\mathtt{AST}$) or $\mathrm{edl}(\mathfrak{T}) = \infty$ (to disprove $\mathtt{PAST}$) holds. In addition, one has to find a $\to_{\mathcal{P}}$-RT $\mathfrak{T}'$ and an *embedding* $\mathsf{e} : N^{\mathfrak{T}} \to N^{\mathfrak{T}'}$ which ensures $|\mathfrak{T}'| \leq |\mathfrak{T}|$ and $\mathrm{edl}(\mathfrak{T}) \leq \mathrm{edl}(\mathfrak{T}')$. If $|\mathfrak{T}| < 1$, then we have disproven $\mathtt{AST}(\to_{\mathcal{P}})$, and if $\mathrm{edl}(\mathfrak{T}) = \infty$, then we have disproven $\mathtt{PAST}(\to_{\mathcal{P}})$.

Definition 9.1.4 – RT-Embedding.

*Let $i \in \{1, 2\}$, let $(A_i, \to_i)$ be a PARS, and let $\mathfrak{T}_i$ be a $\to_i$-RT. An embedding from $\mathfrak{T}_1$ to $\mathfrak{T}_2$ is an injective mapping $\mathsf{e} : N^{\mathfrak{T}_1} \to N^{\mathfrak{T}_2}$, where $N^{\mathfrak{T}_i}$ denotes the set of nodes of $\mathfrak{T}_i$, such that*

- *$p_v = p_{\mathsf{e}(v)}$ for all $v \in N^{\mathfrak{T}_1}$ (the probability of $v$ in $\mathfrak{T}_1$ and the probability of its image $\mathsf{e}(v)$ in $\mathfrak{T}_2$ are the same),*
- *if there is a path from a node $v$ to $w$ in $\mathfrak{T}_1$, then there is a path from $\mathsf{e}(v)$ to $\mathsf{e}(w)$ in $\mathfrak{T}_2$ as well, and*
- *if $w$ is a direct successor of $v$ in $\mathfrak{T}_1$, then the path from $\mathsf{e}(v)$ to $\mathsf{e}(w)$ in $\mathfrak{T}_2$ does not contain any other nodes from the image $\mathsf{e}(N^{\mathfrak{T}_1})$.*

Let us first define some additional notation and state some important observations and lemmas for embeddings on RTs.

Definition 9.1.5 – Predecessors and Successors.

*Let $T = (N, E)$ be a tree. By $\mathrm{Pre}(v)$ we denote the set of all (not necessarily direct) predecessors of a node $v \in N$ including the node $v$ itself. Similarly, let $\mathrm{Post}(v)$ be the set of all (not necessarily direct) successors of a node $v \in N$ including the node $v$ itself. By $\mathrm{Post}_{\mathrm{Leaf}}(v) \subseteq \mathrm{Post}(v)$ we denote the set of all leaves in $\mathrm{Post}(v)$.*

Lemma 9.1.6 – Node Images Have Disjoint Leaf Descendants.

*Let $(A_1, \to_1)$, $(A_2, \to_2)$ be two PARSs, and let $\mathfrak{T}_i$ be a $\to_i$-RT for $i \in \{1, 2\}$ such that there exists an embedding $\mathsf{e}$ from $\mathfrak{T}_1$ to $\mathfrak{T}_2$. Let $u_1, u_2 \in N^{\mathfrak{T}_1}$ with $u_1 \neq u_2$ be two nodes that are incomparable in $\mathfrak{T}_1$, i.e., $u_1 \notin \mathrm{Pre}(u_2)$ and $u_2 \notin \mathrm{Pre}(u_1)$. Then, the sets $\mathrm{Post}_{\mathrm{Leaf}}(\mathsf{e}(u_1))$ and $\mathrm{Post}_{\mathrm{Leaf}}(\mathsf{e}(u_2))$ are disjoint.*

An illustration of Lemma 9.1.6 is shown in Figure 9.3.

*Proof.* Assume for a contradiction that there is a leaf $w \in \mathrm{Leaf}(\mathfrak{T}_2)$ with $w \in \mathrm{Post}_{\mathrm{Leaf}}(\mathsf{e}(u_1)) \cap \mathrm{Post}_{\mathrm{Leaf}}(\mathsf{e}(u_2))$. Then both $\mathsf{e}(u_1)$ and $\mathsf{e}(u_2)$ are predecessors of $w$, and therefore, one of them is above the other, i.e., there is a path from $\mathsf{e}(u_1)$ to $\mathsf{e}(u_2)$ or from $\mathsf{e}(u_2)$ to $\mathsf{e}(u_1)$ in $\mathfrak{T}_2$.
W.l.o.g., let $\mathsf{e}(u_1)$ be above $\mathsf{e}(u_2)$. Let $v_0 \to v_1 \to \cdots \to v_n = u_2$ be the path in $\mathfrak{T}_1$ from the root to $u_2$. By path preservation of the embedding $\mathsf{e}$, for every $i < n$ there *Proof*

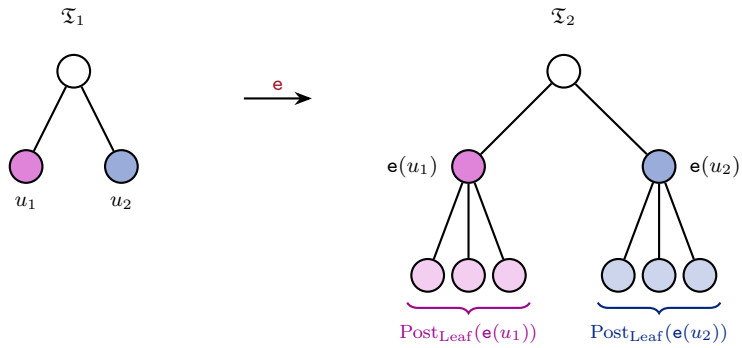



Figure 9.3: Illustration of Lemma 9.1.6 for the special case where $u_1, u_2$ are leaves.

is a path from $\mathsf{e}(v_i)$ to $\mathsf{e}(v_{i+1})$ in $\mathfrak{T}_2$, and each such path does not contain any other nodes from $\mathsf{e}(N^{\mathfrak{T}_1})$ by Definition 9.1.4.
Since $\mathsf{e}(u_1)$ is a strict predecessor of $\mathsf{e}(u_2) = \mathsf{e}(v_n)$, there is a smallest index $i \in \{0, \dots, n-1\}$ such that $\mathsf{e}(u_1)$ is a predecessor of $\mathsf{e}(v_{i+1})$. Note that $\mathsf{e}(u_1)$ is not a predecessor of $\mathsf{e}(v_i)$: for $i \geq 1$ this follows from the minimality of $i$, and for $i = 0$ it holds because $\mathsf{e}(v_0) = \mathsf{e}(\mathfrak{r})$ is a predecessor of *every* node in $\mathsf{e}(N^{\mathfrak{T}_1})$, so $\mathsf{e}(u_1)$ being a predecessor of $\mathsf{e}(v_0)$ would imply $\mathsf{e}(u_1) = \mathsf{e}(v_0)$ and hence $u_1 = \mathfrak{r} = v_0$ by injectivity, but $v_0$ is a predecessor of $v_n = u_2$, contradicting the incomparability of $u_1$ and $u_2$. Hence, $\mathsf{e}(u_1)$ is an inner node on the path from $\mathsf{e}(v_i)$ to $\mathsf{e}(v_{i+1})$. Since $u_1 \notin \mathrm{Pre}(u_2)$, the node $u_1$ does not lie on the path $v_0, \dots, v_n$ from the root to $u_2$; in particular, $u_1 \neq v_i$ and $u_1 \neq v_{i+1}$. Injectivity therefore yields $\mathsf{e}(u_1) \neq \mathsf{e}(v_i)$ and $\mathsf{e}(u_1) \neq \mathsf{e}(v_{i+1})$. This contradicts the direct-successor condition in Definition 9.1.4. *QED*

Lemma 9.1.7 – Leaves of the Target Tree Have Embedded Leaf Predecessors.

*Let $(A_1, \to_1)$, $(A_2, \to_2)$ be two PARSs, and let $\mathfrak{T}_i$ be a $\to_i$-RT for $i \in \{1, 2\}$ such that there exists an embedding $\mathsf{e}$ from $\mathfrak{T}_1$ to $\mathfrak{T}_2$. Then for every $w \in \mathrm{Leaf}(\mathfrak{T}_2)$, there is a predecessor $v \in \mathrm{Pre}(w)$ and a leaf $u \in \mathrm{Leaf}(\mathfrak{T}_1)$ such that $v = \mathsf{e}(u)$.*

An illustration of Lemma 9.1.7 is shown in Figure 9.4.

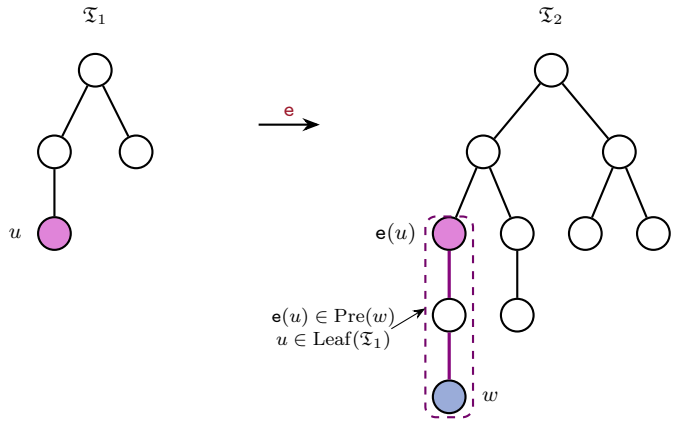



Figure 9.4: Illustration of Lemma 9.1.7.

*Proof.* We start with some basic observations. *Proof*

I. Since we have $p_v = p_{\mathsf{e}(v)}$ for all $v \in N^{\mathfrak{T}_1}$ by definition of an embedding (Definition 9.1.4) and $p_{\mathfrak{r}} = 1$ for the root node $\mathfrak{r}$ of $\mathfrak{T}_1$ by definition of a $\to$-RT, we must map the root $\mathfrak{r}$ of $\mathfrak{T}_1$ to some node $v \in N^{\mathfrak{T}_2}$ with $p_v = p_{\mathsf{e}(\mathfrak{r})} = p_{\mathfrak{r}} = 1$.

II. The nodes of $\mathfrak{T}_2$ with probability 1, i.e., the nodes $v \in N^{\mathfrak{T}_2}$ with $p_v = 1$, form a connected path. Let $v'$ be the node with the greatest depth within this path. Every leaf $w \in \mathrm{Leaf}(\mathfrak{T}_2)$ is a successor of $v'$, i.e., $w \in \mathrm{Post}(v')$. Thus, together with I. we obtain that every leaf $w \in \mathrm{Leaf}(\mathfrak{T}_2)$ has a predecessor $v \in \mathrm{Pre}(w)$ such that $v = \mathsf{e}(u)$ for some $u \in N^{\mathfrak{T}_1}$.

It remains to show that for every leaf $w \in \mathrm{Leaf}(\mathfrak{T}_2)$, there is a predecessor $v \in \mathrm{Pre}(w)$ and a leaf $u \in \mathrm{Leaf}(\mathfrak{T}_1)$ such that $v = \mathsf{e}(u)$. Thus, let $w \in \mathrm{Leaf}(\mathfrak{T}_2)$. By Observation II., there is some predecessor of $w$ in the image of $\mathsf{e}$. We choose such a predecessor $v = \mathsf{e}(u)$ with maximal depth. Assume that $u$ is not a leaf of $\mathfrak{T}_1$. Let $u_1, \dots, u_k$ be the direct successors of $u$ in $\mathfrak{T}_1$. By path preservation, all $\mathsf{e}(u_j)$ are successors of $v$. Moreover,

$$p_v = p_{\mathsf{e}(u)} = p_u = \sum_{j=1}^{k} p_{u_j} = \sum_{j=1}^{k} p_{\mathsf{e}(u_j)}.$$

Since $u_1, \dots, u_k$ are pairwise incomparable, the nodes $\mathsf{e}(u_1), \dots, \mathsf{e}(u_k)$ are pairwise incomparable as well by the proof of Lemma 9.1.6. Moreover, the probabilities of pairwise incomparable successors of a node $v$ sum up to at most $p_v$, and all nodes have a positive probability. Hence, the whole probability mass below $v$ is exhausted by the subtrees of $\mathfrak{T}_2$ with the roots $\mathsf{e}(u_1), \dots, \mathsf{e}(u_k)$: if the leaf $w$ below $v$ did not belong to one of these subtrees, then the probabilities of the pairwise incomparable nodes $\mathsf{e}(u_1), \dots, \mathsf{e}(u_k), w$ would sum up to $p_v + p_w > p_v$. So we have $\mathsf{e}(u_j) \in \mathrm{Pre}(w)$ for some $j$. But this contradicts the maximality of the depth of $v = \mathsf{e}(u)$. Hence, $u$ is a leaf. *QED*

With both of these lemmas, we are now able to prove the following main theorem to disprove `AST` or `PAST`.

**Theorem 9.1.8** – Disproving `AST` and `PAST` via Embeddings.

Let $(A_1, \to_1)$ and $(A_2, \to_2)$ be two PARSs, and let $\mathfrak{T}_i$ be a $\to_i$-RT for $i \in \{1, 2\}$ such that there exists an embedding $\mathsf{e}$ from $\mathfrak{T}_1$ to $\mathfrak{T}_2$. Then, $|\mathfrak{T}_2| \leq |\mathfrak{T}_1|$ and $\mathrm{edl}(\mathfrak{T}_1) \leq \mathrm{edl}(\mathfrak{T}_2)$.

*Proof.* We prove both inequalities individually. *Proof*

▶ *Proof*($|\mathfrak{T}_2| \leq |\mathfrak{T}_1|$): By Lemma 9.1.7, every leaf $v \in \mathrm{Leaf}(\mathfrak{T}_2)$ belongs to $\mathrm{Post}_{\mathrm{Leaf}}(\mathsf{e}(u))$ for some leaf $u \in \mathrm{Leaf}(\mathfrak{T}_1)$. By Lemma 9.1.6, these sets are pairwise disjoint. Therefore, the leaves of $\mathfrak{T}_2$ are partitioned by the sets $\mathrm{Post}_{\mathrm{Leaf}}(\mathsf{e}(u))$ for $u \in \mathrm{Leaf}(\mathfrak{T}_1)$. Moreover, for every such leaf $u$, we have $p_{\mathsf{e}(u)} = p_u$ and $\sum_{v \in \mathrm{Post}_{\mathrm{Leaf}}(\mathsf{e}(u))} p_v \leq p_{\mathsf{e}(u)}$. Thus, we obtain

$$\begin{aligned} |\mathfrak{T}_2| &= \textstyle\sum_{v \in \mathrm{Leaf}(\mathfrak{T}_2)} p_v &&= \textstyle\sum_{u \in \mathrm{Leaf}(\mathfrak{T}_1)} \sum_{v \in \mathrm{Post}_{\mathrm{Leaf}}(\mathsf{e}(u))} p_v \\ &\leq \textstyle\sum_{u \in \mathrm{Leaf}(\mathfrak{T}_1)} p_{\mathsf{e}(u)} &&= \textstyle\sum_{u \in \mathrm{Leaf}(\mathfrak{T}_1)} p_u \qquad = |\mathfrak{T}_1|. \end{aligned}$$

▶ *Proof*($\mathrm{edl}(\mathfrak{T}_1) \leq \mathrm{edl}(\mathfrak{T}_2)$): If $|\mathfrak{T}_2| < 1$, then we have $\mathrm{edl}(\mathfrak{T}_2) = \infty$ and thus, the claim is trivial. Hence, let $|\mathfrak{T}_2| = 1$. Due to $|\mathfrak{T}_2| \leq |\mathfrak{T}_1|$, this also implies $|\mathfrak{T}_1| = 1$.

Note that a path from the root to a leaf $v$ in $\mathfrak{T}_2$ is at least as long as the path from the root to the corresponding leaf $u$ in $\mathfrak{T}_1$ (where $\mathtt{e}(u) = w$ for a $w \in \mathrm{Pre}(v)$), since $\mathtt{e}$ is injective and "path preserving". Therefore, we have $\mathrm{d}(u) \leq \mathrm{d}(v)$ for all $v \in \mathrm{Post}_{\mathrm{Leaf}}(\mathtt{e}(u))$, and thus

$$\begin{array}{rcl}
\mathrm{edl}(\mathfrak{T}_2) & = & \sum_{u \in \mathrm{Leaf}(\mathfrak{T}_1)} \sum_{v \in \mathrm{Post}_{\mathrm{Leaf}}(\mathtt{e}(u))} \mathrm{d}(v) \cdot p_v \\
& \geq & \sum_{u \in \mathrm{Leaf}(\mathfrak{T}_1)} \sum_{v \in \mathrm{Post}_{\mathrm{Leaf}}(\mathtt{e}(u))} \mathrm{d}(u) \cdot p_v \\
& = & \sum_{u \in \mathrm{Leaf}(\mathfrak{T}_1)} \mathrm{d}(u) \cdot \sum_{v \in \mathrm{Post}_{\mathrm{Leaf}}(\mathtt{e}(u))} p_v \\
& = & \sum_{u \in \mathrm{Leaf}(\mathfrak{T}_1)} \mathrm{d}(u) \cdot p_{\mathtt{e}(u)} \\
& = & \sum_{u \in \mathrm{Leaf}(\mathfrak{T}_1)} \mathrm{d}(u) \cdot p_u \\
& = & \mathrm{edl}(\mathfrak{T}_1).
\end{array}$$

Here, the first and the last step use the second consequence above for $\mathfrak{T}_2$ and $\mathfrak{T}_1$, respectively. To conclude that $\sum_{v \in \mathrm{Post}_{\mathrm{Leaf}}(\mathtt{e}(u))} p_v = p_{\mathtt{e}(u)}$ in the fourth step, note that $|\mathfrak{T}_2| = 1$ implies $p_w = \sum_{v \in \mathrm{Post}_{\mathrm{Leaf}}(w)} p_v$ for every node $w$ of $\mathfrak{T}_2$. *QED*

SECTION 9.2

## Embedding Random Walks Based on Term Occurrences

We have seen in the previous section that embeddings can be used to disprove AST or PAST. It remains to show how to automatically find such embeddings and what kind of probabilistic reduction we start our embedding from. For the latter we use random walk programs from Section 2.3.2.

To embed random walks, we again search for a loop where $t$ rewrites to $C[t\sigma]$, but now we also consider the *probability* of these terms and the *number of loops* represented by the term $C[t\sigma]$.

Example 9.2.1. Consider the PTRS $\mathcal{P}_{14}$ with the rule $\mathtt{g}(x) \to \{{}^{2/3}x, {}^{1/3}\mathtt{g}(\mathtt{g}(\mathtt{g}(x)))\}$ modeling a symmetric random walk on the number of g's, because their number increases in each rule application by $2/3 \cdot (-1) + 1/3 \cdot 2 = 0$ in expectation. While $\mathcal{P}_{13}$ from Counterexample 9.1.1 is PAST, $\mathcal{P}_{14}$ is not. The difference is that the symbol g occurs three instead of two times in the second choice of the distribution on the right-hand side, see Figure 9.5.

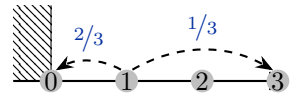


Figure 9.5: $\mathcal{P}_{14}$ as an RWP.

Example 9.2.1 illustrates that it is important how often a looping term like $\mathtt{g}(x)$ occurs on the right-hand side. In this example, the three subterms of $\mathtt{g}(\mathtt{g}(\mathtt{g}(x)))$ that are instances of $\mathtt{g}(x)$ can indeed be counted separately, because they do not overlap at a non-variable position.

We first solve the problem of finding the maximal number of such non-overlapping instances (called *occurrences*).

Definition 9.2.2 – Term Occurrences.
*Let $t, s \in \mathcal{T}$ be terms. We say that $t$ occurs in $s$ at position $\pi$ (denoted $t \blacktriangleleft_\pi s$) if $s|_\pi = t\sigma$ for some substitution $\sigma$. Two positions $\pi_1$ and $\pi_2$ are overlapping w.r.t. $t$ if there is a position $\tau \in \mathrm{pos}_\Sigma(t)$ such that $\pi_1 = \pi_2.\tau$ or $\pi_2 = \pi_1.\tau$. Let* $\mathrm{NO}(t, s)$

**Algorithm 8:** Compute $\mathrm{maxNO}(t, s)$ for $t \notin \mathcal{V}$

$q \leftarrow \mathrm{EmptyList}()$
**for** $s' \trianglelefteq s$ **do**
  $\alpha_{s'} \leftarrow 0,\ \beta_{s'} \leftarrow 0,\ \gamma_{s'} \leftarrow 0$ // Initialize values
q.enqueue(leaves) // Start with the leaves
**while** not q.isEmpty() **do**
  $s' \leftarrow$ q.dequeue()
  $\gamma_{s'} \leftarrow 1$
  **if** $s'$ is leaf **then**
    $\alpha_{s'} \leftarrow \begin{cases} 1, & \text{if } t \blacktriangleleft_\varepsilon s' \\ 0, & \text{otherwise} \end{cases}$ // $t$ matches $s'$ / // $t$ does not match $s'$
  **else**
    $\beta_{s'} \leftarrow \sum_{j=1}^{k} \alpha_{v_j}$ where $s' = f(v_1, \ldots, v_k)$
    $\alpha_{s'} \leftarrow \begin{cases} \max\left\{\beta_{s'}, \sum_{\pi \in \mathrm{pos}_{\mathcal{V}}(t)} \alpha_{s'|_\pi} + 1\right\}, & \text{if } t \blacktriangleleft_\varepsilon s' \\ \beta_{s'}, & \text{otherwise} \end{cases}$ // $t$ matches $s'$ / // $t$ does not match $s'$
  **if** $s'$ is not the root and $\forall v \in s'.\mathrm{parent}().\mathrm{children}() : \gamma_v = 1$ // All siblings of $s'$ processed **then**
    q.enqueue($s'$.parent()) // Enqueue parent
**return** $\alpha_s$

*denote the set of all sets of pairwise non-overlapping occurrences of $t$ in $s$. So for $S \subseteq \mathrm{pos}(s)$ we have $S \in \mathrm{NO}(t, s)$ iff*

- *if $\pi \in S$ then $t \blacktriangleleft_\pi s$, and*
- *if $\pi_1, \pi_2 \in S$ with $\pi_1 \neq \pi_2$ then $\pi_1, \pi_2$ are non-overlapping w.r.t. $t$.*

*Let $\mathrm{maxNO}(t, s) = \max_{S \in \mathrm{NO}(t,s)} |S|$ be the maximal cardinality of a set in $\mathrm{NO}(t, s)$.*

So if $t \blacktriangleleft_{\pi_1} s$ and $t \blacktriangleleft_{\pi_2} s$ where $\pi_1, \pi_2$ are non-overlapping w.r.t. $t$, then $\pi_1, \pi_2$ are either orthogonal, or we have $s|_{\pi_1} = t\sigma_1$ and $s|_{\pi_2} = t\sigma_2$ for some substitutions $\sigma_1, \sigma_2$ such that $t\sigma_2 \trianglelefteq \sigma_1(x)$ or $t\sigma_1 \trianglelefteq \sigma_2(x)$ for some variable $x \in \mathcal{V}(t)$.

Example 9.2.3. We have $\mathsf{g}(x) \blacktriangleleft_\pi \mathsf{g}(\mathsf{g}(\mathsf{g}(x)))$ for all $\pi \in \{\varepsilon, 1, 1.1\}$ and $\mathrm{NO}(\mathsf{g}(x), \mathsf{g}(\mathsf{g}(\mathsf{g}(x))))$ consists of all subsets of $\{\varepsilon, 1, 1.1\}$. Therefore, we have $\mathrm{maxNO}(\mathsf{g}(x), \mathsf{g}(\mathsf{g}(\mathsf{g}(x)))) = 3$. So there are three non-overlapping occurrences of the term $\mathsf{g}(x)$ in $\mathsf{g}(\mathsf{g}(\mathsf{g}(x)))$, i.e., these occurrences can indeed all be counted when analyzing non-termination.

Example 9.2.4. If we consider $\mathsf{g}(\mathsf{g}(x))$ instead of $\mathsf{g}(x)$, then $\mathsf{g}(\mathsf{g}(x)) \blacktriangleleft_\pi \mathsf{g}(\mathsf{g}(\mathsf{g}(x)))$ only holds for $\pi \in \{\varepsilon, 1\}$, and we have $\mathrm{NO}(\mathsf{g}(\mathsf{g}(x)), \mathsf{g}(\mathsf{g}(\mathsf{g}(x)))) = \{\varnothing, \{\varepsilon\}, \{1\}\}$, but $\{\varepsilon, 1\} \notin \mathrm{NO}(\mathsf{g}(\mathsf{g}(x)), \mathsf{g}(\mathsf{g}(\mathsf{g}(x))))$, because $\varepsilon$ and 1 are overlapping w.r.t. $\mathsf{g}(\mathsf{g}(x))$, since $1 \in \mathrm{pos}_\Sigma(\mathsf{g}(\mathsf{g}(x)))$. Thus, we just obtain $\mathrm{maxNO}(\mathsf{g}(\mathsf{g}(x)), \mathsf{g}(\mathsf{g}(\mathsf{g}(x)))) = 1$, i.e., we can only count one instead of two occurrences of the term $\mathsf{g}(\mathsf{g}(x))$ in $\mathsf{g}(\mathsf{g}(\mathsf{g}(x)))$. Such overlapping occurrences cannot be counted separately, because rewriting one occurrence may interfere with the possibility to rewrite the other one, see Counterexample 9.2.8.

Computing $\mathrm{maxNO}(t, s)$ can be done via a dynamic programming algorithm traversing the tree structure of $s$ from the leaves to the root, see Algorithm 8.[1] For every node, i.e., every subterm $s' \trianglelefteq s$, we compute two numbers: (1) $\alpha_{s'} = \mathrm{maxNO}(t, s')$ and (2)

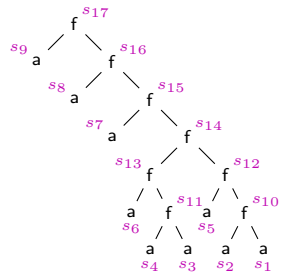
[1] A closely related problem is the vertex-disjoint tree-packing problem, which can be solved via a similar algorithm, see [Mas92].

$\beta_{s'} = \max_{S \in \mathrm{NO}(t,s'),\varepsilon \notin S} |S|$, i.e., the maximal number of non-overlapping occurrences of $t$ in $s'$ if we do not consider an occurrence of $t$ at the root of $s'$. For the leaves, we set $\beta_{s'} = 0$ and $\alpha_{s'} = 1$ if $t \blacktriangleleft_\varepsilon s'$ and $\alpha_{s'} = 0$ otherwise. For each inner node where $s'$ has the form $f(v_1, \ldots, v_k)$, we set $\beta_{s'} = \sum_{1 \leq j \leq k} \alpha_{v_j}$, and then check whether there is an occurrence of $t$ at the root of $s'$, i.e., whether $t \blacktriangleleft_\varepsilon s'$. If not, then we simply set $\alpha_{s'} = \beta_{s'} = \sum_{1 \leq j \leq k} \alpha_{v_j}$. If there is an occurrence of $t$ at the root of $s'$, then $\alpha_{s'} = \max\{\beta_{s'}, \sum_{\pi \in \mathrm{pos}_{\mathcal{V}}(t)} \alpha_{s'|_\pi} + 1\}$, i.e., $\alpha_{s'}$ is the maximum of $\beta_{s'}$ and the number obtained when considering the root position and maxNO for all subterms of $s'$ corresponding to variable positions of $t$. In Algorithm 8, we use a flag $\gamma_{s'}$ where $\gamma_{s'} = 1$ indicates that we have already computed the values $\alpha_{s'}$ and $\beta_{s'}$, and otherwise we have $\gamma_{s'} = 0$. Concerning the runtime of Algorithm 8, the algorithm has to consider each subterm $s'$ of $s$ and check whether $t$ matches $s'$, which runs in time $\mathcal{O}(|s'|)$, where $|s'|$ is the number of positions in $s'$. So Algorithm 8 runs in $\mathcal{O}(|s|^2)$.

Example 9.2.5. As an example, Figure 9.6 shows the dynamic programming table for the two terms $t = \mathsf{f}(\mathsf{a}, \mathsf{f}(\mathsf{a}, x))$ and $s = \mathsf{f}(\mathsf{a}, \mathsf{f}(\mathsf{a}, \mathsf{f}(\mathsf{a}, \mathsf{f}(\mathsf{f}(\mathsf{a}, \mathsf{f}(\mathsf{a}, \mathsf{a})), \mathsf{f}(\mathsf{a}, \mathsf{f}(\mathsf{a}, \mathsf{a}))))))$ computed by Algorithm 8. There exists no occurrence of $t$ in $s_i$ for any $1 \leq i \leq 11$. For $s_{12}$ and $s_{13}$ there is one occurrence of $t$ at the root. Therefore, we have $\alpha_{s_{12}} = \alpha_{s_{13}} = 1$. At $s_{14}$ we have no occurrence of $t$ at the root, but two occurrences below the root, thus $\alpha_{s_{14}} = \beta_{s_{14}} = 2$. The same holds for $s_{15}$. Then, we have another occurrence at the root of $s_{16}$ that does not overlap with the occurrences at the roots of $s_{12}$ and $s_{13}$, so $\alpha_{s_{16}} = 3$. Finally, there is another occurrence at the root of $s_{17}$, but it is overlapping (w.r.t. $t$) with the preceding occurrence at the root of $s_{16}$. Thus, we obtain $\mathrm{maxNO}(t, s) = \alpha_s = \alpha_{s_{17}} = 3$.

| Subterm $s'$ | $\alpha_{s'}$ | $\beta_{s'}$ |
|---|---|---|
| $s_1$–$s_{11}$ | 0 | 0 |
| $s_{12}$–$s_{13}$ | 1 | 0 |
| $s_{14}$ | 2 | 2 |
| $s_{15}$ | 2 | 2 |
| $s_{16}$ | 3 | 2 |
| $s_{17}$ | 3 | 3 |

(a) Dynamic programming table.




(b) Tree structure of $s$.




(c) Tree structure of $t$.

Figure 9.6: Computation of $\mathrm{maxNO}(t, s)$.

### 9.2.1 Counting Linear Term Occurrences

We now embed random walks based on the maximal number of non-overlapping occurrences of a looping term $t$. In general, we start with a $\to_{\mathcal{P}}$-RT $\mathfrak{T}$ with $\mathrm{root}(\mathfrak{T}) = t$ where $t$ is *linear* (we will explain the reason for the linearity requirement in Counterexample 9.2.9). Remember, a term $t \in \mathcal{T}$ is *linear* if $|t|_x \leq 1$ for all $x \in \mathcal{V}$, where $|t|_x$ denotes the number of positions $\pi \in \mathrm{pos}(t)$ such that $t|_\pi = x$.

Then, we extend the reduction tree $\mathfrak{T}$, i.e., we rewrite the terms in its leaves, until we reach a point where there are enough non-overlapping occurrences of $t$ in the leaves in order to disprove `AST` or `PAST`. Here, one stops after reaching a suitable finite RT $\mathfrak{T}$, and the embedding theorems of this and the next section require $\mathfrak{T}$ to be finite. We can then use the (finite) tree $\mathfrak{T}$ as a (finite) representation of an infinite $\to_{\mathcal{P}}$-RT $\mathfrak{T}^\infty$, where we can embed a random walk that disproves `AST` or `PAST`.

**Theorem 9.2.6** – Embedding Random Walks via Occurrences (1).

Let $\mathcal{P}$ be a PTRS. Let $\mathfrak{T}$ be a finite $\to_{\mathcal{P}}$-RT with $\mathrm{h}(\mathfrak{T}) > 0$ and $\mathrm{root}(\mathfrak{T}) = t$ where $t$ is linear.

- If we have $\sum_{v \in \mathrm{Leaf}(\mathfrak{T})} p_v \cdot \mathrm{maxNO}(t, t_v) > 1$, then $\mathcal{P}$ is not `AST`.
- If $\sum_{v \in \mathrm{Leaf}(\mathfrak{T})} p_v \cdot \mathrm{maxNO}(t, t_v) \geq 1$, then $\mathcal{P}$ is not `PAST`.

So for every leaf $v$, we multiply its probability $p_v$ by the number of non-overlapping occurrences of $t$ in the term $t_v$ of the leaf. This number of occurrences in the leaves gives rise to a random walk program $\mu$ with $\mu(x) = \sum_{v \in \mathrm{Leaf}(\mathfrak{T}), x = \mathrm{maxNO}(t,t_v)-1} p_v$ for all $x \in \mathbb{Z}$. The condition $\sum_{v \in \mathrm{Leaf}(\mathfrak{T})} p_v \cdot \mathrm{maxNO}(t, t_v) > 1$ means that $\mu$ is positively drifting, while $\sum_{v \in \mathrm{Leaf}(\mathfrak{T})} p_v \cdot \mathrm{maxNO}(t, t_v) \geq 1$ means that $\mu$ is a loop walk, symmetric, or positively drifting. By repeatedly rewriting an innermost occurrence of $t$ in the leaves of the tree according to the rules used in $\mathfrak{T}$, we obtain an infinite $\to_{\mathcal{P}}$-RT $\mathfrak{T}^{\infty}$, where we can embed the computation of the random walk $\mu$. This disproves `AST` or `PAST` of $\mathcal{P}$, respectively. An innermost occurrence must be chosen due to rules where a variable occurs more often on the left-hand side than on the right-hand side. Such rules might decrease the number of occurrences of $t$ in the arguments when rewriting at a non-innermost position. Note that it is not a problem if different instances of $t$ occur at different positions in a leaf. The reason is that if $t$ starts a loop, then so does every instance of $t$.

*Proof of Theorem 9.2.6.* If we have $|\mathfrak{T}| = \sum_{v \in \mathrm{Leaf}(\mathfrak{T})} p_v < 1$, then $\mathfrak{T}$ is itself a witness that $\mathcal{P}$ is not `AST` and not `PAST`. Therefore, we now consider the case $|\mathfrak{T}| = 1$.
We define the random walk $\mu$ by $\mu(x) = \sum_{v \in \mathrm{Leaf}(\mathfrak{T}), x = \mathrm{maxNO}(t,t_v)-1} p_v$ for all $x \in \mathbb{Z}$. Note that

*Proof*

$$\begin{aligned}
\mathbb{E}(\mu) &= \sum_{v \in \mathrm{Leaf}(\mathfrak{T})} p_v \cdot (\mathrm{maxNO}(t, t_v) - 1) \\
&= \Big(\sum_{v \in \mathrm{Leaf}(\mathfrak{T})} p_v \cdot \mathrm{maxNO}(t, t_v)\Big) - \Big(\sum_{v \in \mathrm{Leaf}(\mathfrak{T})} p_v\Big) \\
&\qquad \downarrow \quad (\text{since } \sum_{v \in \mathrm{Leaf}(\mathfrak{T})} p_v = 1) \\
&= \Big(\sum_{v \in \mathrm{Leaf}(\mathfrak{T})} p_v \cdot \mathrm{maxNO}(t, t_v)\Big) - 1.
\end{aligned}$$

Therefore, if $\sum_{v \in \mathrm{Leaf}(\mathfrak{T})} p_v \cdot \mathrm{maxNO}(t, t_v) > 1$, then we have $\mathbb{E}(\mu) > 0$, i.e., $\mu$ is positively drifting. Similarly, if $\sum_{v \in \mathrm{Leaf}(\mathfrak{T})} p_v \cdot \mathrm{maxNO}(t, t_v) \geq 1$, then we have $\mathbb{E}(\mu) \geq 0$, i.e., $\mu$ is a loop walk, symmetric, or positively drifting. Note that $\mu$ has finite support and hence a finite drift, since $\mathfrak{T}$ is finite.
We can represent $\mu$ as a PARS $(\mathbb{Z}, \hookrightarrow_{\mu})$ with $y \hookrightarrow_{\mu} \{{}^{p_v}y + \mathrm{maxNO}(t, t_v) - 1 \mid v \in \mathrm{Leaf}(\mathfrak{T})\}$ for every $y > 0$. Let $\mathfrak{T}_1$ be the infinite $\hookrightarrow_{\mu}$-RT that starts at 1. By Theorem 2.3.11 for the start value $x_0 = 1$, we have $|\mathfrak{T}_1| < 1$ if $\mu$ is positively drifting, and $\mathrm{edl}(\mathfrak{T}_1) = \infty$ if $\mu$ is a loop walk, symmetric, or positively drifting. The only difference between $\mu$ and the PARS $(\mathbb{Z}, \hookrightarrow_{\mu})$ is that $\hookrightarrow_{\mu}$ is defined via multi-distributions which may have multiple different pairs ${}^{p_1}x, \ldots, {}^{p_k}x$ for the same number $x$, while $\mu$ maps every value $x$ to a single probability ($\mu(x) = \sum_{i=1}^{k} p_i$). However, since for every number $x$ there is only a single rewrite rule with $x$ as its left-hand

side, the termination probability and expected derivation length are equal for $\mu$ and $(\mathbb{Z}, \hookrightarrow_\mu)$.

We construct an infinite $\to_{\mathcal{P}}$-RT $\mathfrak{T}^\infty$ based on $\mathfrak{T}$ where we always rewrite an innermost occurrence of $t$ in a leaf according to the rules used to generate $\mathfrak{T}$. If we have $\mathrm{maxNO}(t, t_v) = 0$ in a leaf $v$, then it remains a leaf in $\mathfrak{T}^\infty$. We have to prove that $\mathfrak{T}^\infty$ behaves like the reduction tree of our random walk program $\mu$.

We first show that if we have a term $s \in \mathcal{T}$ with $\mathrm{maxNO}(t, s) = k \in \mathbb{N}_{>0}$, then instead of the tree $\mathfrak{T}$ whose root is labeled with $t$, we can construct a tree $\mathfrak{T}_s$ whose root is labeled with $s$, where $\mathfrak{T}_s$ is like $\mathfrak{T}$ when rewriting an innermost occurrence of $t$ in $s$ instead, according to the rules used in $\mathfrak{T}$. Thus, we can define a bijective embedding $\mathsf{e}_s : N^{\mathfrak{T}} \to N^{\mathfrak{T}_s}$ and obtain $\mathrm{maxNO}(t, t_{\mathsf{e}_s(v)}) \geq \mathrm{maxNO}(t, t_v) + k - 1$ for all leaves $v \in \mathrm{Leaf}(\mathfrak{T})$. (While $k$ was the original number of occurrences of $t$ in $s$, this number is now modified according to the random walk $\mu$.) To prove this inequality, let $\{\pi_1, \ldots, \pi_k\} \in \mathrm{NO}(t, s)$ be a witness for the $k$ pairwise non-overlapping occurrences of $t$ in $s$. The term $s$ has the form $s = C[t\delta]$ for some context $C$ with a hole at position $\pi_1$ and substitution $\delta$. Let $\pi_1$ be the position of an innermost occurrence of $t$, i.e., there is no $\pi_j$ with $1 \leq j \leq k$ strictly below $\pi_1$. To construct $\mathfrak{T}_s$, we rewrite $C[t\delta]$ according to $\mathfrak{T}$ at position $\pi_1$. This is possible, since if $t$ can be rewritten via the rules used in $\mathfrak{T}$, then so can every instance of $t$. Let $v \in \mathrm{Leaf}(\mathfrak{T})$ and $\mathsf{e}_s(v) \in \mathrm{Leaf}(\mathfrak{T}_s)$ be the corresponding leaf in $\mathfrak{T}_s$ which we get by following the same path in $\mathfrak{T}_s$ as in $\mathfrak{T}$. We have $t_{\mathsf{e}_s(v)} = C[t_v\delta]$. Since $t$ is linear and $\{\pi_1, \ldots, \pi_k\}$ are pairwise non-overlapping w.r.t. $t$, we still have $\{\pi_2, \ldots, \pi_k\} \in \mathrm{NO}(t, C[t_v\delta])$. Moreover, all occurrences of $t$ in $t_v$ also exist in $t_v\delta$. Finally, the occurrences of $t$ in $t_v\delta$ are non-overlapping with $\{\pi_2, \ldots, \pi_k\}$ because they are below $\pi_1$. Thus, we get $\mathrm{maxNO}(t, t_{\mathsf{e}_s(v)}) = \mathrm{maxNO}(t, C[t_v\delta]) \geq \mathrm{maxNO}(t, t_v\delta) + k - 1 \geq \mathrm{maxNO}(t, t_v) + k - 1$.

Now we define the embedding $\mathsf{e} : N^{\mathfrak{T}_1} \to N^{\mathfrak{T}^\infty}$ from the $\hookrightarrow_\mu$-RT by induction on the depth of the node in $\mathfrak{T}_1$. Moreover, for every node $w \in N^{\mathfrak{T}_1}$ that is labeled with the number $a_w$ in $\mathfrak{T}_1$, we prove $a_w \leq \mathrm{maxNO}(t, t_{\mathsf{e}(w)})$.

We start by mapping the root $w$ of $\mathfrak{T}_1$ to the root of $\mathfrak{T}^\infty$. Here, we have $a_w = 1$ and $\mathrm{maxNO}(t, t_{\mathsf{e}(w)}) = \mathrm{maxNO}(t, t) = 1$.

Now assume that we have already defined the mapping for a node $w \in N^{\mathfrak{T}_1}$ and we have $a_w \leq \mathrm{maxNO}(t, t_{\mathsf{e}(w)})$. Every child $u$ of $w$ in $N^{\mathfrak{T}_1}$ corresponds to an element ${}^{p_v}a_w + \mathrm{maxNO}(t, t_v) - 1$ of the multi-distribution in the rule of $\hookrightarrow_\mu$ for $a_w$, i.e., to a leaf $v \in \mathrm{Leaf}(\mathfrak{T})$, and it is labeled by the number $a_u = a_w + x$ with $x = \mathrm{maxNO}(t, t_v) - 1$. Consider the tree $\mathfrak{T}_{t_{\mathsf{e}(w)}}$. As $a_w \leq \mathrm{maxNO}(t, t_{\mathsf{e}(w)})$, we know that if $w$ has a child in $\mathfrak{T}_1$, then $1 \leq a_w \leq \mathrm{maxNO}(t, t_{\mathsf{e}(w)})$. Thus, by the inequality proved above, there is a bijective embedding $\mathsf{e}_{t_{\mathsf{e}(w)}}$ of $\mathfrak{T}$ into $\mathfrak{T}_{t_{\mathsf{e}(w)}}$ where $\mathrm{maxNO}(t, t_{\mathsf{e}_{t_{\mathsf{e}(w)}}(v)}) \geq \mathrm{maxNO}(t, t_v) + k - 1$ for $k = \mathrm{maxNO}(t, t_{\mathsf{e}(w)})$. As $a_w \leq k$, this implies $\mathrm{maxNO}(t, t_{\mathsf{e}_{t_{\mathsf{e}(w)}}(v)}) \geq \mathrm{maxNO}(t, t_v) + a_w - 1$. In $\mathfrak{T}^\infty$, the node $\mathsf{e}(w)$ eventually reaches a node corresponding to $\mathsf{e}_{t_{\mathsf{e}(w)}}(v)$. Thus, $\mathsf{e}(u)$ is defined to be this node. Then we have $a_u = a_w + x = a_w + \mathrm{maxNO}(t, t_v) - 1 \leq \mathrm{maxNO}(t, t_{\mathsf{e}_{t_{\mathsf{e}(w)}}(v)}) = \mathrm{maxNO}(t, t_{\mathsf{e}(u)})$.

Injectivity, path preservation, and equal probabilities for the embedding $\mathsf{e}$ follow by construction.

By Theorem 9.1.8, we get $|\mathfrak{T}^\infty| \leq |\mathfrak{T}_1|$ and $\mathrm{edl}(\mathfrak{T}^\infty) \geq \mathrm{edl}(\mathfrak{T}_1)$. So if $|\mathfrak{T}_1| < 1$ holds, then $\mathcal{P}$ is not AST, and if $\mathrm{edl}(\mathfrak{T}_1) = \infty$ holds, then $\mathcal{P}$ is not PAST. *QED*

We now illustrate Theorem 9.2.6 with a concrete example.

Example 9.2.7 (Embedding via Innermost Occurrences). Consider the PTRS $\mathcal{P}_{15}$ with the rules $\mathsf{g}(x) \to \{{}^{1/2}\mathsf{a}, {}^{1/2}\mathsf{f}(x)\}$ and $\mathsf{f}(x) \to \{{}^{1}\mathsf{g}(\mathsf{g}(x))\}$. If we count the occurrences

of $\mathsf{g}(x)$, then the $\to_{\mathcal{P}_{15}}$-RT $\mathfrak{T}$ and the symmetric random walk $\mu$, both depicted in Figure 9.7, can be used to disprove PAST of $\mathcal{P}_{15}$. Here, we represent $\mu$ as a PARS $(\mathbb{Z}, \hookrightarrow_\mu)$ with $y \hookrightarrow_\mu \{^{p_v} y + \mathrm{maxNO}(t, t_v) - 1 \mid v \in \mathrm{Leaf}(\mathfrak{T})\}$ for every $y > 0$. The $\hookrightarrow_\mu$-RT with infinite expected derivation length can be embedded in the tree that results from extending $\mathfrak{T}$ by rewriting the innermost subterm $\mathsf{g}(x)$ repeatedly according to the rules used in $\mathfrak{T}$. So since $\mu$ is not PAST, $\mathcal{P}_{15}$ is not PAST either. However, since $\mu$ is AST, this does not yield any information about whether $\mathcal{P}_{15}$ is AST.

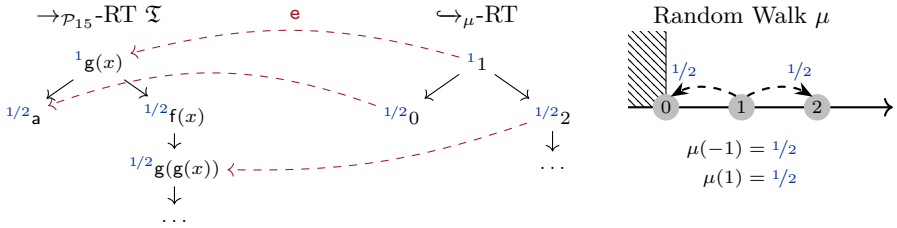



Figure 9.7: Embedding of a random walk into a $\to_{\mathcal{P}_{15}}$-RT to disprove PAST of $\mathcal{P}_{15}$.

The conditions on $t$, namely that we only count non-overlapping occurrences of $t$, and that $t$ is linear, cannot be dropped as shown in the following two counterexamples.

Counterexample 9.2.8 (Non-Overlappingness is Required). Non-overlappingness of the different occurrences of $t$ in a leaf guarantees that rewriting an innermost occurrence of $t$ does not interfere with the possibility to rewrite the other occurrences of $t$ later on. To see this, consider the PTRS $\mathcal{P}_{16}$ with the rule $\mathsf{g}(\mathsf{g}(x)) \to \{^{1/3}x, {}^{2/3}\mathsf{g}(\mathsf{g}(\mathsf{g}(x)))\}$, which is AST. If one counted both occurrences of $\mathsf{g}(\mathsf{g}(x))$ in the term $\mathsf{g}(\mathsf{g}(\mathsf{g}(x)))$ in spite of their overlap, then one could embed the random walk $\mu_3$ from Figure 2.2, and thus, falsely disprove AST of $\mathcal{P}_{16}$. Here, the problem is that rewriting the innermost subterm $\mathsf{g}(\mathsf{g}(x))$ of $\mathsf{g}(\mathsf{g}(\mathsf{g}(x)))$ could yield $\mathsf{g}(x)$, i.e., then the outermost occurrence of $\mathsf{g}(\mathsf{g}(x))$ in $\mathsf{g}(\mathsf{g}(\mathsf{g}(x)))$ would be "destroyed".

Counterexample 9.2.9 (Linearity of $t$ is Required). Linearity of $t$ is required in Theorem 9.2.6, because otherwise rewriting an innermost occurrence of $t$ in a leaf may "destroy" other occurrences of $t$ in that leaf. As an example, consider the PTRS $\mathcal{P}_{17}$ with the rule $\mathsf{f}(x, x) \to \{^{1/3}\mathsf{a}, {}^{1/3}\mathsf{b}, {}^{1/3}\mathsf{f}(\mathsf{f}(x, x), \mathsf{f}(x, x))\}$. If we count the occurrences of the term $\mathsf{f}(x, x)$, then the $\to_{\mathcal{P}_{17}}$-RT $\mathfrak{T}$ where we perform a single rewrite step starting in $\mathsf{f}(x, x)$ gives rise to the random walk $\mu_5$ with $\mu_5(-1) = 2/3$ and $\mu_5(2) = 1/3$, since $\mathrm{maxNO}(\mathsf{f}(x, x), \mathsf{f}(\mathsf{f}(x, x), \mathsf{f}(x, x))) = 3$. Since $\mu_5$ is not PAST, we would disprove PAST of $\mathcal{P}_{17}$. However, $\mathcal{P}_{17}$ is PAST (even SAST), so this disproof would be wrong. The problem is that rewriting the proper subterms of $\mathsf{f}(\mathsf{f}(x, x), \mathsf{f}(x, x))$ may yield terms like $\mathsf{f}(\mathsf{a}, \mathsf{b})$, where the two arguments of $\mathsf{f}$ are not equal. Thus, rewriting an innermost occurrence of $t = \mathsf{f}(x, x)$ in $\mathsf{f}(\mathsf{f}(x, x), \mathsf{f}(x, x))$ may "destroy" the occurrence of $t$ at the root.

### 9.2.2 Counting in Non-Variable-Decreasing RTs

When rewriting innermost occurrences of $t$ according to the rules used in $\mathfrak{T}$, we need linearity of $t$. Instead, one could also try to rewrite outermost occurrences. Then, instead of linearity of $t$, we have to require that there is no leaf $v \in \mathrm{Leaf}(\mathfrak{T})$ where a variable occurs more often in the looping term $t$ than in the term $t_v$ of the leaf $v$.

**Definition 9.2.10** – Non-Variable-Decreasing.

*A $\to_{\mathcal{P}}$-RT $\mathfrak{T}$ is non-variable-decreasing (*NVD*) if it satisfies $|\operatorname{root}(\mathfrak{T})|_x \leq |t_v|_x$ for all $v \in \operatorname{Leaf}(\mathfrak{T})$ and all $x \in \mathcal{V}$.*

Rewriting an outermost occurrence of $t$ according to an NVD $\to_{\mathcal{P}}$-RT $\mathfrak{T}$ does not affect any other non-overlapping occurrences of $t$.

**Theorem 9.2.11** – Embedding Random Walks via Occurrences (2).

Let $\mathcal{P}$ be a PTRS and let $\mathfrak{T}$ be a finite NVD $\to_{\mathcal{P}}$-RT with $\mathrm{h}(\mathfrak{T}) > 0$ and $\operatorname{root}(\mathfrak{T}) = t$.

- If we have $\sum_{v \in \operatorname{Leaf}(\mathfrak{T})} p_v \cdot \operatorname{maxNO}(t, t_v) > 1$, then $\mathcal{P}$ is not AST.
- If we have $\sum_{v \in \operatorname{Leaf}(\mathfrak{T})} p_v \cdot \operatorname{maxNO}(t, t_v) \geq 1$, then $\mathcal{P}$ is not PAST.

*Proof.* The proof is analogous to the one of Theorem 9.2.6, but now we construct the infinite $\to_{\mathcal{P}}$-RT $\mathfrak{T}^\infty$ based on $\mathfrak{T}$ by always rewriting an outermost occurrence of $t$. Again, if $s \in \mathcal{T}$ with $\operatorname{maxNO}(t, s) = k \in \mathbb{N}_{>0}$, then instead of the tree $\mathfrak{T}$ whose root is labeled with $t$, we show that we can construct a tree $\mathfrak{T}_s$ whose root is labeled with $s$, where $\mathfrak{T}_s$ is like $\mathfrak{T}$ when rewriting an outermost occurrence of $t$ in $s$ instead, according to the rules used in $\mathfrak{T}$. As in the proof of Theorem 9.2.6, we can define a bijective embedding $\mathtt{e}_s : N^{\mathfrak{T}} \to N^{\mathfrak{T}_s}$ and obtain $\operatorname{maxNO}(t, t_{\mathtt{e}_s(v)}) \geq \operatorname{maxNO}(t, t_v) + k - 1$ for all leaves $v \in \operatorname{Leaf}(\mathfrak{T})$. To prove this inequality, let $\{\pi_1, \ldots, \pi_k\} \in \operatorname{NO}(t, s)$ be a witness for the $k$ pairwise non-overlapping occurrences in $s$. The term $s$ has the form $s = C[t\delta]$ for some context $C$ with a hole at position $\pi_1$ and substitution $\delta$. Let $\pi_1$ be the position of an outermost occurrence of $t$, i.e., there is no $\pi_j$ with $1 \leq j \leq k$ strictly above $\pi_1$. We rewrite $C[t\delta]$ according to $\mathfrak{T}$ at position $\pi_1$. Since $\mathfrak{T}$ is non-variable-decreasing, for every position $\tau \in \operatorname{pos}_{\mathcal{V}}(t)$ and every leaf $v \in \operatorname{Leaf}(\mathfrak{T})$, there exists a position $\tau' \in \operatorname{pos}_{\mathcal{V}}(t_v)$ with $t|_\tau = t_v|_{\tau'}$, where different positions $\tau$ can be mapped to different positions $\tau'$, as $\mathfrak{T}$ is NVD.
As in the proof of Theorem 9.2.6, we get $t_{\mathtt{e}_s(v)} = C[t_v\delta]$. All occurrences of $t$ at positions that are orthogonal to $\pi_1$ are still present in $t_{\mathtt{e}_s(v)}$. All occurrences at a position below $\pi_1$, i.e., $\pi_j = \pi_1.\kappa$, can be written as $\pi_j = \pi_1.\tau.\beta$ for some position $\beta \in \mathbb{N}^*$ and $\tau \in \operatorname{pos}_{\mathcal{V}}(t)$. Moreover, since $t \trianglelefteq_{\pi_j} C[t\delta]$ we get $t \trianglelefteq_\beta \delta(x)$ for the variable $x = t|_\tau$. By the previous paragraph, we can find such a position $\tau' \in \operatorname{pos}_{\mathcal{V}}(t_v)$ with $t|_\tau = t_v|_{\tau'}$ (injectively for the different $\tau$), and therefore, all such occurrences are still present in $t_{\mathtt{e}_s(v)} = C[t_v\delta]$, because we still have $t \trianglelefteq_{\pi_1.\tau'.\beta} C[t_v\delta] \iff t \trianglelefteq_\beta \delta(x)$ for the variable $x = t_v|_{\tau'} = t|_\tau$. Note that there are no occurrences above $\pi_1$ as it is an outermost occurrence. Let $S'$ result from $\{\pi_2, \ldots, \pi_k\}$ by replacing every position $\pi_j = \pi_1.\tau.\beta$ below $\pi_1$ by $\pi_1.\tau'.\beta$, where $\tau'$ is the image of $\tau$ described above. Then we obtain $S' \in \operatorname{NO}(t, C[t_v\delta])$ with $|S'| = k - 1$. Moreover, all occurrences of $t$ in $t_v$ also exist in $t_v\delta$. Finally, the occurrences of $t$ in $t_v$ (at positions $\pi_1.\chi$ in $C[t_v\delta]$) are non-overlapping with the occurrences of $S'$: the positions of $S'$ are orthogonal to $\pi_1$ or at or below a variable position $\pi_1.\tau'$ of $t_v$, and such a position can neither be a function symbol position of $t$ at $\pi_1.\chi$ nor lie strictly above a position $\pi_1.\chi$ with $\chi \in \operatorname{pos}(t_v)$. Hence, the $k - 1$ occurrences of $S'$ together with a maximal set of non-overlapping occurrences of $t$ in $t_v$ are pairwise non-overlapping in $C[t_v\delta]$, and therefore we get $\operatorname{maxNO}(t, t_{\mathtt{e}_s(v)}) = \operatorname{maxNO}(t, C[t_v\delta]) \geq \operatorname{maxNO}(t, t_v) + k - 1$ as desired. The rest of the proof (i.e., the definition of the embedding from the $\hookrightarrow_\mu$-RT to $\mathfrak{T}^\infty$) is as in the proof of Theorem 9.2.6. *QED*

To demonstrate an application of Theorem 9.2.11, we consider the following example.

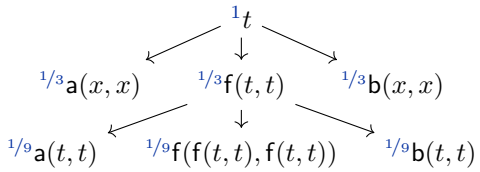



Figure 9.8: $\to_{\mathcal{P}'_{17}}$-RT.

Example 9.2.12 (Embedding via Outermost Occurrences). Consider the PTRS $\mathcal{P}'_{17}$ (similar to $\mathcal{P}_{17}$ from Counterexample 9.2.9) with the only rule $\mathsf{f}(x,x) \to \{^{1/3}\mathsf{a}(x,x), ^{1/3}\mathsf{b}(x,x), ^{1/3}\mathsf{f}(\mathsf{f}(x,x),\mathsf{f}(x,x))\}$. Rewriting at the outermost position yields the NVD $\to_{\mathcal{P}'_{17}}$-RT $\mathfrak{T}$ in Figure 9.8, where $t = \mathsf{f}(x,x)$. Because of $\mathrm{maxNO}(t,\mathsf{a}(t,t)) = \mathrm{maxNO}(t,\mathsf{b}(t,t)) = 2$, $\mathrm{maxNO}(t,\mathsf{f}(\mathsf{f}(t,t),\mathsf{f}(t,t))) = 7$, and $\mathrm{maxNO}(t,\mathsf{a}(x,x)) = \mathrm{maxNO}(t,\mathsf{b}(x,x)) = 0$, we obtain $\sum_{v\in\mathrm{Leaf}(\mathfrak{T})} p_v \cdot \mathrm{maxNO}(t,t_v) = {}^{11}/_{9} > 1$. Thus, by Theorem 9.2.11 this disproves AST and PAST of $\mathcal{P}'_{17}$.

### 9.2.3 Counting Orthogonal Term Occurrences in RTs

If $t$ is not linear and the RT $\mathfrak{T}$ is not NVD, then we can only count *orthogonal* occurrences.

Definition 9.2.13 – Orthogonal Occurrences.
*The maximal number of orthogonal occurrences of $t$ in $s$ is*

$$\mathrm{maxOO}(t,s) = \max\{|S| \mid S \in \mathrm{NO}(t,s), \forall \pi_1,\pi_2 \in S.\ \pi_1 \neq \pi_2 \implies \pi_1 \perp \pi_2\}.$$

To compute $\mathrm{maxOO}(t,s)$, we can adjust the computation of $\alpha_{s'}$ for inner nodes $s'$ in Algorithm 8 in the case of $t \trianglelefteq_\varepsilon s'$. Instead of setting $\alpha_{s'}$ to the maximum of $\beta_{s'}$ and $\sum_{\pi\in\mathrm{pos}_{\mathcal{V}}(t)} \alpha_{s'|_\pi} + 1$, we set $\alpha_{s'}$ to $\max\{\beta_{s'},1\}$, because now we do not count occurrences below another occurrence anymore. The runtime of the adjusted algorithm is still in $\mathcal{O}(|s|^2)$.

Example 9.2.14. In Figure 9.9 we can see the dynamic programming table for the two terms $t = \mathsf{f}(\mathsf{a},\mathsf{f}(\mathsf{a},x))$ and $s = \mathsf{f}(\mathsf{a},\mathsf{f}(\mathsf{a},\mathsf{f}(\mathsf{a},\mathsf{f}(\mathsf{f}(\mathsf{a},\mathsf{f}(\mathsf{a},\mathsf{a})),\mathsf{f}(\mathsf{a},\mathsf{f}(\mathsf{a},\mathsf{a}))))))$ computed by the algorithm for $\mathrm{maxOO}(t,s)$ described above. Initially, the example begins in the same way as in Example 9.2.5. There exists no occurrence of $t$ in $s_i$ for any $1 \leq i \leq 11$. For $s_{12}$ and $s_{13}$ there is one occurrence of $t$ at the root. Therefore, we have $\alpha_{s_{12}} = \alpha_{s_{13}} = 1$. Beginning with $s_{14}$ up to $s_{17}$ we either have one occurrence or no occurrence of $t$ at the root, but always two occurrences below the root, thus $\alpha_{s_i} = \beta_{s_i} = 2$ for all $14 \leq i \leq 17$. Thus, we obtain $\mathrm{maxOO}(t,s) = \alpha_s = \alpha_{s_{17}} = 2$.

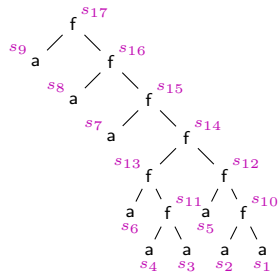
Counting orthogonal occurrences instead of non-overlapping occurrences yields the following theorem. In this case, the choice of the rewritten occurrence for generating the infinite extension $\mathfrak{T}^\infty$ does not matter anymore, due to the orthogonality of all occurrences.

Theorem 9.2.15 – Embedding Random Walks via Occurrences (3).
Let $\mathcal{P}$ be a PTRS and let $\mathfrak{T}$ be a finite $\to_{\mathcal{P}}$-RT with $\mathrm{h}(\mathfrak{T}) > 0$ and $\mathrm{root}(\mathfrak{T}) = t$.

- If we have $\sum_{v\in\mathrm{Leaf}(\mathfrak{T})} p_v \cdot \mathrm{maxOO}(t,t_v) > 1$, then $\mathcal{P}$ is not AST.

| Subterm $s'$ | $\alpha_{s'}$ | $\beta_{s'}$ |
|---|---|---|
| $s_1$–$s_{11}$ | 0 | 0 |
| $s_{12}$–$s_{13}$ | 1 | 0 |
| $s_{14}$ | 2 | 2 |
| $s_{15}$ | 2 | 2 |
| $s_{16}$ | 2 | 2 |
| $s_{17}$ | 2 | 2 |

(a) Dynamic programming table.




(b) Tree structure of $s$.




(c) Tree structure of $t$.

Figure 9.9: Computation of $\mathrm{maxOO}(t, s)$.

- If we have $\sum_{v \in \mathrm{Leaf}(\mathfrak{T})} p_v \cdot \mathrm{maxOO}(t, t_v) \geq 1$, then $\mathcal{P}$ is not PAST.

*Proof.* Compared to the proof of Theorem 9.2.6, we now define the random walk $\mu$ via $\mathrm{maxOO}(t, t_v)$ instead of $\mathrm{maxNO}(t, t_v)$. The remaining proof only differs in the proof of the inequality $\mathrm{maxOO}(t, t_{\mathsf{e}_s(v)}) \geq \mathrm{maxOO}(t, t_v) + k - 1$, where $\mathrm{maxOO}(t, s) = k \in \mathbb{N}_{>0}$. Let $\{\pi_1, \ldots, \pi_k\} \in \mathrm{NO}(t, s)$ be a witness for the $k$ orthogonal occurrences of $t$ in $s$. The term $s$ has the form $s = C[t\delta]$ for some context $C$ with a hole at position $\pi_1$ and substitution $\delta$. We rewrite $C[t\delta]$ according to $\mathfrak{T}$ at position $\pi_1$, and get $t_{\mathsf{e}_s(v)} = C[t_v\delta]$. All the occurrences of $t$ at the positions $\pi_2, \ldots, \pi_k$ are still present, because positions orthogonal to $\pi_1$ remain in $C[t_v\delta]$. Moreover, all orthogonal occurrences of $t$ in $t_v\delta$ are orthogonal with all occurrences at a position in $\{\pi_2, \ldots, \pi_k\}$ because they are below $\pi_1$. Hence, we get $\mathrm{maxOO}(t, t_{\mathsf{e}_s(v)}) = \mathrm{maxOO}(t, C[t_v\delta]) \geq \mathrm{maxOO}(t, t_v\delta) + k - 1 \geq \mathrm{maxOO}(t, t_v) + k - 1$ as desired. *Proof* *QED*

We illustrate Theorem 9.2.15 with the following example.

Example 9.2.16 (Embedding via Orthogonal Occurrences). Consider the PTRS $\mathcal{P}''_{17}$ with the only rule $\mathsf{f}(x, x) \to \{{}^{1/4}\mathsf{a}(x), {}^{1/4}\mathsf{b}(x), {}^{1/2}\mathsf{c}(\mathsf{f}(x, x), \mathsf{f}(x, x))\}$. Consider the $\to_{\mathcal{P}''_{17}}$-RT $\mathfrak{T}$ induced by the only rewrite rule, and let $t = \mathsf{f}(x, x)$. Because of $\mathrm{maxOO}(t, \mathsf{a}(x)) = \mathrm{maxOO}(t, \mathsf{b}(x)) = 0$, and $\mathrm{maxOO}(t, \mathsf{c}(\mathsf{f}(x, x), \mathsf{f}(x, x))) = 2$, we obtain $\sum_{v \in \mathrm{Leaf}(\mathfrak{T})} p_v \cdot \mathrm{maxOO}(t, t_v) = 1$. Thus, by Theorem 9.2.15 this disproves PAST of $\mathcal{P}''_{17}$.

To automate Theorems 9.2.6, 9.2.11 and 9.2.15, we have to find a $\to_{\mathcal{P}}$-RT satisfying one of the two constraints. To this end, we first search for a term $t$ that loops at all. Here, we reuse the non-probabilistic loop detection algorithms from [GTS05] on the non-probabilistic variant $\mathrm{np}(\mathcal{P})$. After finding a loop of $\mathrm{np}(\mathcal{P})$ starting with a term $t$ (which is already an undecidable problem in general), we first check whether $t$ is linear. If $t$ is linear, then we reconstruct the corresponding $\to_{\mathcal{P}}$-RT $\mathfrak{T}$ for this path. On all remaining terms $u$ in the leaves of $\mathfrak{T}$, we check whether we can possibly reach a term $s$ from $u$ such that $t \trianglelefteq_\pi s$ for some $\pi \in \mathrm{pos}(s)$. This is undecidable in general, but we use the *symbol transition graph* from [SY19] as a sufficient criterion for non-reachability. On those terms $u$ where we may potentially reach such a term $s$, we extend the RT by performing further *rewrite steps*, in order to obtain leaves that contain more occurrences of $t$. This is performed repeatedly until we have constructed an RT that satisfies one of the conditions of Theorem 9.2.6 or until one reaches a certain threshold for the number of rewrite steps. In the latter case, we try to find another looping term to generate an RT in order to embed a suitable random walk. If

the looping term $t$ is not linear, then we proceed in an analogous way in order to apply Theorem 9.2.11 or Theorem 9.2.15, depending on whether the initial tree $\mathfrak{T}$ is NVD or not. In the former case we compute both values $\mathrm{maxNO}(t, t_v)$ and $\mathrm{maxOO}(t, t_v)$ for every leaf $v$, since rewriting the leaves of $\mathfrak{T}$ may yield a tree that is not NVD anymore.

> Remark 9.2.17 – Theoretical Comparison of Embedding Techniques.
>
> Since $\mathrm{maxNO}(t, s) \geq \mathrm{maxOO}(t, s)$ for all terms $t, s$, Theorem 9.2.6 subsumes Theorem 9.2.15 for linear $t$, and Theorem 9.2.11 subsumes Theorem 9.2.15 for NVD trees. Apart from that, the techniques are incomparable:
>
> - Only Theorem 9.2.6 can disprove PAST of $\mathcal{P}_{15}$ (Example 9.2.7).
> - Only Theorem 9.2.15 can disprove PAST of $\mathcal{P}''_{17}$ (Example 9.2.16).
> - Only Theorem 9.2.11 can disprove PAST of the PTRS $\mathcal{P}'''_{17}$ with the only rule $\mathsf{f}(x,x,y) \to \{{}^{1/2}\mathsf{a}(x,x,y), {}^{1/2}\mathsf{f}(x,x,\mathsf{f}(x,x,y))\}$ over the signature induced by the PTRS: rewriting $t = \mathsf{f}(x,x,y)$ once yields an NVD RT with $1/2 \cdot 0 + 1/2 \cdot 2 = 1$, whereas every term with a redex is non-linear (as there are no constants), and the occurrences of the root term in the leaves of any RT are never orthogonal, since they all lie on the path of third arguments.

SECTION 9.3

## Embedding Random Walks Based on Pattern Terms

Instead of counting the occurrences of a single term, we can also count the number of *instantiations* applied to a certain *base term*.

Counterexample 9.3.1. Consider the PTRS $\mathcal{P}_{18}$ with the only rule $\mathsf{f}(\mathsf{g}(x)) \to \{{}^{1/3}\mathsf{f}(x), {}^{2/3}\mathsf{f}(\mathsf{g}(\mathsf{g}(x)))\}$ modeling a positively drifting random walk on the number of g's *directly below an* f in a term (illustrated in Figure 9.10) and the corresponding $\to_{\mathcal{P}_{18}}$-RT in Figure 9.11. Here, it is not enough to count the occurrences of $\mathsf{f}(\mathsf{g}(x))$ to disprove AST, but instead we have to count the occurrences of $\mathsf{g}(x)$ below an f symbol. Moreover, note that the f in $\mathsf{f}(x)$ is crucial. The PTRS $\mathcal{P}'_{18}$ with the rule $\mathsf{f}(\mathsf{g}(x)) \to \{{}^{1/3}x, {}^{2/3}\mathsf{f}(\mathsf{g}(\mathsf{g}(x)))\}$ does not model a random walk anymore, since every rewrite step of a term $\mathsf{f}(\mathsf{g}^n(x))$ yields the normal form $\mathsf{g}^{n-1}(x)$ with probability $1/3$ (removing the outer f). So $\mathcal{P}_{18}$ is not AST, but $\mathcal{P}'_{18}$ is even PAST.

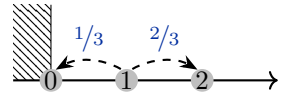


Figure 9.10: $\mathcal{P}_{18}$ as an RWP.

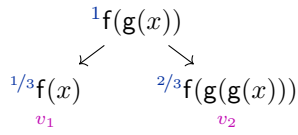


Figure 9.11: $\to_{\mathcal{P}_{18}}$-RT with the leaves $v_1$ and $v_2$.

We formalize this with the concept of a *pattern term* $\langle t, \sigma \rangle$ that represents all terms which result from applying the *pumping substitution* $\sigma$ repeatedly to the *base term* $t$, i.e., $\langle t, \sigma \rangle$ represents the terms $t$, $t\sigma$, $t\sigma^2$, etc. For example, if $t = \mathsf{f}(x)$ and $\sigma = [x/\mathsf{g}(x)]$, then $\langle t, \sigma \rangle$ represents the terms $\mathsf{f}(x)$, $\mathsf{f}(\mathsf{g}(x))$, $\mathsf{f}(\mathsf{g}(\mathsf{g}(x)))$, etc. A similar notion was defined in [EEG12] to prove *non-looping non-termination*. In contrast, here we want to

*count loops*, and we want to count how often the substitution $\sigma$ is applied. Therefore, we have to distinguish $t\sigma^m$ and $t\sigma^{m'}$ whenever $m \neq m'$.

Definition 9.3.2 – Pattern Term.

*A pair $\langle t, \sigma \rangle$ is a pattern term if $t\sigma^m \neq t\sigma^{m'}$ holds whenever $m \neq m'$. We call $t$ the base term and $\sigma$ the pumping substitution.*

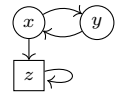
To check whether $\langle t, \sigma \rangle$ is a pattern term, we use the *variable transition graph*.

Definition 9.3.3 – Variable Transition Graph.

*For any substitution $\sigma$, $G_\sigma$ is the graph that has all variables as nodes and there is an edge from $x$ to $y$ if $y \in \mathcal{V}(x\sigma)$. For any term $t$, the variable transition graph of $\sigma$ w.r.t. $t$ is the subgraph $G_{\sigma,t}$ of $G_\sigma$ which contains only those nodes that are reachable from a node in $\mathcal{V}(t)$.*

Here and in the following, $\mathrm{dom}(\sigma) = \{x \in \mathcal{V} \mid x\sigma \neq x\}$ denotes the (finite) domain of a substitution $\sigma$, and for two substitutions $\sigma$ and $\kappa$, their composition $\sigma\kappa$ is the substitution with $x(\sigma\kappa) = (x\sigma)\kappa$ for all $x \in \mathcal{V}$.

Example 9.3.4. Let $t = \mathsf{f}(x, y)$ and $\sigma = [x/\mathsf{f}(z, y), y/x]$. The graph $G_{\sigma,t}$ is shown on the right. There is an edge from $z$ to $z$ because $z \notin \mathrm{dom}(\sigma)$, and hence, $z\sigma = z$. The nodes from $\mathcal{V}(t) = \{x, y\}$ are marked by circles.




The variable transition graph $G_{\sigma,t}$ of a pair $\langle t, \sigma \rangle$ yields a decidable, sound, and complete criterion to detect pattern terms.

Theorem 9.3.5 – Detecting Pattern Terms.

For a term $t$ and a substitution $\sigma$, $\langle t, \sigma \rangle$ is a pattern term if and only if some cycle of $G_{\sigma,t}$ (where self-loops are cycles as well) contains a variable $x$ such that $x\sigma \notin \mathcal{V}$.

*Proof.* We prove both directions individually. *Proof*

▶ *Proof*(*Soundness*, $\Longleftarrow$): We prove that the condition of Theorem 9.3.5 is sufficient. Let $x$ be a variable on a cycle of $G_{\sigma,t}$ with $x\sigma \notin \mathcal{V}$, and let $k \in \mathbb{N}_{>0}$ be the length of this cycle, so that $x \in \mathcal{V}(x\sigma^k)$. Since $x\sigma \notin \mathcal{V}$, we also have $x\sigma^k \notin \mathcal{V}$. Hence, $x$ occurs in $x\sigma^k$ at some position $\tau \neq \varepsilon$, and for every $n \in \mathbb{N}_{>0}$, the term $x\sigma^{(n-1)\cdot k}$ is a proper subterm of $x\sigma^{n\cdot k} = (x\sigma^k)\sigma^{(n-1)\cdot k}$ at position $\tau$. So the sizes of the terms $x\sigma^{n\cdot k}$ are strictly increasing in $n$. Moreover, $x$ is reachable from some $y \in \mathcal{V}(t)$ in $G_{\sigma,t}$, i.e., there is a $j \in \mathbb{N}$ with $x \in \mathcal{V}(y\sigma^j) \subseteq \mathcal{V}(t\sigma^j)$. Then $t\sigma^{j+n\cdot k}$ contains the subterm $x\sigma^{n\cdot k}$ for every $n \in \mathbb{N}$, so the sizes of the terms $t\sigma^m$ are unbounded, and the set $\{t\sigma^m \mid m \in \mathbb{N}\}$ is infinite. Now assume that $t\sigma^a = t\sigma^b$ for some $a < b$. Then $t\sigma^{a+i} = t\sigma^{b+i}$ for all $i \in \mathbb{N}$, so every term $t\sigma^m$ is contained in the finite set $\{t\sigma^0, \ldots, t\sigma^{b-1}\}$, which is a contradiction. Hence, the terms $t, t\sigma, t\sigma^2, \ldots$ are pairwise different.

▶ *Proof*(*Completeness*, $\Longrightarrow$): We prove that the condition of Theorem 9.3.5 is necessary. We assume the contrary, i.e., assume that for all variables $x$ in cycles of $G_{\sigma,t}$ we have $x\sigma \in \mathcal{V}$. This means that for every such variable $x$ there exists a $k_x > 0$ such that $x\sigma^{k_x} = x$.

Moreover, $x\sigma \in \mathcal{V}$ implies that every variable in a cycle of $G_{\sigma,t}$ only has one outgoing edge. Therefore, $t\sigma^{|\mathrm{dom}(\sigma)|}$ only contains variables on cycles of $G_{\sigma,t}$. Let $k = k_{x_1} \cdot \ldots \cdot k_{x_n}$ where $x_1, \ldots, x_n$ are all variables on cycles of $G_{\sigma,t}$. Thus, for all

$m \in \mathbb{N}$ we have $t\sigma^{|\mathrm{dom}(\sigma)|} = t\sigma^{|\mathrm{dom}(\sigma)|+m\cdot k}$, which shows that $\langle t, \sigma\rangle$ is not a pattern term. QED

Example 9.3.6. The pair $\langle t, \sigma\rangle$ with $t = \mathsf{f}(x, y)$ and $\sigma = [x/\mathsf{f}(z, y), y/x]$ is a pattern term, because $G_{\sigma,t}$ has a cycle that contains the variable $x$ and $x\sigma = \mathsf{f}(z, y) \notin \mathcal{V}$.

Next, we consider the problem of counting pattern term occurrences. Here, we directly introduce both settings where we count non-overlapping occurrences and where we only count orthogonal occurrences. In contrast to the counting of ordinary term occurrences, an occurrence of a pattern term may be counted at *any* multiplicity up to its maximal one.

Definition 9.3.7 – Pattern Occurrences.
*Let $\langle t, \sigma\rangle$ be a pattern term and let $s \in \mathcal{T}$ be a term. We say that $\langle t, \sigma\rangle$ occurs with multiplicity $q \in \mathbb{N}$ at position $\pi$ in $s$ (denoted by $\langle t, \sigma\rangle \trianglelefteq_\pi^q s$) if $t\sigma^q \trianglelefteq_\pi s$. The maximal multiplicity of an occurrence at position $\pi$ in $s$ is denoted by $m_\pi$. Two occurrences $\langle t, \sigma\rangle \trianglelefteq_{\pi_1}^{q_1} s$ and $\langle t, \sigma\rangle \trianglelefteq_{\pi_2}^{q_2} s$ with $\pi_1 \neq \pi_2$ are overlapping w.r.t. $\langle t, \sigma\rangle$ if there is a position $\tau \in \mathrm{pos}_\Sigma(t\sigma^{q_1})$ with $\pi_2 = \pi_1.\tau$ or there is a position $\tau \in \mathrm{pos}_\Sigma(t\sigma^{q_2})$ with $\pi_1 = \pi_2.\tau$.*
*The set of all sets of pairwise non-overlapping pattern occurrences of $\langle t, \sigma\rangle$ in $s$ is denoted by $\mathrm{NPO}(t, \sigma, s)$. So for a set $S$ of pairs $(\pi, q_\pi)$ of positions with multiplicities, where the positions are pairwise different, we have $S \in \mathrm{NPO}(t, \sigma, s)$ iff*

- *if $(\pi, q_\pi) \in S$ then $\langle t, \sigma\rangle \trianglelefteq_\pi^{q_\pi} s$, and*
- *if $(\pi_1, q_{\pi_1}), (\pi_2, q_{\pi_2}) \in S$ with $\pi_1 \neq \pi_2$, then the corresponding occurrences are non-overlapping w.r.t. $\langle t, \sigma\rangle$.*

*For every set $S \in \mathrm{NPO}(t, \sigma, s)$, let $m_S = \sum_{(\pi,q_\pi)\in S} q_\pi$. Let $\mathrm{maxNPO}(t, \sigma, s) = \max\{m_S \mid S \in \mathrm{NPO}(t, \sigma, s)\}$ denote the maximal value that one can obtain by adding the multiplicities for a set $S$ of pairwise non-overlapping occurrences of $\langle t, \sigma\rangle$ in $s$. For orthogonal pattern occurrences we define $\mathrm{maxOPO}(t, \sigma, s) = \max\{m_S \mid S \in \mathrm{NPO}(t, \sigma, s), \forall(\pi_1, q_{\pi_1}), (\pi_2, q_{\pi_2}) \in S.\ \pi_1 \neq \pi_2 \Longrightarrow \pi_1 \perp \pi_2\}$.*

Example 9.3.8. As an example, consider the term $t = \mathsf{f}(x)$, the substitution $\sigma = [x/\mathsf{g}(x)]$, and $s = \mathsf{c}(\mathsf{f}(\mathsf{g}(x)), \mathsf{f}(\mathsf{g}(\mathsf{g}(x))))$. Then $\langle t, \sigma\rangle$ occurs in $s$ at position 1 with maximal multiplicity $m_1 = 1$, and at position 2 with maximal multiplicity $m_2 = 2$. Since both occurrences are at orthogonal positions, it is optimal to count both of them at their maximal multiplicities, i.e., for $S = \{(1, 1), (2, 2)\} \in \mathrm{NPO}(t, \sigma, s)$ we have $\mathrm{maxNPO}(t, \sigma, s) = \mathrm{maxOPO}(t, \sigma, s) = m_S = 1 + 2 = 3$.

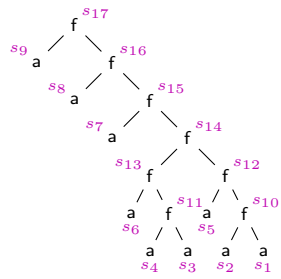
To compute $\mathrm{maxOPO}(t, \sigma, s)$, we use Algorithm 8 with $t\sigma$ instead of $t$. Moreover, we adjust the computation of $\alpha_{s'}$ for inner nodes $s'$ in Algorithm 8 in the case of $t\sigma \trianglelefteq_\varepsilon s'$. Instead of setting $\alpha_{s'}$ to $\max\{\beta_{s'}, 1\}$ as in the computation of $\mathrm{maxOO}(t\sigma, s)$, we set $\alpha_{s'}$ to $\max\{\beta_{s'}, m\}$ where $m$ is the maximal multiplicity of the occurrence of $\langle t, \sigma\rangle$ at the root of $s'$ (for orthogonal positions, counting at the maximal multiplicity is always optimal). Note that we only consider orthogonal occurrences here. So if we consider the occurrence at the root of $s'$, then occurrences below the root are ignored. To compute $\mathrm{maxNPO}(t, \sigma, s)$ we set $\alpha_{s'}$ to the maximum of $\beta_{s'}$ and $\sum_{\pi\in\mathrm{pos}_\mathcal{V}(t\sigma^q)} \alpha_{s'|_\pi} + q$ for all $1 \leq q \leq m$, since here it can be beneficial to count the occurrence at the root of $s'$ at a non-maximal multiplicity. To find $m$, we compute $t\sigma$, $t\sigma^2$, ... as long as $|t\sigma^q| \leq |s'|$

holds, and we check for each of these terms whether it matches $s'$. This requires at most $|\mathrm{dom}(\sigma)| \cdot |s'|$ matching checks, since the size of $t\sigma^q$ never decreases and increases at least once every $|\mathrm{dom}(\sigma)|$ steps. This yields a runtime of $\mathcal{O}(|\mathrm{dom}(\sigma)| \cdot |s|^3)$, i.e., $\mathcal{O}(|s|^3)$ for a fixed pattern term.

Example 9.3.9 (Computing $\mathrm{maxOPO}(t', \sigma, s)$). Consider the pattern term $\langle t', \sigma \rangle$ with base term $t' = \mathsf{f}(\mathsf{a}, x)$ and pumping substitution $\sigma = [x/\mathsf{f}(\mathsf{a}, x)]$, and let $t = \mathsf{f}(\mathsf{a}, \mathsf{f}(\mathsf{a}, x))$ and $s$ be as in Example 9.2.5. Note that $t = \mathsf{f}(\mathsf{a}, x)[x/\mathsf{f}(\mathsf{a}, x)]$. We have occurrences of $\langle t', \sigma \rangle$ with multiplicity 0 at $s_{10}$, $s_{11}$, and $s_{15}$; with multiplicity 1 at $s_{12}$, $s_{13}$, and $s_{16}$; and with multiplicity 2 at the root $s_{17}$. Since we only consider occurrences at orthogonal positions, the maximum is obtained by adding the multiplicities for the orthogonal subterms $s_{12}$ and $s_{13}$ or by considering the multiplicity at the root. Thus, we obtain $\mathrm{maxOPO}(t', \sigma, s) = \alpha_s = \alpha_{s_{17}} = 2$. The full table is shown in Figure 9.12.

| Subterm $s'$ | $\alpha_{s'}$ | $\beta_{s'}$ |
|---|---|---|
| $s_1$–$s_{11}$ | 0 | 0 |
| $s_{12}$–$s_{13}$ | 1 | 0 |
| $s_{14}$ | 2 | 2 |
| $s_{15}$ | 2 | 2 |
| $s_{16}$ | 2 | 2 |
| $s_{17}$ | 2 | 2 |

(a) Dynamic programming table.




(b) Tree structure of $s$.



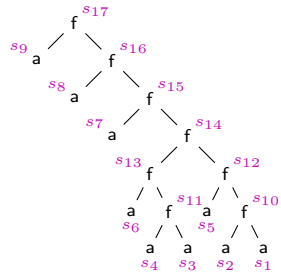

(c) Tree structure of $t$.

Figure 9.12: Computation of $\mathrm{maxOPO}(t', \sigma, s)$.

Example 9.3.10 (Computing $\mathrm{maxNPO}(t', \sigma, s)$). Consider the same terms and substitution as in Example 9.3.9, but let us compute $\mathrm{maxNPO}(t', \sigma, s)$ instead of $\mathrm{maxOPO}(t', \sigma, s)$. The only difference between the computation of $\mathrm{maxNPO}(t', \sigma, s)$ and $\mathrm{maxOPO}(t', \sigma, s)$ is at the root of $s$. While previously we could only use the occurrence at the root or the two orthogonal occurrences below the root, we can now consider all of them simultaneously: the two occurrences at $s_{12}$ and $s_{13}$ do not overlap with the occurrence of $\mathsf{f}(\mathsf{a}, x)[x/\mathsf{f}(\mathsf{a}, x)]^2$ at the root. Therefore, we get $\mathrm{maxNPO}(t', \sigma, s) = \alpha_s = \alpha_{s_{17}} = 2 + 1 + 1 = 4$. The full table is shown in Figure 9.13.

| Subterm $s'$ | $\alpha_{s'}$ | $\beta_{s'}$ |
|---|---|---|
| $s_1$–$s_{11}$ | 0 | 0 |
| $s_{12}$–$s_{13}$ | 1 | 0 |
| $s_{14}$ | 2 | 2 |
| $s_{15}$ | 2 | 2 |
| $s_{16}$ | 3 | 2 |
| $s_{17}$ | 4 | 3 |

(a) Dynamic programming table.




(b) Tree structure of $s$.




(c) Tree structure of $t$.

Figure 9.13: Computation of $\mathrm{maxNPO}(t', \sigma, s)$.

As demonstrated by the PTRS $\mathcal{P}'_{18}$ in Counterexample 9.3.1, if $t\sigma$ can be rewritten to a term without any occurrence of $t$, then the corresponding rewrite step of $t\sigma^m$ does not necessarily reduce the multiplicity by 1, but it may directly lead to a normal form.

Hence, this pattern term cannot be used to disprove PAST. So a pattern term $\langle t, \sigma \rangle$ may only be used for disproving PAST if we have a $\to_{\mathcal{P}}$-RT where every leaf contains an occurrence of $t$. Moreover, if we have a $\to_{\mathcal{P}}$-RT $\mathfrak{T}$ that starts with $t\sigma^1$ and has an occurrence $\langle t, \sigma \rangle \blacktriangleleft_\pi^q t_v$ in a leaf $v$, then we also need that the tree can be "generalized" from 1 to an arbitrary multiplicity $m \in \mathbb{N}_{>0}$, i.e., rewriting the term $t\sigma^m$ using the same rules as in $\mathfrak{T}$ at the same positions must result in a leaf $v'$ with an occurrence $\langle t, \sigma \rangle \blacktriangleleft_\pi^{q+m-1} t_{v'}$. For an occurrence $\langle t, \sigma \rangle \blacktriangleleft_\pi^q s$ in a term $s$, every substitution $\kappa$ with $t\sigma^q\kappa = s|_\pi$ is called a *closing substitution* of the occurrence (note that $\kappa$ is only unique on the variables of $t\sigma^q$). We call the occurrence *commuting* if it has a closing substitution $\kappa$ that commutes with the pumping substitution $\sigma$, i.e., $\sigma\kappa = \kappa\sigma$. Then we ensure this generalization property by requiring that every leaf contains at least one commuting occurrence, and by only counting sets of occurrences that are *anchored* by such a commuting occurrence.[2] A sufficient condition for an occurrence to be commuting that can be checked automatically is that its closing substitution $\kappa$ with $\mathrm{dom}(\kappa) \subseteq \mathcal{V}(t\sigma^q)$ commutes with $\sigma$.

Definition 9.3.11 – Pattern Tree.
*Let $\mathcal{P}$ be a PTRS, let $\mathfrak{T}$ be a $\to_{\mathcal{P}}$-RT, and let $\langle t, \sigma \rangle$ be a pattern term. $\mathfrak{T}$ is a $\mathcal{P}$-pattern tree (or $\mathcal{P}$-PT) for $\langle t, \sigma \rangle$ if $\mathrm{root}(\mathfrak{T}) = t\sigma$ and for every leaf $v \in \mathrm{Leaf}(\mathfrak{T})$ there exist $\pi \in \mathrm{pos}(t_v)$ and $q \in \mathbb{N}$ such that $\langle t, \sigma \rangle \blacktriangleleft_\pi^q t_v$ is a commuting occurrence.*

Example 9.3.12. The $\to_{\mathcal{P}_{1s}}$-RT in Figure 9.11 is a pattern tree for the pattern term $\langle t, \sigma \rangle$ where $t = \mathsf{f}(x)$ and $\sigma = [x/\mathsf{g}(x)]$. Here, the occurrences at the roots of the two leaves $v_1$ and $v_2$ (with the multiplicities 0 and 2) are commuting, since the identity is a closing substitution of both of them, and it commutes with $\sigma$. Indeed, this tree can be "generalized" to arbitrary multiplicities $m > 0$, because the corresponding $\to_{\mathcal{P}_{1s}}$-RT starting with $\mathsf{f}(\mathsf{g}^m(x))$ at the root has $\mathsf{f}(\mathsf{g}^{m-1}(x))$ and $\mathsf{f}(\mathsf{g}^{2+m-1}(x))$ in its leaves.

For the counting in pattern trees, we restrict ourselves to occurrence sets that contain a commuting occurrence as an *anchor*.

Definition 9.3.13 – Anchored Occurrence Sets.
*Let $\langle t, \sigma \rangle$ be a pattern term and let $s \in \mathcal{T}$. A set $S \in \mathrm{NPO}(t, \sigma, s)$ is anchored if it contains a commuting occurrence $(\chi, q) \in S$ such that there is no $(\pi', q') \in S$ with $\chi < \pi'$, i.e., no other occurrence of $S$ lies strictly below the anchor. We define $\mathrm{maxNPO}^{\mathrm{anc}}(t, \sigma, s) = \max\{m_S \mid S \in \mathrm{NPO}(t, \sigma, s), S \text{ is anchored}\}$ and $\mathrm{maxOPO}^{\mathrm{anc}}(t, \sigma, s) = \max\{m_S \mid S \in \mathrm{NPO}(t, \sigma, s), S \text{ is anchored}, \forall (\pi_1, q_{\pi_1}), (\pi_2, q_{\pi_2}) \in S.\ \pi_1 \neq \pi_2 \implies \pi_1 \perp \pi_2\}$, where $\max \varnothing = 0$.*

Let us also illustrate the difference between anchored and non-anchored sets.

Example 9.3.14. Consider the pattern term $\langle t', \sigma \rangle$ with base term $t' = \mathsf{f}(y, x)$ and pumping substitution $\sigma = [y/\mathsf{g}(y)]$, and consider the term $s = \mathsf{f}(\mathsf{g}(\mathsf{g}(y)), \mathsf{f}(\mathsf{g}(y), x))$. The pattern term $\langle t', \sigma \rangle$ occurs in $s$ at two positions. The occurrence at the root has multiplicity 2, since $t'\sigma^2\kappa = s$ holds for the closing substitution $\kappa = [x/\mathsf{f}(\mathsf{g}(y), x)]$.

[2] This differs from [KNSG26], where $\sigma$ has to commute with the closing substitutions of *all* occurrences in all leaves (so that every set of occurrences is anchored). We use the weaker requirement because of the newly added results on linear and NVD pattern trees (Theorems 9.3.30 and 9.3.35), where natural examples like Examples 9.3.31 and 9.3.36 contain non-commuting occurrences. Moreover, the weaker variant is the one realized in our implementation in AProVE.

This occurrence is not commuting, since every closing substitution $\kappa$ of it satisfies $x(\sigma\kappa) = x\kappa = \mathsf{f}(\mathsf{g}(y), x)$ but $x(\kappa\sigma) = \mathsf{f}(\mathsf{g}(\mathsf{g}(y)), x)$. In contrast, the occurrence at position 2 with multiplicity 1 is commuting, since the identity is a closing substitution of it. The two occurrences are non-overlapping, as position 2 is the position of the variable $x$ in $t'\sigma^2$, but they are not orthogonal. Hence, the set $\{(\varepsilon, 2), (2, 1)\}$ of occurrences is anchored via the commuting occurrence $(2, 1)$, below which there is no further occurrence of the set, and we have $\mathrm{maxNPO}^{\mathrm{anc}}(t', \sigma, s) = 2 + 1 = 3$. For orthogonal occurrences, however, the set $\{(\varepsilon, 2)\}$ may not be counted, since it is not anchored, and $\{(2, 1)\}$ is the anchored set of occurrences at pairwise orthogonal positions with the maximal value. Thus, we only have $\mathrm{maxOPO}^{\mathrm{anc}}(t', \sigma, s) = 1$, whereas $\mathrm{maxOPO}(t', \sigma, s) = 2$.

The anchored maxima can be computed by a variant of the dynamic programming algorithm above that additionally computes, for every node $s'$, the maximal value of an anchored set of occurrences in $s'$. If no occurrence at the root of $s'$ is counted, then the anchor has to lie in one of the children of $s'$. Otherwise, the occurrence at the root of $s'$ can either be the anchor itself (if it is commuting), and then no occurrence below it may be counted, or, for non-overlapping sets, it can be combined with an anchored set in one of the subterms at its variable positions.

The anchor of a set can be pumped, i.e., its multiplicity can be increased, without causing overlaps with the other occurrences of the set.

Lemma 9.3.15 – Pumping the Anchor.

*Let $\langle t, \sigma\rangle$ be a pattern term, let $s \in \mathcal{T}$, let $S \in \mathrm{NPO}(t, \sigma, s)$ be anchored by the commuting occurrence $(\chi, r) \in S$, and finally, let $n \in \mathbb{N}$. Then, the set $S' = (S \setminus \{(\chi, r)\}) \cup \{(\chi, r+n)\}$ satisfies $S' \in \mathrm{NPO}(t, \sigma, s\sigma^n\theta)$ for every substitution $\theta$, and $m_{S'} = m_S + n$. If the positions of $S$ are pairwise orthogonal, then so are the positions of $S'$.*

*Proof.* Let $\kappa$ be a closing substitution of $(\chi, r)$ with $\sigma\kappa = \kappa\sigma$. Then $(s\sigma^n\theta)|_\chi = t\sigma^r\kappa\sigma^n\theta = t\sigma^{r+n}\kappa\theta$, so $(\chi, r + n)$ is an occurrence in $s\sigma^n\theta$. Every other $(\pi, q) \in S$ with a closing substitution $\kappa'$ is an occurrence in $s\sigma^n\theta$ with the closing substitution $\kappa'\sigma^n\theta$. So all positions and all multiplicities except the one of the anchor are unchanged, which yields $m_{S'} = m_S + n$. Whether two occurrences overlap only depends on their positions and multiplicities, so it remains to consider pairs of the new anchor $(\chi, r+n)$ and some $(\pi, q) \in S$ with $\pi \neq \chi$. As $S$ is anchored by $(\chi, r)$, $\pi$ does not lie strictly below $\chi$. If $\pi$ is orthogonal to $\chi$, then the two occurrences are non-overlapping. Otherwise, $\pi$ lies strictly above $\chi$, and then the two occurrences are non-overlapping if and only if there is no $\tau \in \mathrm{pos}_\Sigma(t\sigma^q)$ with $\chi = \pi.\tau$, which does not depend on the multiplicity of the anchor. Finally, $S$ and $S'$ have the same positions, so orthogonality is preserved. *Proof* *QED*

## 9.3.1 Counting Orthogonal Pattern Occurrences

Similar to Theorem 9.2.15, by rewriting and counting orthogonal occurrences, we obtain a technique to embed random walks via patterns in a $\to_{\mathcal{P}}$-RT.

**Theorem 9.3.16** – Embedding RWs via Patterns.

Let $\mathcal{P}$ be a PTRS, $\langle t, \sigma\rangle$ be a pattern term, and let $\mathfrak{T}$ be a finite $\mathcal{P}$-PT for $\langle t, \sigma\rangle$ with $\mathrm{h}(\mathfrak{T}) > 0$.

- If we have $\sum_{v\in\mathrm{Leaf}(\mathfrak{T})} p_v \cdot \mathrm{maxOPO}^{\mathrm{anc}}(t,\sigma,t_v) > 1$, then $\mathcal{P}$ is not $\mathtt{AST}$.
- If we have $\sum_{v\in\mathrm{Leaf}(\mathfrak{T})} p_v \cdot \mathrm{maxOPO}^{\mathrm{anc}}(t,\sigma,t_v) \geq 1$, then $\mathcal{P}$ is not $\mathtt{PAST}$.

*Proof.* The construction of the infinite $\to_{\mathcal{P}}$-RT $\mathfrak{T}^\infty$, of the random walk program $\mu$, and of the embedding follows the proof of Theorem 9.2.6, where we now define $\mu$ via $\mathrm{maxOPO}^{\mathrm{anc}}(t,\sigma,t_v)$ instead of $\mathrm{maxNO}(t,t_v)$. The only difference is that for the terms $s$ of the nodes of $\mathfrak{T}^\infty$, we use the maximum $\mathrm{maxOPO}(t,\sigma,s)$ over *all* (not necessarily anchored) sets of occurrences at pairwise orthogonal positions. So we show $a_w \leq \mathrm{maxOPO}(t,\sigma,t_{\mathtt{e}(w)})$ for every node $w$ of the $\hookrightarrow_\mu$-RT $\mathfrak{T}_1$ that is labeled with the number $a_w$. For the root $w$ of $\mathfrak{T}_1$, we have $a_w = 1 \leq \mathrm{maxOPO}(t,\sigma,t\sigma)$, since $(\varepsilon, 1)$ is an occurrence of $\langle t,\sigma\rangle$ in $t\sigma$.
In $\mathfrak{T}^\infty$, a node with a term $s$ such that $\mathrm{maxOPO}(t,\sigma,s) = k > 0$ is extended as follows. Let $W$ be a set of occurrences at pairwise orthogonal positions in $s$ with $m_W = k$. Omitting occurrences with multiplicity 0 does not change $m_W$, so we may assume that all chosen multiplicities in $W$ are positive. We choose an occurrence $(\pi_1, q_1) \in W$, which does not have to be a commuting one. Let $\delta$ be a closing substitution of $(\pi_1, q_1)$, i.e., $s = C[t\sigma^{q_1}\delta]$ for the context $C$ with the hole at position $\pi_1$. Since $t\sigma^{q_1}\delta = (t\sigma)\sigma^{q_1-1}\delta$ and rewriting is closed under substitutions and contexts, we can rewrite $s$ at the positions $\pi_1.\tau$ using the same rules as $\mathfrak{T}$ at the positions $\tau$. This yields a tree $\mathfrak{T}_s$ and a bijective embedding $\mathtt{e}_s : N^{\mathfrak{T}} \to N^{\mathfrak{T}_s}$ with $t_{\mathtt{e}_s(v)} = C[t_v\sigma^{q_1-1}\delta]$ for every leaf $v \in \mathrm{Leaf}(\mathfrak{T})$. As in the proof of Theorem 9.2.6, it remains to prove

$$\mathrm{maxOPO}(t,\sigma,t_{\mathtt{e}_s(v)}) \geq \mathrm{maxOPO}^{\mathrm{anc}}(t,\sigma,t_v) + k - 1.$$

All occurrences of $W' = W \setminus \{(\pi_1, q_1)\}$ are orthogonal to $\pi_1$, so they are still present in $t_{\mathtt{e}_s(v)}$ with unchanged positions and multiplicities, and we have $m_{W'} = k - q_1$. Since $\mathfrak{T}$ is a pattern tree, $t_v$ contains a commuting occurrence $(\chi, r)$, and the set $\{(\chi, r)\}$ is anchored. Hence, there is an anchored set $S$ of occurrences at pairwise orthogonal positions in $t_v$ with $m_S = \mathrm{maxOPO}^{\mathrm{anc}}(t,\sigma,t_v)$. By Lemma 9.3.15 (with $n = q_1 - 1$ and $\theta = \delta$), there is a set $S'$ of occurrences at pairwise orthogonal positions in $t_v\sigma^{q_1-1}\delta$ with $m_{S'} = m_S + q_1 - 1$. So $\{(\pi_1.\chi, r) \mid (\chi, r) \in S'\}$ is a set of occurrences in $t_{\mathtt{e}_s(v)} = C[t_v\sigma^{q_1-1}\delta]$ whose positions are pairwise orthogonal and below $\pi_1$, i.e., orthogonal to all positions of $W'$. Together with $W'$, we obtain

$$\begin{aligned} &\mathrm{maxOPO}(t,\sigma,t_{\mathtt{e}_s(v)}) \\ \geq\ & m_{W'} + m_{S'} \\ =\ & (k - q_1) + \mathrm{maxOPO}^{\mathrm{anc}}(t,\sigma,t_v) + q_1 - 1 \\ =\ & \mathrm{maxOPO}^{\mathrm{anc}}(t,\sigma,t_v) + k - 1. \end{aligned}$$

*QED*

Example 9.3.17. Using Theorem 9.3.16, we can now embed the positively drifting random walk from Figure 9.10 with $\mu_3(-1) = 1/3$ and $\mu_3(1) = 2/3$ in the $\to_{\mathcal{P}_{18}}$-RT of Figure 9.11 corresponding to the only rule $\mathtt{f}(\mathtt{g}(x)) \to \{{}^{1/3}\mathtt{f}(x), {}^{2/3}\mathtt{f}(\mathtt{g}(\mathtt{g}(x)))\}$ of $\mathcal{P}_{18}$ and disprove $\mathtt{AST}(\to_{\mathcal{P}_{18}})$.

Example 9.3.18. By counting the number of applications of the pumping substitution $\sigma = [x/\mathtt{g}(y), y/\mathtt{f}(x)]$ to the base term $t = \mathtt{c}(y, x)$, we can disprove $\mathtt{AST}$ of the PTRS $\mathcal{P}_{19}$

with the only rule $\mathsf{c}(\mathsf{f}(x), \mathsf{g}(y)) \to \{{}^{1}/_{3}\mathsf{c}(y, x), {}^{2}/_{3}\mathsf{c}(\mathsf{f}(\mathsf{g}(y)), \mathsf{g}(\mathsf{f}(x)))\}$ via Theorem 9.3.16. Again, we can embed the random walk from Figure 9.10 in the RT corresponding to the only rewrite rule, because the child $t = \mathsf{c}(y, x)$ has the probability ${}^{1}/_{3}$, and the second child $t\sigma^2 = \mathsf{c}(\mathsf{f}(\mathsf{g}(y)), \mathsf{g}(\mathsf{f}(x)))$ has the probability ${}^{2}/_{3}$.

To automate Theorem 9.3.16, we adapt our implementation of Theorem 9.2.15. After finding the pattern $t\sigma$, we have to rewrite the leaves until we obtain a pattern tree. When computing the values for the leaves, our implementation only considers anchored sets of occurrences, i.e., it computes $\mathrm{maxNPO}^{\mathrm{anc}}$ and $\mathrm{maxOPO}^{\mathrm{anc}}$. Note that we can directly stop if our reachability analysis shows that some leaf cannot reach a term containing an occurrence of $t$.

### 9.3.2 Counting Linear Pattern Term Occurrences

If we want to count non-overlapping occurrences instead of just orthogonal ones, then we have to restrict ourselves to linear pattern terms or NVD pattern trees. However, linearity and the NVD condition now concern infinitely many instances $t$, $t\sigma$, $t\sigma^2$, etc., for pattern terms and pattern trees. In this subsection, we first consider the problem of checking for linearity.

Definition 9.3.19 – Linear Pattern Term.
*A pattern term $\langle t, \sigma\rangle$ is linear if $t\sigma^m$ is a linear term for all $m \in \mathbb{N}$.*

Example 9.3.20. The pattern term $\langle \mathsf{f}(x), [x/\mathsf{g}(x)]\rangle$ from Counterexample 9.3.1 is linear. Every pumped term $\mathsf{f}(x)[x/\mathsf{g}(x)]^m = \mathsf{f}(\mathsf{g}^m(x))$ contains $x$ exactly once.

Example 9.3.21. Consider $t = \mathsf{f}(x, y)$ and $\sigma = [x/\mathsf{g}(x, y)]$ with $\mathrm{dom}(\sigma) = \{x\}$. The pair $\langle t, \sigma\rangle$ is a pattern term: $G_{\sigma,t}$ has a self-loop at $x$ and $x\sigma = \mathsf{g}(x, y) \notin \mathcal{V}$, so the condition of Theorem 9.3.5 is satisfied. However, $\langle t, \sigma\rangle$ is *not* linear. Already $t\sigma = \mathsf{f}(\mathsf{g}(x, y), y)$ is non-linear, as $y$ appears at position 1.2 (inside the image of $x$) and at position 2 (the original occurrence in $t$, since $y \notin \mathrm{dom}(\sigma)$).

To check linearity automatically, we characterize it via three conditions on $t$ and $\sigma$. For readability, we first present a criterion that is only sound (and incomplete), but illustrates the core idea.

Lemma 9.3.22 – Detecting Linear Pattern Terms (Sound).
*Let $\langle t, \sigma\rangle$ be a pattern term. Then $\langle t, \sigma\rangle$ is linear if the following three conditions hold:*

(i) *$t$ is linear,*
(ii) *$x\sigma$ is linear for all $x \in \mathcal{V}$,*
(iii) *$x, y \in \mathcal{V},\ x \neq y \implies \mathcal{V}(x\sigma) \cap \mathcal{V}(y\sigma) = \varnothing$.*

*Proof.* We show by induction on $m$ that $t\sigma^m$ is linear for all $m \in \mathbb{N}$. For $m = 0$ this is Condition (i). Assume $t\sigma^m$ is linear. To get $t\sigma^{m+1}$, every variable $x \in \mathrm{dom}(\sigma)$ appearing in $t\sigma^m$ is replaced by $x\sigma$. Condition (ii) ensures that $x\sigma$ itself is linear, so no duplicates arise within a single image. Condition (iii) ensures that no variable $z$ is contributed by two different sources: for two distinct $x, y \in \mathcal{V}$ both appearing in $t\sigma^m$, the images $x\sigma$ and $y\sigma$ share no variable, so replacements at different positions do not

Proof

create duplicates. Hence, $t\sigma^{m+1}$ is linear. QED

Example 9.3.23. Consider the pattern term $\langle t, \sigma \rangle$ with $t = \mathsf{f}(x, y)$ and $\sigma = [x/\mathsf{g}(x),\ y/\mathsf{h}(y)]$ with $\mathrm{dom}(\sigma) = \{x, y\} = \mathcal{V}(t)$. All three conditions of Lemma 9.3.22 hold: $\mathsf{f}(x, y)$ is linear (i); $x\sigma = \mathsf{g}(x)$ and $y\sigma = \mathsf{h}(y)$ are linear (ii); and for all $z \in \mathcal{V}$ we have $\mathcal{V}(z\sigma) = \{z\}$, so (iii) holds as well. Hence, $\langle t, \sigma \rangle$ is linear, which is also apparent from the pumped terms $t\sigma^m = \mathsf{f}(\mathsf{g}^m(x), \mathsf{h}^m(y))$, each containing $x$ and $y$ exactly once.

Example 9.3.24. Consider the pattern term $\langle t, \sigma \rangle$ with $t = \mathsf{f}(x, y)$ and $\sigma = [x/z,\ \ y/z,\ \ z/\mathsf{g}(z)]$ with $\mathrm{dom}(\sigma) = \{x, y, z\}$, $\mathcal{V}(t) = \{x, y\}$, and $z \notin \mathcal{V}(t)$. Conditions (i)–(ii) hold, but Condition (iii) fails: $\mathcal{V}(x\sigma) \cap \mathcal{V}(y\sigma) = \{z\} \cap \{z\} = \{z\} \neq \varnothing$. Indeed, $t\sigma^1 = \mathsf{f}(z, z)$ is already non-linear, as $z$ appears at positions 1 and 2. Hence, $\langle t, \sigma \rangle$ is not linear.

In order to find a decidable, sound criterion that is also complete, we have to consider the variable transition graph $G_{\sigma,t}$ again. In fact, a variable $y \in \mathcal{V}$ only needs to satisfy the pairwise disjointness requirement with $x \in \mathcal{V}$ if $x$ and $y$ can be *simultaneously active* in some $t\sigma^m$, i.e., if $x, y \in \mathcal{V}(t\sigma^m)$. We call such a pair of variables $(x, y)$ *concurrent*. The sound and complete criterion requires checking Condition (iii) for all concurrent pairs.

Example 9.3.25. Consider the pattern term $\langle t, \sigma \rangle$ with term $t = \mathsf{f}(x)$ and substitution $\sigma = [x/\mathsf{g}(y),\ y/\mathsf{h}(y, z),\ z/\mathsf{k}(w)]$. The graph $G_{\sigma,t}$ is shown in Figure 9.14. Tracing the active variable set $S_m$ (the set of variables present in $t\sigma^m$) reveals the non-linearity. At each step, every active variable $x$ is replaced by the variables of $x\sigma$:

$$S_0 = \{x\}, \quad S_1 = \{y\}, \quad S_2 = \{y, z\}, \quad S_3 = \{y, z, w\}.$$

At step 4, applying $\sigma$ to each active variable gives $y \mapsto \{y, z\}$, $z \mapsto \{w\}$, $w \mapsto \{w\}$, so

$$S_4 = \{y, z\} \cup \{w\} \cup \{w\} = \{y, z, w\}.$$

Linearity is disproven by the *concurrent* pair $(z, w)$, both present in $S_3$, with $\mathcal{V}(z\sigma) \cap \mathcal{V}(w\sigma) = \{w\} \cap \{w\} = \{w\} \neq \varnothing$. Indeed, $t\sigma^4 = \mathsf{f}(\mathsf{g}(\mathsf{h}(\mathsf{h}(\mathsf{h}(y, z), \mathsf{k}(w)), \mathsf{k}(w))))$ has $w$ at two distinct positions, and thus, $\langle t, \sigma \rangle$ is not linear.

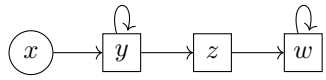



Figure 9.14: Graph $G_{\sigma,t}$ for $t = \mathsf{f}(x)$ and $\sigma = [x/\mathsf{g}(y), y/\mathsf{h}(y, z), z/\mathsf{k}(w)]$.

Definition 9.3.26 – Active Variable Set and Concurrent Pairs.
*For a pattern term $\langle t, \sigma \rangle$, the active variable sets $S_m$ for $m \in \mathbb{N}$ are defined by*

$$S_0 = \mathcal{V}(t) \quad \text{and} \quad S_{m+1} = \bigcup_{x \,\in\, S_m} \mathcal{V}(x\sigma).$$

*Two distinct variables $x, y$ are concurrent* w.r.t. *$\langle t, \sigma \rangle$ if $x, y \in S_m$ for some $m \in \mathbb{N}$.*

Since $\mathcal{V}(u\sigma) = \bigcup_{x \in \mathcal{V}(u)} \mathcal{V}(x\sigma)$ holds for every term $u$, we have $S_m = \mathcal{V}(t\sigma^m)$ for all $m \in \mathbb{N}$ by induction on $m$. Moreover, a variable is reachable from $\mathcal{V}(t)$ in $G_{\sigma,t}$ if and only if it is contained in some $S_m$.

The set $C$ of all concurrent pairs is the least set of pairs of distinct variables such that

(a) $(x, y) \in C$ for all distinct $x, y \in \mathcal{V}(t)$,
(b) $(x', y') \in C$ for all distinct $x', y' \in \mathcal{V}(x\sigma)$, where $x$ is reachable from $\mathcal{V}(t)$ in $G_{\sigma,t}$, and
(c) $(x', y') \in C$ for all $(x, y) \in C$, $x' \in \mathcal{V}(x\sigma)$, and $y' \in \mathcal{V}(y\sigma)$ with $x' \neq y'$.

Indeed, (a)–(c) only add pairs of variables that lie in a common set $S_m$, and conversely, by induction on $m$, every pair of distinct variables in $S_{m+1}$ is added by (b) (if both stem from the same variable of $S_m$) or by (c) (otherwise). Hence, $C$ can be computed by a fixpoint iteration that adds pairs according to (a)–(c) until no new pair is added, which terminates after at most $n^2$ iterations, where $n$ is the (finite) number of variables reachable from $\mathcal{V}(t)$ in $G_{\sigma,t}$.

Example 9.3.27. We compute the concurrent pairs $C$ for $\langle \mathsf{f}(x), \sigma \rangle$ from Example 9.3.25. Again, we have

$$S_0 = \{x\}, \quad S_1 = \{y\}, \quad S_2 = \{y, z\}, \quad S_3 = \{y, z, w\}, \quad S_4 = \{y, z, w\}, \quad \dots$$

where $S_m = S_3$ for all $m \geq 3$, since $S_4 = S_3$. Therefore, the set of all concurrent pairs is $C = \{(y, z), (y, w), (z, w)\}$. The fixpoint iteration yields the same set: (a) does not add any pair, (b) adds $(y, z)$ for the reachable variable $y$, and (c) adds $(y, w)$ and $(z, w)$ for the pair $(y, z)$. Checking Condition (iii) of Lemma 9.3.28 for each pair yields

- $\mathcal{V}(y\sigma) \cap \mathcal{V}(z\sigma) = \{y, z\} \cap \{w\} = \varnothing$,
- $\mathcal{V}(y\sigma) \cap \mathcal{V}(w\sigma) = \{y, z\} \cap \{w\} = \varnothing$, but
- $\mathcal{V}(z\sigma) \cap \mathcal{V}(w\sigma) = \{w\} \cap \{w\} = \{w\} \neq \varnothing$.

The pair $(z, w)$ correctly witnesses non-linearity.

In addition to restricting (iii) to concurrent pairs, we only have to require (ii) for those variables that are eventually active, i.e., reachable from $\mathcal{V}(t)$ in $G_{\sigma,t}$. This then yields the desired sound and complete criterion to check for linearity.

Lemma 9.3.28 – Detecting Linear Pattern Terms (Sound and Complete).
*Let $\langle t, \sigma \rangle$ be a pattern term. Then $\langle t, \sigma \rangle$ is linear if and only if the following hold:*

(i) *$t$ is linear,*
(ii) *$x\sigma$ is linear for all $x \in \mathrm{dom}(\sigma)$ reachable from $\mathcal{V}(t)$ in $G_{\sigma,t}$,*
(iii) *$(x, y)$ concurrent w.r.t. $\langle t, \sigma \rangle$ and $x \neq y \implies \mathcal{V}(x\sigma) \cap \mathcal{V}(y\sigma) = \varnothing$.*

*Proof.* We prove both directions individually. *Proof*

► *Proof*(*Soundness*, $\Longleftarrow$): We show by induction on $m$ that $t\sigma^m$ is linear for all $m \in \mathbb{N}$. For $m = 0$ this is Condition (i). Assume that $t\sigma^m$ is linear. Every variable $x \in \mathcal{V}(t\sigma^m) = S_m$ is reachable from $\mathcal{V}(t)$ in $G_{\sigma,t}$, so $x\sigma$ is linear by Condition (ii) (if $x \notin \mathrm{dom}(\sigma)$, then $x\sigma = x$ is linear anyway). Moreover, any two distinct variables $x, y \in S_m$ are concurrent, so $\mathcal{V}(x\sigma) \cap \mathcal{V}(y\sigma) = \varnothing$ by Condition (iii). As in the proof of Lemma 9.3.22, this implies that $t\sigma^{m+1}$ is linear.

► *Proof*(*Completeness*, $\Longrightarrow$): We show that if any condition fails, $\langle t, \sigma \rangle$ is not linear.

Note that $|s\sigma|_z = \sum_{w \in \mathcal{V}} |s|_w \cdot |w\sigma|_z$ holds for any term $s$ and variable $z$.
*Condition (i)*: If Condition (i) fails, then $t = t\sigma^0$ is already non-linear.
*Condition (ii)*: If Condition (ii) fails, there exists $x \in \mathrm{dom}(\sigma)$ reachable from $\mathcal{V}(t)$ in $G_{\sigma,t}$ such that $x\sigma$ is not linear, i.e., $|x\sigma|_z \geq 2$ for some variable $z$. Since $x$ is reachable, $x \in S_m$ for some $m$, i.e., $|t\sigma^m|_x \geq 1$. Then $|t\sigma^{m+1}|_z \geq |t\sigma^m|_x \cdot |x\sigma|_z \geq 1 \cdot 2 = 2$, so $t\sigma^{m+1}$ is not linear.
*Condition (iii)*: If Condition (iii) fails, there exists a concurrent pair $x \neq y$ with $z \in \mathcal{V}(x\sigma) \cap \mathcal{V}(y\sigma)$ for some variable $z$. Let $m$ be the first step at which both $x$ and $y$ are active, i.e., $|t\sigma^m|_x \geq 1$ and $|t\sigma^m|_y \geq 1$. Then $|t\sigma^{m+1}|_z \geq |t\sigma^m|_x \cdot |x\sigma|_z + |t\sigma^m|_y \cdot |y\sigma|_z \geq 1 + 1 = 2$, so $t\sigma^{m+1}$ is not linear. *QED*

Example 9.3.29. We verify linearity for the two pattern terms from Examples 9.3.20 and 9.3.21 using their variable transition graphs.

For $\langle \mathsf{f}(x), [x/\mathsf{g}(x)] \rangle$, the graph $G_{\sigma,t}$ has a single node $x$ with a self-loop $x \to x$. Since $\mathsf{f}(x)$ is linear, Condition (i) holds. The only node reachable from $\mathcal{V}(t) = \{x\}$ is $x$ itself and $x\sigma = \mathsf{g}(x)$ is linear, so Condition (ii) holds. Condition (iii) holds, since there are no concurrent pairs. By Lemma 9.3.28, $\langle \mathsf{f}(x), [x/\mathsf{g}(x)] \rangle$ is linear.

For $\langle \mathsf{f}(x,y), [x/\mathsf{g}(x,y)] \rangle$, the graph $G_{\sigma,t}$ has nodes $\{x, y\}$ with edges $x \to x$, $x \to y$, and $y \to y$. Since $S_0 = \mathcal{V}(t) = \{x, y\}$, the pair $(x, y)$ is concurrent. Condition (iii) fails for this pair: $y \notin \mathrm{dom}(\sigma)$, so $\mathcal{V}(y\sigma) = \{y\}$, but $y \in \mathcal{V}(x\sigma) = \{x, y\}$, giving $\mathcal{V}(x\sigma) \cap \mathcal{V}(y\sigma) = \{y\} \neq \varnothing$. By Lemma 9.3.28, $\langle \mathsf{f}(x,y), [x/\mathsf{g}(x,y)] \rangle$ is not linear.

Since every pumped term $t\sigma^m$ of a linear pattern term is linear, an occurrence of $\langle t, \sigma \rangle$ that lies strictly below a non-overlapping occurrence $(\pi, q)$ lies in the instance of a variable that occurs only once in $t\sigma^q$, so rewriting it does not destroy the occurrence at $\pi$. This allows us to count all pairwise non-overlapping pattern occurrences instead of only orthogonal ones if we always rewrite an innermost counted occurrence.

Theorem 9.3.30 – Embedding RWs via Linear Patterns.

Let $\mathcal{P}$ be a PTRS, $\langle t, \sigma \rangle$ be a linear pattern term, and let $\mathfrak{T}$ be a finite $\mathcal{P}$-PT for $\langle t, \sigma \rangle$ with $\mathrm{h}(\mathfrak{T}) > 0$.

- If we have $\sum_{v \in \mathrm{Leaf}(\mathfrak{T})} p_v \cdot \mathrm{maxNPO}^{\mathrm{anc}}(t, \sigma, t_v) > 1$, then $\mathcal{P}$ is not `AST`.
- If we have $\sum_{v \in \mathrm{Leaf}(\mathfrak{T})} p_v \cdot \mathrm{maxNPO}^{\mathrm{anc}}(t, \sigma, t_v) \geq 1$, then $\mathcal{P}$ is not `PAST`.

*Proof.* We proceed as in the proof of Theorem 9.3.16, but we now use $\mathrm{maxNPO}(t, \sigma, s)$ for the terms $s$ of the nodes of $\mathfrak{T}^\infty$ and define the random walk program $\mu$ via $\mathrm{maxNPO}^{\mathrm{anc}}(t, \sigma, t_v)$. Again, we have $a_w = 1 \leq \mathrm{maxNPO}(t, \sigma, t\sigma)$ for the root $w$ of $\mathfrak{T}_1$. Let $s$ be the term of a node of $\mathfrak{T}^\infty$ with $\mathrm{maxNPO}(t, \sigma, s) = k > 0$, and let $W \in \mathrm{NPO}(t, \sigma, s)$ with $m_W = k$, where all chosen multiplicities are positive. This time, we choose $(\pi_1, q_1) \in W$ to be *innermost*, i.e., no position of $W$ lies strictly below $\pi_1$, and we rewrite $s = C[t\sigma^{q_1}\delta]$ as in the proof of Theorem 9.3.16. It remains to prove

$$\mathrm{maxNPO}(t, \sigma, t_{\mathsf{e}_s(v)}) \geq \mathrm{maxNPO}^{\mathrm{anc}}(t, \sigma, t_v) + k - 1$$

for every $v \in \mathrm{Leaf}(\mathfrak{T})$, where $t_{\mathsf{e}_s(v)} = C[t_v \sigma^{q_1 - 1} \delta]$.
Every occurrence $(\pi, q)$ of $W' = W \setminus \{(\pi_1, q_1)\}$ is either orthogonal to $\pi_1$, and then it is not affected by the rewrite steps, or it lies strictly above $\pi_1$. In the latter case, we have $\pi_1 = \pi.\tau.\alpha$ for some $\tau \in \mathrm{pos}_{\mathcal{V}}(t\sigma^q)$ and some position $\alpha$, since the two occurrences *Proof*

are non-overlapping. As $\langle t, \sigma\rangle$ is linear, the variable $x = t\sigma^q|_\tau$ occurs only once in $t\sigma^q$. So the rewrite steps below $\pi_1$ only change the instance of $x$, and $(\pi, q)$ is still an occurrence in $t_{\mathsf{e}_s(v)}$. Hence, all occurrences of $W'$ are still present in $t_{\mathsf{e}_s(v)}$ with unchanged positions and multiplicities, and they are still pairwise non-overlapping, since this only depends on their positions and multiplicities. Moreover, we have $m_{W'} = k - q_1$.

As in the proof of Theorem 9.3.16, by Lemma 9.3.15 there is a set $S' \in \mathrm{NPO}(t, \sigma, t_v\sigma^{q_1-1}\delta)$ with $m_{S'} = \mathrm{maxNPO}^{\mathrm{anc}}(t, \sigma, t_v) + q_1 - 1$, and $\{(\pi_1.\chi, r) \mid (\chi, r) \in S'\}$ is a set of pairwise non-overlapping occurrences in $t_{\mathsf{e}_s(v)}$. These occurrences are orthogonal to all occurrences of $W'$ that are orthogonal to $\pi_1$. For every $(\pi, q) \in W'$ with $\pi_1 = \pi.\tau.\alpha$ as above, their positions lie at or below the variable position $\pi.\tau$ of the occurrence $(\pi, q)$, i.e., they are not function symbol positions of $t\sigma^q$ at $\pi$, and they are not above $\pi$. So the union of both sets is in $\mathrm{NPO}(t, \sigma, t_{\mathsf{e}_s(v)})$, and we obtain

$$\begin{aligned} &\mathrm{maxNPO}(t, \sigma, t_{\mathsf{e}_s(v)}) \\ \geq\ & m_{W'} + m_{S'} \\ =\ & (k - q_1) + \mathrm{maxNPO}^{\mathrm{anc}}(t, \sigma, t_v) + q_1 - 1 \\ =\ & \mathrm{maxNPO}^{\mathrm{anc}}(t, \sigma, t_v) + k - 1. \end{aligned}$$

*QED*

We illustrate Theorem 9.3.30 with the following example.

Example 9.3.31. Consider the linear pattern term $\langle t', \sigma\rangle$ with base term $t' = \mathsf{f}(y, x)$ and pumping substitution $\sigma = [y/\mathsf{g}(y)]$. Let $\mathcal{P}_{20}$ be the PTRS with the single rule $\mathsf{f}(\mathsf{g}(y), x) \to \{{}^{2/3}\mathsf{f}(y, x),\ {}^{1/3}\mathsf{f}(\mathsf{g}(\mathsf{g}(y)), \mathsf{f}(\mathsf{g}(y), x))\}$. The corresponding $\to_{\mathcal{P}_{20}}$-RT rooted at $t'\sigma = \mathsf{f}(\mathsf{g}(y), x)$ is a pattern tree for $\langle t', \sigma\rangle$ and is depicted in Figure 9.15. The two leaves are $t_{v_1} = \mathsf{f}(y, x)$ and $t_{v_2} = \mathsf{f}(\mathsf{g}(\mathsf{g}(y)), \mathsf{f}(\mathsf{g}(y), x))$. We have $\mathrm{maxNPO}^{\mathrm{anc}}(t', \sigma, t_{v_1}) = 0$, where the occurrence of $t'$ at the root of $t_{v_1}$ (with multiplicity 0 and the identity as closing substitution) is commuting, and $\mathrm{maxNPO}^{\mathrm{anc}}(t', \sigma, t_{v_2}) = 3$ since the occurrence of $t'$ at the root of $t_{v_2}$ has multiplicity 2 and there is another occurrence of $t'$ at position 2 of $t_{v_2}$ with multiplicity 1. Hence, $2/3 \cdot 0 + 1/3 \cdot 3 = 1$, so $\mathcal{P}_{20}$ is not PAST by Theorem 9.3.30. Note that $t_{v_2}$ is exactly the term $s$ from Example 9.3.14. Hence, the occurrence at position 2 of $t_{v_2}$ is commuting, so $\mathfrak{T}$ is indeed a pattern tree, and the witness set $\{(\varepsilon, 2), (2, 1)\}$ used above is anchored. In contrast, we only have $\mathrm{maxOPO}^{\mathrm{anc}}(t', \sigma, t_{v_2}) = 1$, so Theorem 9.3.16 cannot disprove PAST with this tree.

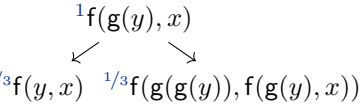



Figure 9.15: $\mathcal{P}_{20}$-PT.

Note that while $t\sigma^n$ is linear for all $n \in \mathbb{N}$, the nodes in the pattern tree do not have to be linear.

### 9.3.3 Counting in Non-Variable-Decreasing Pattern Trees

In this last subsection, we generalize the NVD condition to pattern trees. Recall that the tree $\mathfrak{T}_m$ obtained by starting from $t\sigma^m$ and using the same rewrite rules as the pattern tree $\mathfrak{T}$ at the same positions has the leaves $t_v\sigma^{m-1}$, where $v \in \mathrm{Leaf}(\mathfrak{T})$. Hence, we have to require the NVD condition of Definition 9.2.10 to hold for each such tree $\mathfrak{T}_m$.

**Definition 9.3.32** – Non-Variable-Decreasing Pattern Tree.

*Let $\mathcal{P}$ be a PTRS and let $\mathfrak{T}$ be a $\mathcal{P}$-PT for $\langle t, \sigma\rangle$. $\mathfrak{T}$ is non-variable-decreasing (*NVD*) if for every $m \in \mathbb{N}_{>0}$, every $v \in \mathrm{Leaf}(\mathfrak{T})$, and every $x \in \mathcal{V}$ we have*

$$|t\sigma^m|_x \;\leq\; |t_v\sigma^{m-1}|_x.$$

Example 9.3.33. Reconsider the $\to_{\mathcal{P}_{18}}$-RT $\mathfrak{T}$ of Figure 9.11 as a pattern tree for the pattern term $\langle \mathsf{f}(x), [x/\mathsf{g}(x)]\rangle$. The root is $t\sigma = \mathsf{f}(\mathsf{g}(x))$ and the leaves are $\mathsf{f}(x)$ and $\mathsf{f}(\mathsf{g}(\mathsf{g}(x)))$. Since $\sigma = [x/\mathsf{g}(x)]$ preserves the single occurrence of $x$, we have $|t\sigma^m|_x = 1$ and $|t_v\sigma^{m-1}|_x = 1$ for every $m \geq 1$ and both leaves. Hence, $\mathfrak{T}$ is NVD.

Checking Definition 9.3.32 for all $m \in \mathbb{N}_{>0}$ reduces to comparing variable counts of the concrete root term $t\sigma$ against those of the finitely many leaf terms, i.e., the same linear-time procedure as for ordinary NVD RTs.

**Lemma 9.3.34** – Criterion for NVD Pattern Trees.

*A $\mathcal{P}$-PT $\mathfrak{T}$ for $\langle t, \sigma\rangle$ is* NVD *if and only if $|t\sigma|_x \leq |t_v|_x$ for all $v \in \mathrm{Leaf}(\mathfrak{T})$ and all $x \in \mathcal{V}$.*

*Proof.* We prove both directions individually. *Proof*

► *Proof(Soundness, $\Longleftarrow$)*: We prove $|t\sigma^m|_x \leq |t_v\sigma^{m-1}|_x$ for all $m \geq 1$ by induction on $m$. The base case $m = 1$ follows from the assumption. For the induction step, recall that for any term $s$ and variable $x$: $|s\sigma|_x = \sum_{z\in\mathcal{V}} |s|_z \cdot |z\sigma|_x$. Applying this identity to both sides and using the induction hypothesis $|t\sigma^m|_z \leq |t_v\sigma^{m-1}|_z$ for all $z \in \mathcal{V}$:

$$|t\sigma^{m+1}|_x \;=\; \sum_{z\in\mathcal{V}} |t\sigma^m|_z \cdot |z\sigma|_x \;\leq\; \sum_{z\in\mathcal{V}} |t_v\sigma^{m-1}|_z \cdot |z\sigma|_x \;=\; |t_v\sigma^m|_x,$$

where the inequality holds component-wise since $|z\sigma|_x \geq 0$.

► *Proof(Completeness, $\Longrightarrow$)*: Setting $m = 1$ in Definition 9.3.32 directly yields $|t\sigma|_x \leq |t_v|_x$ for all $v$ and $x$. *QED*

Using NVD pattern trees, once again pairwise non-overlapping (rather than only orthogonal) pattern occurrences can be counted.

**Theorem 9.3.35** – Embedding RWs via NVD Pattern Trees.

Let $\mathcal{P}$ be a PTRS, $\langle t, \sigma\rangle$ be a pattern term, and let $\mathfrak{T}$ be a finite $\mathcal{P}$-PT for $\langle t, \sigma\rangle$ with $\mathrm{h}(\mathfrak{T}) > 0$ that is NVD.

- If we have $\sum_{v\in\mathrm{Leaf}(\mathfrak{T})} p_v \cdot \mathrm{maxNPO}^{\mathrm{anc}}(t, \sigma, t_v) > 1$, then $\mathcal{P}$ is not AST.
- If we have $\sum_{v\in\mathrm{Leaf}(\mathfrak{T})} p_v \cdot \mathrm{maxNPO}^{\mathrm{anc}}(t, \sigma, t_v) \geq 1$, then $\mathcal{P}$ is not PAST.

*Proof.* We proceed as in the proof of Theorem 9.3.30, but we now choose $(\pi_1, q_1) \in W$ to be *outermost*, i.e., no position of $W$ lies strictly above $\pi_1$. Again, it remains to prove *Proof*

$$\mathrm{maxNPO}(t, \sigma, t_{\mathsf{e}_s(v)}) \;\geq\; \mathrm{maxNPO}^{\mathrm{anc}}(t, \sigma, t_v) + k - 1$$

for every $v \in \mathrm{Leaf}(\mathfrak{T})$, where $t_{\mathsf{e}_s(v)} = C[t_v\sigma^{q_1-1}\delta]$.

Every occurrence $(\pi, q)$ of $W' = W \setminus \{(\pi_1, q_1)\}$ is either orthogonal to $\pi_1$, and then it is not affected by the rewrite steps, or it lies strictly below $\pi_1$. In the latter

case, we have $\pi = \pi_1.\tau.\beta$ for some $\tau \in \mathrm{pos}_{\mathcal{V}}(t\sigma^{q_1})$, since the two occurrences are non-overlapping, and $\beta$ is a position of $\delta(x)$ for the variable $x = t\sigma^{q_1}|_\tau$. By the NVD condition (Definition 9.3.32), we have $|t_v\sigma^{q_1-1}|_x \geq |t\sigma^{q_1}|_x$ for every variable $x$. So there is an injective mapping $\tau \mapsto \tau'$ from the variable positions of $t\sigma^{q_1}$ to the variable positions of $t_v\sigma^{q_1-1}$ with $t\sigma^{q_1}|_\tau = t_v\sigma^{q_1-1}|_{\tau'}$. Since $t_{\mathtt{e}_s(v)}|_{\pi_1.\tau'} = \delta(x) = s|_{\pi_1.\tau}$, the occurrence $(\pi_1.\tau.\beta, q)$ reappears as the occurrence $(\pi_1.\tau'.\beta, q)$ in $t_{\mathtt{e}_s(v)}$. These moved occurrences are pairwise non-overlapping: moved occurrences with the same $\tau$ keep their relative positions, and moved occurrences with different $\tau$ lie below different variable positions $\pi_1.\tau'$, which are orthogonal. Moreover, they are orthogonal to the unaffected occurrences of $W'$, which are orthogonal to $\pi_1$. So we obtain a set $W'' \in \mathrm{NPO}(t, \sigma, t_{\mathtt{e}_s(v)})$ with $m_{W''} = m_{W'} = k - q_1$.

As in the proof of Theorem 9.3.16, by Lemma 9.3.15 (with $\theta$ being the identity) there is a set $S' \in \mathrm{NPO}(t, \sigma, t_v\sigma^{q_1-1})$ with $m_{S'} = \mathrm{maxNPO}^{\mathrm{anc}}(t, \sigma, t_v) + q_1 - 1$, where we may again omit all occurrences with multiplicity 0. As occurrences in $t_v\sigma^{q_1-1}$ are also occurrences in its instance $t_v\sigma^{q_1-1}\delta$, the set $\{(\pi_1.\chi, r) \mid (\chi, r) \in S'\}$ consists of pairwise non-overlapping occurrences in $t_{\mathtt{e}_s(v)}$, and these occurrences are orthogonal to the unaffected occurrences of $W'$. They are also non-overlapping with the moved occurrences at the positions $\pi_1.\tau'.\beta$. To see this, note that for every $(\chi, r) \in S'$, all function symbol positions of $t\sigma^r$ at $\chi$ are function symbol positions of $t_v\sigma^{q_1-1}$, whereas $\tau'$ is a variable position of $t_v\sigma^{q_1-1}$. So the position $\pi_1.\tau'.\beta$ is not a function symbol position of $t\sigma^r$ at $\pi_1.\chi$. Moreover, $\pi_1.\chi$ can only lie at or below $\pi_1.\tau'.\beta$ if $\chi = \tau'$ and $\beta = \varepsilon$. But then $t\sigma^r$ would match the variable $t_v\sigma^{q_1-1}|_{\tau'}$, which is impossible, as $r \geq 1$ and $t\sigma \notin \mathcal{V}$ (the root $t\sigma$ of $\mathfrak{T}$ is rewritten, and variables are normal forms). Hence, we obtain

$$\begin{aligned} &\mathrm{maxNPO}(t, \sigma, t_{\mathtt{e}_s(v)}) \\ \geq\ & m_{W''} + m_{S'} \\ =\ & (k - q_1) + \mathrm{maxNPO}^{\mathrm{anc}}(t, \sigma, t_v) + q_1 - 1 \\ =\ & \mathrm{maxNPO}^{\mathrm{anc}}(t, \sigma, t_v) + k - 1. \end{aligned}$$

*QED*

The following example shows how to utilize Theorem 9.3.35.

Example 9.3.36. Consider the non-linear pattern term $\langle t', \sigma\rangle$ with base term $t' = \mathtt{f}(y, x)$ and pumping substitution $\sigma = [y/\mathtt{g}(y, y)]$. Let $\mathcal{P}_{21}$ be the PTRS with the single rule $\mathtt{f}(\mathtt{g}(y, y), x) \to \{^{2/3}\mathtt{c}(\mathtt{f}(y, x), y),\ ^{1/3}\mathtt{f}(\mathtt{g}(\mathtt{g}(y, y), \mathtt{g}(y, y)), \mathtt{f}(\mathtt{g}(y, y), x))\}$. The corresponding $\to_{\mathcal{P}_{21}}$-RT rooted at $t'\sigma = \mathtt{f}(\mathtt{g}(y, y), x)$ is a pattern tree for $\langle t', \sigma\rangle$ and NVD and is depicted in Figure 9.16. The two leaves are $t_{v_1} = \mathtt{c}(\mathtt{f}(y, x), y)$ and $t_{v_2} = \mathtt{f}(\mathtt{g}(\mathtt{g}(y, y), \mathtt{g}(y, y)), \mathtt{f}(\mathtt{g}(y, y), x))$. We have $\mathrm{maxNPO}^{\mathrm{anc}}(t', \sigma, t_{v_1}) = 0$, and $\mathrm{maxNPO}^{\mathrm{anc}}(t', \sigma, t_{v_2}) = 3$ since the occurrence of $t'$ at the root of $t_{v_2}$ has multiplicity 2 and there is another occurrence of $t'$ at position 2 of $t_{v_2}$ with multiplicity 1. Hence, $^2/_3 \cdot 0 + {}^1/_3 \cdot 3 = 1$, so $\mathcal{P}_{21}$ is not PAST by Theorem 9.3.35. As in Example 9.3.31, the occurrence at the root of $t_{v_2}$ is not commuting, but the occurrence at position 2 of $t_{v_2}$ is, and the occurrence of $t'$ at position 1 of $t_{v_1}$ (with multiplicity 0 and the identity as closing substitution) is commuting as well. So $\mathfrak{T}$ is indeed a pattern tree, and the witness set $\{(\varepsilon, 2), (2, 1)\}$ is anchored. In contrast, as in Example 9.3.31, we only have $\mathrm{maxOPO}^{\mathrm{anc}}(t', \sigma, t_{v_2}) = 1$ for orthogonal occurrences, so Theorem 9.3.16 cannot disprove PAST with this tree.

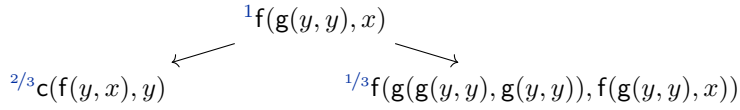



Figure 9.16: $\mathcal{P}_{21}$-PT.

Remark 9.3.37 – Theoretical Comparison of Embedding Techniques via Pattern Terms.

Since $\mathrm{maxNPO}^{\mathrm{anc}}(t,\sigma,s) \geq \mathrm{maxOPO}^{\mathrm{anc}}(t,\sigma,s)$ for all pattern terms $\langle t,\sigma\rangle$ and all terms $s$, Theorem 9.3.30 subsumes Theorem 9.3.16 for linear pattern terms, and Theorem 9.3.35 subsumes Theorem 9.3.16 for NVD pattern trees. Apart from that, they are again incomparable (considering only terms over the function symbols of the respective PTRS):

- Only Theorem 9.3.16 can disprove AST of the PTRS $\mathcal{P}''_{18}$ with the only rule $\mathsf{f}(\mathsf{g}(x,x)) \to \{^{1/3}\mathsf{f}(x), ^{2/3}\mathsf{f}(\mathsf{g}(\mathsf{g}(x,x),\mathsf{g}(x,x)))\}$: for $\langle \mathsf{f}(x), [x/\mathsf{g}(x,x)]\rangle$, the RT of this rule is a pattern tree with $^2/_3 \cdot 2 = {}^4/_3 > 1$, whereas every term with a redex is non-linear (as there are no constants), and no pattern tree is NVD, since always using the first alternative strictly decreases the number of variable occurrences.
- With the RT of its only rule, only Theorem 9.3.30 can disprove PAST of the PTRS $\mathcal{P}'_{20}$ with the rule $\mathsf{f}(\mathsf{g}(y),x) \to \{^{2/3}\mathsf{f}(y,\mathsf{a}),\ ^{1/3}\mathsf{f}(\mathsf{g}(\mathsf{g}(y)),\mathsf{f}(\mathsf{g}(y),x))\}$: as in Example 9.3.31, the sum is 1, whereas this tree is not NVD (since $x$ does not occur in $\mathsf{f}(y,\mathsf{a})$), and the sum for Theorem 9.3.16 is only $^1/_3 \cdot 1 = {}^1/_3$.
- With the RT of its only rule, only Theorem 9.3.35 can disprove PAST of $\mathcal{P}_{21}$ (Example 9.3.36): the pattern term is not linear, and the sum for Theorem 9.3.16 is only $^1/_3 \cdot 1 = {}^1/_3$.

Note that Theorem 9.3.16 is *not* a generalization of Theorem 9.2.15. We can disprove PAST of the PTRS $\mathcal{P}_{22}$ with the rule $\mathsf{g} \to \{^{1/2}\mathsf{a}, ^{1/2}\mathsf{c}(\mathsf{g},\mathsf{g})\}$ via Theorem 9.2.15 by simply taking the tree $\mathfrak{T}$ corresponding to the only rewrite rule and counting the occurrences of $\mathsf{g}$. However, there is no pattern term $\langle t,\sigma\rangle$ such that $t\sigma = \mathsf{g}$ (otherwise, $t$ would be $\mathsf{g}$ or a variable that $\sigma$ maps to $\mathsf{g}$, and in both cases, we would have $t\sigma = t\sigma^2$). So this tree is not a pattern tree, and Theorem 9.3.16 cannot be applied to it.

SECTION 9.4

## Evaluation

As in Section 4.5, we implemented all techniques of this chapter in AProVE and applied them to all 199 PTRSs from the *Termination Problem Database* [TPDB]. All experiments were performed on a machine with 16 GB of RAM and a timeout of 30 seconds for each example.

To isolate these techniques, AProVE here only tries to *disprove* (and does not try to prove) AST and SAST (for full rewriting and arbitrary start terms). Recall that all our disproofs embed a random walk by counting occurrences of loops or looping pattern terms. Table 9.1 reports, for each of the six embedding theorems *in isolation*,

how many PTRSs it disproves on its own. We distinguish counting occurrences of plain looping *terms* (Theorems 9.2.6, 9.2.11 and 9.2.15) from counting occurrences of looping *pattern terms* (Theorems 9.3.16, 9.3.30 and 9.3.35), and within each family the three variants that count *orthogonal*, *linear* (non-overlapping, for linear loops), or *non-variable-decreasing* (NVD) occurrences. The final row ("all together") enables all six embeddings simultaneously. When disproving AST and SAST for other evaluation strategies, AProVE additionally exploits the strategy-equivalence results of Chapter 4: whenever innermost and full rewriting coincide for the PTRS, it moves the innermost problem to full rewriting, where the random-walk disproving techniques apply directly.

| Counting method | Non-AST | Non-SAST |
|---|---|---|
| *Counting terms* | | |
| orthogonal (Theorem 9.2.15) | 26 (13.1%) | 43 (21.6%) |
| linear (Theorem 9.2.6) | 26 (13.1%) | 50 (25.1%) |
| NVD (Theorem 9.2.11) | 26 (13.1%) | 51 (25.6%) |
| *Counting pattern terms* | | |
| orthogonal (Theorem 9.3.16) | 14 (7.0%) | 26 (13.1%) |
| linear (Theorem 9.3.30) | 16 (8.0%) | 30 (15.1%) |
| NVD (Theorem 9.3.35) | 16 (8.0%) | 32 (16.1%) |
| all together | 36 (18.1%) | 72 (36.2%) |

Table 9.1: Evaluation of the embedding theorems for disproving AST and SAST.

All percentages are relative to the 199 benchmarks.

In total, AProVE disproves AST for 36 and SAST for 72 of the 199 PTRSs. The isolated view shows that both the plain-term and the pattern-term embeddings contribute. For AST, counting plain terms alone yields up to 26 disproofs and counting pattern terms up to 16, and only their combination reaches all 36. For SAST the non-overlapping counting is clearly useful: the linear plain-term embedding alone reaches 50 disproofs (versus 43 for orthogonal counting), and enabling all six embeddings raises the total to 72. This shows that counting all different types of occurrences is often essential when disproving SAST.

SECTION 9.5

## Related Work & Open Problems

There exist several techniques to prove non-termination of non-probabilistic term rewriting, see, e.g., [GZ99, GTS05, EEG12, Pay08, Pay24, EZ15], and corresponding tools like NTI [MP23]. Moreover, disproving termination w.r.t. specific evaluation strategies has also been studied for non-probabilistic rewriting, e.g., detecting loops w.r.t. the innermost evaluation strategy [TGS08, TS09b].

A related problem to proving non-termination is analyzing reachability (or infeasibility) for TRSs, see, e.g., [SY19, LG18, GL20a, Yam22a].

For imperative programs, there are also numerous techniques for proving non-termination, e.g., [GHM+08, BSOG12, CCF+14, LNO+14, LH18, FF22, FG23]. There also exist *decision* procedures for termination of certain subclasses of programs that we have already discussed in Section 6.5. Apart from decision procedures for AST

of restricted classes of stochastic systems (see Section 6.5), we are only aware of approaches for probabilistic *imperative* programs to automatically disprove (P)AST: The technique of [CNŽ17] works via repulsing supermartingales, and the technique of [TOUH21] uses fixpoints and Park induction. Based on [CNŽ17], martingale-based proof rules which can also disprove `AST` and `PAST` were developed in [MBKK21a] and implemented in the tool Amber [MBKK21b].

Random walks have already been used in the literature to analyze termination of probabilistic programs. For example, in [BO21], random walks are also used to analyze `AST` (of higher-order probabilistic functional programs). However, here one over-approximates all runs by random walks in order to prove `AST` by counting recursive calls. Instead, our goal is to under-approximate a possible run in order to disprove `AST`. This requires additional conditions to ensure that the counted calls are indeed executable. Moreover, the termination behavior of polynomial random walks themselves was recently analyzed in [WK26].

Finally, a closely related problem to our counting problem for the term or pattern occurrences is the vertex-disjoint graph-packing problem (VGP). The VGP asks how many copies of a graph $H$ can be found in another graph $G$ such that no two copies overlap. For the special case of mapping a tree into another tree, an algorithm similar to Algorithm 8 has been presented in [Mas92]. The more general case quickly becomes NP-hard, see [GJ79]. A quite recent result analyzes the problem under fixed structural parameters of the graphs, e.g., under a fixed tree-width [CEM25].

As future work one can investigate how the techniques for probabilistic *imperative* programs would behave for PTRSs.

> Open Problem 9.5.1 – Repulsing Supermartingales.
>
> Develop a technique to disprove `AST` and `PAST` based on repulsing supermartingales, see [CNŽ17].

> Open Problem 9.5.2 – Park Induction.
>
> Develop a technique to disprove `AST` and `PAST` based on Park induction and fixpoints, see [TOUH21].

In this chapter, we only disprove `AST` and `SAST`, but we do not infer (finite) lower bounds on the expected complexity of PTRSs. For non-probabilistic TRSs, lower bounds on the runtime complexity are inferred in [FGH+17]. In the probabilistic setting, lower bounds are considerably harder to obtain than upper bounds: for probabilistic imperative programs, the induction principle that yields upper bounds on expected values and expected runtimes is unsound for lower bounds, and sound induction rules for lower bounds require additional conditions such as uniform integrability [HKGK20]. Lower bounds on expected runtimes and expected values of probabilistic imperative programs are also inferred in [FCS+23].

> Open Problem 9.5.3 – Lower Bounds.
>
> Develop techniques to infer lower bounds on the expected derivational and runtime complexity of PTRSs.

# 10 Automatic Analysis of PTRSs in AProVE

All the techniques presented in the previous chapters of this thesis are not only of theoretical interest, but they have been fully implemented and are publicly available in the tool AProVE. This is crucial, since the main goal of this thesis is the *fully automatic* analysis of termination and expected runtime complexity of probabilistic term rewriting. A technique that cannot be automated is of little interest for the verification of real-world software. AProVE [GAB+17] (Automated Program Verification Environment) is a tool for the fully automatic verification of programs in several programming languages. It has recently been made publicly available as an open-source tool, which can be found at [APrR]. This release included the new AProVE logo depicted in Figure 10.1.




Figure 10.1: The new AProVE logo.

In this chapter, we describe the tool AProVE, how it can be used, how it is structured internally, and how each of our contributions is realized within the tool. We first give a general overview of AProVE's capabilities and its input formats in Section 10.1, and explain the different ways in which AProVE can be used in Section 10.2. Afterwards, we present the modular internal architecture of AProVE in Section 10.3, which is built around the four central notions of *problems*, *processors*, *proofs*, and *strategies*. This architecture mirrors the theoretical structure of the (annotated) dependency pair frameworks of Chapter 7 and Chapter 8 that are dealing with ADP problems and that use sound processors to transform them into simpler ones. AProVE uses the same structure even for the preprocessing steps directly on the input PTRS. Finally, in Section 10.4 we present the concrete strategy and processors that AProVE uses to analyze `AST` or `SAST`, and to infer bounds on the expected runtime complexity of PTRSs.

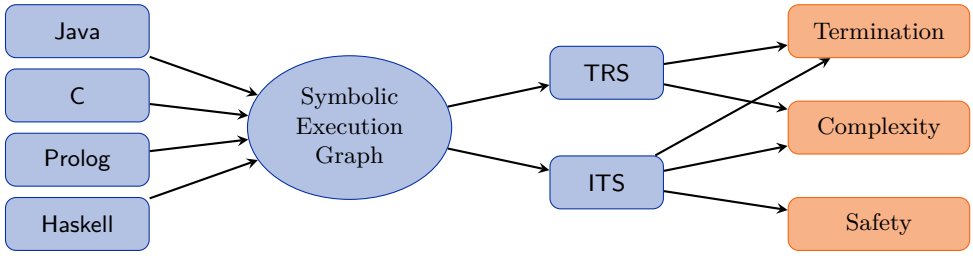



Figure 10.2: Analyzing complex *frontend* languages by transforming them into a *backend* language that is then analyzed for termination, complexity, or safety.

SECTION 10.1

## Capabilities and Input Formats

AProVE can analyze termination, complexity, and safety of programs written in several programming languages, e.g., Java [BOvG10, BOG11, OBvG10, BMOG12, FG17b], C [SGB+14, HGFS16, SGB+17, HGFS18, HG23], Haskell [GSST06, GRS+11], and Prolog [SGST07, SGST09], see Chapter 1. To ensure the correctness of its analyses, AProVE can additionally generate *certificates* that can be checked independently by external certifiers like CeTA [TS09a].

As already discussed in Chapter 1, AProVE strictly separates the analyzed programming languages into *frontend* languages and *backend* languages, see Figure 10.2. *Frontend languages* are the languages that users actually write their software in, e.g., Java, C, Haskell, and Prolog. For each such language, AProVE symbolically executes a given program and represents all its possible runs in a finite graph, namely the *symbolic execution graph*, which over-approximates the behavior of the program while taking care of language-specific features such as pointers in C, heap usage in Java, or lazy evaluation in Haskell. Afterwards, this graph is transformed into a *backend language*, i.e., an analysis-oriented representation that only captures the essential behavior needed for the actual termination, complexity, or safety analysis. Typical backend languages are *term rewrite systems* and *integer transition systems*. The main advantage of this separation is its modularity. The language-specific features of each frontend language only have to be handled once during the construction of the symbolic execution graph, while every improvement of the analysis of a backend language immediately benefits all frontend languages that are translated into it.

In this thesis, we focus on AProVE's analysis of a specific backend language: *probabilistic term rewriting*. Here, AProVE supports four different analysis goals for a given PTRS: certain termination, `AST`, `SAST`, and inferring upper bounds on the expected derivational or runtime complexity. Note that AProVE only allows the user to analyze `AST` and `SAST`, but not `PAST`. The reason is that `PAST` and `SAST` coincide for almost all practically relevant PTRSs. As shown in Theorem 3.4.24, whenever the signature contains at least one function symbol of arity greater than one, then there is no difference between `PAST` and `SAST` for finite PTRSs. Since AProVE only analyzes finite rewrite systems, we therefore decided to omit a dedicated implementation for `PAST`.

In addition, AProVE is able to analyze different evaluation strategies w.r.t. a specific goal: innermost evaluation, outermost evaluation, and an arbitrary evaluation strategy can be analyzed. Moreover, AProVE allows for fixing different sets of start terms that

may start a computation. Here, we currently allow for either arbitrary start terms or only basic start terms, as it is required for runtime complexity.

For the backend language of (probabilistic) term rewriting, AProVE accepts inputs in the *ARI* format [AHK+23] used at the annual *Termination Competition* [GRS+19].[1] In this *ARI* format, a PTRS is given in an S-expression-based syntax. Its precise grammar is given by the following production rules:

$$
\begin{aligned}
\mathit{ptrs} &::= (\mathsf{format\ PTRS})\ \ \mathit{fun}^*\ \ \mathit{prule}^* \\
\mathit{fun} &::= (\mathsf{fun}\ \ \mathit{identifier}\ \ \mathit{nat}) \\
\mathit{term} &::= \mathit{identifier} \quad | \quad (\mathit{identifier}\ \ \mathit{term}^+) \\
\mathit{prule} &::= (\mathsf{prule}\ \ \mathit{term}\ \ \mathit{dist}) \\
\mathit{dist} &::= (\mathit{outcome}^+) \\
\mathit{outcome} &::= (\mathit{term}\ \ \mathit{prob}^?) \\
\mathit{prob} &::= \mathsf{:prob}\ \ \mathit{pos}
\end{aligned}
$$

Here, *nat* and *pos* denote a natural number and a positive natural number, respectively, and *identifier* denotes a valid SMT-LIB symbol [BFT25], where the reserved words fun, prule, and format must not be used as identifiers and quoted and unquoted identifiers must not be mixed. An input file starts with the header (format PTRS), followed by declarations of the form (fun f n) for every function symbol f of arity n, and probabilistic rewrite rules of the form (prule $\ell$ d) for every rule $\ell \to \mu$ of the PTRS. Terms are written in prefix notation, i.e., a term is either an identifier or of the form (f $t_1$ ... $t_k$), and every identifier that is not declared as a function symbol is treated as a variable. A distribution d is a non-empty list of outcomes ($r$ :prob $w$), where $r$ is a term and the weight $w$ is a positive natural number. Instead of storing the normalized probabilities directly, the format only stores these positive integer weights, and the actual branch probabilities are obtained by normalizing all weights of a rule, i.e., the outcome ($r_i$ :prob $w_i$) is taken with probability $w_i/(w_1 + \cdots + w_k)$. If the annotation :prob $w$ is omitted, the default weight $w = 1$ is used.

For example, the PTRS $\mathcal{P}$ with the single rule $\mathsf{g}(x) \to \{{}^{1/2}\mathsf{g}(\mathsf{g}(x)),\ {}^{1/2}x\}$, which induces a symmetric random walk on the number of g-symbols, is represented by the following input:

```
(format PTRS)
(fun g 1)
(prule (g x) (((g (g x)) :prob 1) (x :prob 1)))
```

Here, g is declared as a function symbol of arity 1 and x is a variable, and the two equal weights 1 are normalized to the probability 1/2 each.

[1] The variant of the *ARI* format used for termination analysis slightly differs from the one used for confluence analysis. A description of the termination variant can be found at the *Termination Portal*: https://termination-portal.org/wiki/Term_Rewriting.

SECTION 10.2

## Running AProVE

During the development of this thesis, AProVE has been released as an *open-source* tool [APrR]. Opening the tool to the community allows researchers and developers to explore its techniques, to experiment with new strategies, and to build AProVE into their own workflows. Depending on whether one just wants to try a quick example or integrate AProVE into a larger development process, there are five different ways to run AProVE, each fitting a different use case.

**Web Interface.** The easiest way to get started is the AProVE *web interface*, which is available on AProVE's website [APrW]. It requires no installation at all: one simply pastes or uploads the input, selects the input language and the analysis goal, and lets AProVE run on our servers. The results are rendered directly as HTML in the browser. Hence, the web interface is ideal for trying out a single example, for teaching, or for quickly inspecting AProVE's answer.

**Eclipse Plug-In.** For interactive local use, AProVE's graphical user interface is available as a plug-in for the Eclipse software development environment [GAB+17]. In this way, AProVE can be used directly while developing the analyzed programs: one can edit the input within the IDE, run the analysis, and interactively inspect the resulting (partial) proof tree. In particular, the latter is useful to find out why a proof attempt failed, e.g., to detect which sub-problem could not be solved, and to experiment with the effect of individual processors.

**Command Line Interface.** The *command line interface* is the most direct way to run AProVE on a single file from a terminal or a shell script, e.g., via:

```
java -ea -jar aprove.jar -m wst example.ari
```

The aprove.jar is provided via [APrR]. A range of flags allows one to tailor the analysis, e.g., to select the result format (-m), the proof output format (-p, with plain text or HTML output), the timeout (-t), and the number of threads (-w), or to insert a custom analysis strategy (-s). One can also override declarations made inside the input file without editing it, e.g., the analysis goal (via --goal) or the evaluation strategy (via --rewrite-strategy).

**API.** For developers who want to embed AProVE into their own Java applications, we provide a programmatic *API*. Instead of writing input files by hand, one constructs (probabilistic) rewrite systems programmatically, configures a proof tree (setting timeout, strategy, and certification options), and runs it asynchronously. The analysis then returns one of the three answers YES, NO, or MAYBE (or in the case of complexity analysis, asymptotic lower bounds and upper bounds), and the corresponding proof can again be retrieved in several formats, e.g., as plain text or HTML.

**Docker.** Finally, an official *Docker* container can be used to run AProVE in a fully isolated and reproducible environment, without having to install external tools such

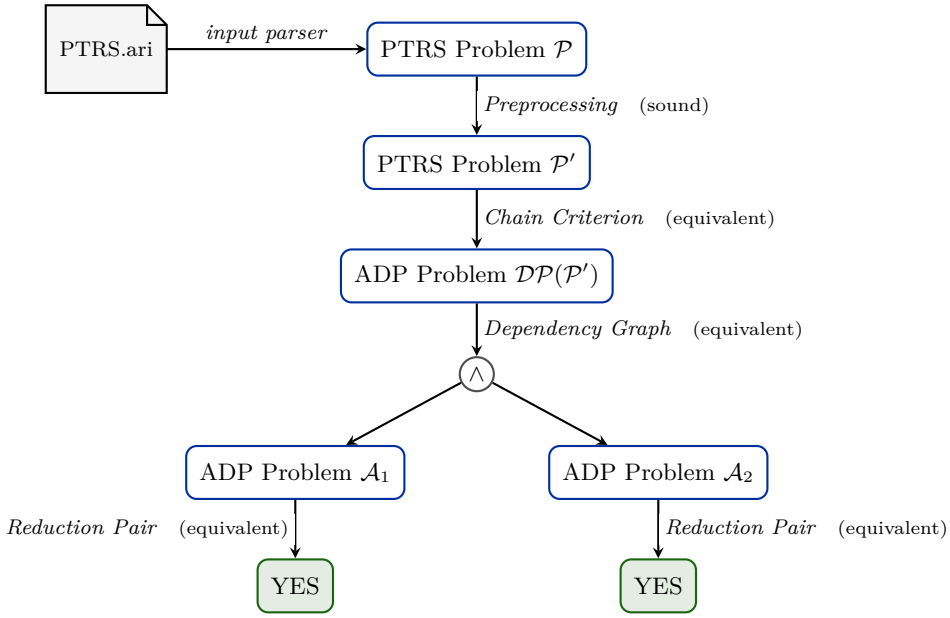



Figure 10.3: A schematic proof tree in AProVE that illustrates the `AST` analysis of a given PTRS. The input file PTRS.ari is read by the *input parser* into the initial PTRS problem at the root. Each node is then a *problem*, and each edge corresponds to applying a *processor*, which transforms a problem into a set of new problems together with the implication (*sound*, *complete*, or *equivalent*) and a junctor ($\wedge$ or $\vee$) that propagates the truth value back to the parent. Once all leaves are solved with YES, we have proven `AST` of $\mathcal{P}$.

as SAT and SMT solvers, e.g., Z3 [dB08], Yices [Dut14], SMT-RAT [CKJ+15], and MiniSat [ES04], or other backend solvers like KoAT [LMG26] and LoAT [FG22b], which all come preconfigured inside the image. This is especially useful for large-scale and reproducible benchmarking, e.g., to run AProVE on a whole problem set in parallel.

SECTION 10.3

## The Modular Architecture of AProVE

No matter which input language and analysis goal are chosen, AProVE always follows the same general workflow: it takes the input, applies a sequence of transformations, and eventually returns an answer, e.g., YES, NO, or MAYBE. This workflow is built around four central notions, namely *problems*, *processors*, *proofs*, and a *redefinable strategy*, see Figure 10.3. These notions correspond directly to the structure of our theoretical results, which is why a technique that fits into the annotated dependency pair framework can be added to AProVE with relatively little effort.

**Problems.** A *problem* is the central data structure that holds all the information required to prove a specific property. For example, a PTRS problem stores the set of rewrite rules, the used signature ($\Sigma = \mathcal{D} \uplus \mathcal{C}$), the evaluation strategy (innermost, outermost, or full rewriting), the analysis goal (`AST`, `SAST`, or complexity), etc. In this

way, the abstract objects that occur in our soundness and completeness proofs, e.g., ADP problems, are precisely the objects that AProVE manipulates at runtime.

**Processors.** A *processor* takes a single problem and transforms it into a (possibly empty) set of new problems. Each processor additionally records how the truth values of the new problems relate to the truth value of the original problem. The two cases that are relevant for us are exactly the ones we use throughout this thesis: a *sound* processor guarantees that the original problem has the desired property (e.g., is AST) whenever all resulting problems do, and a *complete* processor guarantees the converse direction, which is needed to disprove the property. Every processor of our ADP frameworks corresponds to exactly one processor in AProVE. However, we also have additional processors outside the ADP framework, e.g., the move from an arbitrary evaluation strategy to innermost evaluation for PTRSs.

**Proofs.** Whenever a processor is applied, it does not only produce new problems, but it also generates a *proof* object that justifies the transformation in a human-readable form. These proof objects are assembled into a *proof tree* whose root is the original input problem and whose leaves are problems that have been solved. An example of a proof tree is shown in Figure 10.3, which illustrates the interplay between problems and processors. Once the proof tree is complete, AProVE can read off the final answer, and it can export the whole proof as plain text or HTML. For several techniques, AProVE can additionally generate machine-checkable *certificates* that can be verified independently by external certifiers such as CeTA [TS09a]. However, since CeTA does not support probabilistic rewriting yet, AProVE does not generate certificates for the probabilistic techniques of this thesis.

**Strategy.** A *strategy* determines which processor is applied to which problem and in which order. Strategies are written in a small, grammar-like strategy language and are loaded at startup, so that they can be modified, exchanged, or supplied by the user (via the -s flag) *without recompiling* AProVE. Processors are combined by a handful of strategy combinators, e.g., First (try the sub-strategies one after another until one succeeds), Repeat (apply a sub-strategy repeatedly), Any (run several sub-strategies in parallel), Timer (impose a time limit), : (sequential composition of strategies), and many more. This redefinable strategy mechanism is what makes AProVE so flexible: experimenting with a new combination of our processors only amounts to writing a new strategy file.

SECTION 10.4

## Strategies and Processors for AST, SAST, and Expected Complexity

We now describe the concrete strategies that AProVE uses to analyze probabilistic term rewriting. Initially, we always apply some preprocessing steps including trying some simple polynomial or matrix interpretations to quickly get a result for easy examples, and trying to switch to analyzing innermost evaluation (w.r.t. basic start terms when analyzing complexity), as explained in Chapter 4, see Figure 10.4. Afterwards, we move towards the ADP framework for the respective property if possible, or we try more complex interpretations. In addition to this proving strategy, AProVE runs a

disproving strategy in case of AST or SAST in parallel and returns the first conclusive answer.

The *disproving* strategy is similar for AST and SAST, and implements the techniques of Chapter 9. It runs several loop finders in parallel with different time limits, which search for a finite representation of an infinite computation and embed a non-terminating random walk into the rewrite sequences of the PTRS, based either on occurrences of loops or on occurrences of looping pattern terms.

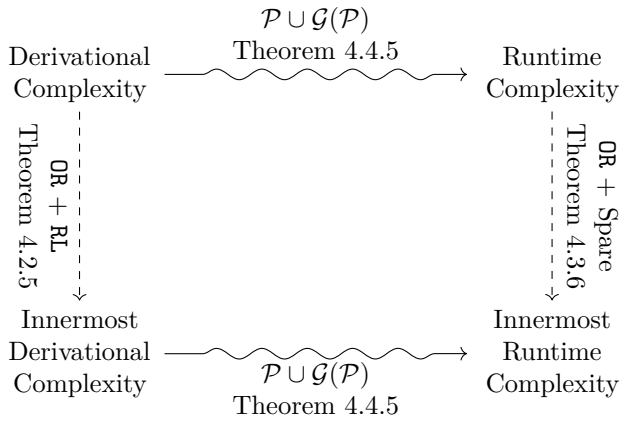



Figure 10.4: Initial preprocessing when analyzing complexity. In the directions from top to bottom we use the relations between innermost evaluation and arbitrary evaluation strategies (Theorem 4.2.5 and Theorem 4.3.6). In the directions from left to right we add the generator rules for the transformational approach from Theorem 4.4.5.

## Almost-Sure Termination and Strong Almost-Sure Termination

The core of the AST analysis and the SAST analysis is the strategy for ADP problems, an excerpt of which is shown in Figure 10.5. It repeatedly applies a set of low-effort *default* processors as long as possible and, whenever none of them is applicable anymore, falls back to the more expensive reduction pair processor or to the transformational processors. The default processors are the structural processors: the dependency graph processor (DependencyGraphProcessor), the usable terms processor (UsableTermsProcessor), the usable rules processor (UsableRulesProcessor), the switching back processor with the decision procedure of Chapter 6 (DecidableRNTProcessor), the probability removal processor (ProbabilityRemovalProcessor), and in addition, the subterm criterion (SubtermCriterion) and a variant of the reduction pair processor with a smaller search space. The reduction pair processor (ReductionPairProcessor) searches for a polynomial or matrix interpretation that orients the annotated dependency pairs and is therefore the most time-consuming due to the additional call to external SMT solvers like Z3 [dB08]. The transformational processors, i.e., the rewriting processor (RewritingProcessor), the (forward) instantiation processors (InstProcessor and ForwardInstProcessor), and the rule overlap instantiation processor (RuleOverlapInstProcessor), transform the ADPs to make the other processors applicable. Each of these processors is the direct implementation of the corresponding processor that we proved sound (and, where applicable, complete) in Chapters 7 and 8. Note that the ADP framework for SAST is only applicable for innermost evaluation w.r.t. basic

```
adp_AST = Repeat(0,*,
            First(
              adp_AST_defaults,
              probTransDecr,
              ReductionPairProcessor,
              probTransIncr
            )
          )

adp_AST_defaults = First(
              DependencyGraphProcessor,
              UsableTermsProcessor,
              UsableRulesProcessor,
              DecidableRNTProcessor,
              ProbabilityRemovalProcessor : qdp,
              SubtermCriterion,
              ReductionPairProcessor
            )
```

Figure 10.5: Excerpt of AProVE's strategy for ADP problems when proving `AST`. Lower-case names denote strategies and upper-case names denote processors. The strategies probTransDecr and probTransIncr contain heuristics for when to apply the transformational ADP processors. The strategy qdp is the strategy for the non-probabilistic DP framework. The currently active strategy can be found in [APrR].

start terms, while the ADP framework for `AST` is applicable for innermost evaluation and, for full evaluation, for non-duplicating PTRSs (Theorem 7.2.19).

### Expected Complexity

For *expected runtime complexity*, AProVE combines a direct application of polynomial interpretations with the complexity variant of the ADP framework of Chapter 8. After transforming the PTRS into an ADP problem, the complexity processors, e.g., the dependency graph, leaf removal, usable terms, usable rules, and knowledge propagation processors, simplify the problem, while the reduction pair processor infers a concrete bound on the expected runtime by iteratively searching for interpretations of increasing degree. As for `SAST`, the bounds derived in this way hold for innermost evaluation starting with a basic term, and the relations of Chapter 4 are used to lift them to derivational complexity and arbitrary evaluation strategies.

SECTION 10.5

## Evaluation

To evaluate the overall power of AProVE with all techniques of this thesis enabled, we applied it to all 199 PTRSs from the *Termination Problem Database* [TPDB], i.e., the benchmarks used for the annual *Termination Competition* [GRS+19], with a timeout of 30 seconds for each example. In contrast to the previous chapters, AProVE here uses its full strategy, combining *all* techniques of this thesis: the relations between evaluation strategies (Chapter 4), the decision procedures (Chapter 6), the ADP frameworks for `AST` (Chapter 7) and `SAST` and expected complexity (Chapter 8), and the disproving techniques (Chapter 9). To give each analysis the full computing resources, we ran

AProVE separately for each analysis goal. As explained in Section 4.5, we do not report execution times, since the most important factor is currently the ability to prove a property at all, especially as we use a fairly short timeout.

For AST, AProVE proves and disproves the numbers of PTRSs shown in Table 10.1, both for full and for innermost evaluation.

| AProVE | full evaluation | innermost evaluation |
|---|---|---|
| AST proved | 90 (45.2%) | 153 (76.9%) |
| AST disproved | 38 (19.1%) | 18 (9.0%) |

Table 10.1: Evaluation of the full AProVE strategy for AST.

All percentages are relative to the 199 benchmarks, and both columns of Table 10.1 consider arbitrary start terms. For disproving, AProVE moves innermost problems to full rewriting (where the random walk techniques are applicable) whenever the strategies coincide. This yields 18 innermost AST disproofs, fewer than the 38 for full evaluation, since the move applies only when innermost and full rewriting coincide.

Since SAST is equivalent to a finite bound on the expected derivational complexity (Lemma 3.3.32), AProVE proves SAST by inferring such a bound. In contrast, a finite bound on the expected runtime complexity only proves SAST w.r.t. basic start terms (see Chapter 4). Therefore, we always analyze the expected complexity and read off SAST (or SAST w.r.t. basic start terms in the case of runtime complexity) from a finite bound. Table 10.2 shows the resulting number of proofs, for expected derivational and runtime complexity, again for full and innermost evaluation.

| SAST proved via | full evaluation | innermost evaluation |
|---|---|---|
| derivational complexity | 43 (21.6%) | 54 (27.1%) |
| runtime complexity | 52 (26.1%) | 69 (34.7%) |

Table 10.2: Evaluation of the full AProVE strategy for SAST via expected complexity. Via runtime complexity, we only prove SAST w.r.t. basic start terms.

The distribution of the finite upper bounds in the strongest (innermost runtime) setting is shown in Table 10.3. Note that the current strategy of AProVE is optimized towards finding *any* finite bound instead of finding an *optimal* bound. This is the reason why we have 3 more finite bounds but less tight bounds compared to the evaluation in Section 8.6.

| Upper bound | $\mathbf{Pol}_0$ | $\mathbf{Pol}_1$ | $\mathbf{Pol}_2$ | **Exp** | **2-Exp** | total |
|---|---|---|---|---|---|---|
| # PTRSs | 23 | 13 | 7 | 25 | 1 | 69 (34.7%) |

Table 10.3: Distribution of the upper bounds on the expected innermost runtime complexity. The total is the number of PTRSs for which we prove SAST w.r.t. basic start terms (innermost).

In addition, AProVE disproves SAST for 72 (36.2%) of the PTRSs. As one would expect, innermost evaluation (with basic start terms for SAST) is the easiest setting to analyze,

so the strongest results are obtained there (153 AST proofs and 69 proofs of SAST w.r.t. basic start terms (innermost)). Together, these results show that the combination of all techniques developed in this thesis yields a powerful tool for the automated termination and complexity analysis of probabilistic term rewrite systems.

SECTION 10.6

## Related Work

There exist numerous approaches and proof rules mainly based on martingales for different properties of probabilistic programs, going back to the seminal works on termination of probabilistic concurrent programs [HSP83] and on ranking supermartingales [CS13]. For example, there are techniques for proving AST and PAST [FFH15, CFG16, ACN18, CFNH18, HFC18, MMKK18, HFCG19, CFN20, AGR21, MBKK21a, CGN+23, MS25], for proving bounds on the termination probability [CNŽ17, KUH19, WSF+21, CGMŽ22, MSBK22, FCS+23, SFCG23], and for upper and lower bounds on expected runtimes and costs [CFM17, KKMO18, NCH18, FC19, GGH19, WFG+19, AMS20, HKGK20, KKM20, MHG21, DWH23, LMG24, CGMŽ24]. Many of these approaches can be automated directly or by extending them with automatic invariant synthesis techniques, e.g., [KMMM10, BEFH16, BTP+22, KMS+22, BCJ+23]. Moreover, one can also use such proof rules within a quantitative program verification infrastructure like Caesar [SBK+23], which builds on a long line of work on deductive verification of probabilistic programs [BKKM18, BKKM21, BCK+21, BBKW24]. Another line of work uses generating functions for exact quantitative reasoning and exact Bayesian inference, as implemented in the tool Prodigy [CKKW22, KWCK23, KBC+24], and inductive synthesis of probabilistic programs is supported by the tool PAYNT [AČJK21, AČJ+21]. Furthermore, for finite-state probabilistic models, quantitative properties can be verified fully automatically with probabilistic model checkers like Storm [HJK+22] and PRISM [KNP11]. Some of these proof rules have already been adapted to term rewriting, e.g., the proof rule of [MMKK18] can be adapted to prove AST of PTRSs via polynomial and matrix interpretations [KG23] (see Theorem 3.4.38) and it is also used within the probabilistic DP framework [KG23, KDG24, KG24] (see Theorem 7.5.1). Further proof rules regarding PAST and SAST have been adapted in [ADLY20] (see Theorem 3.4.33) and again integrated in the corresponding DP framework [KSG25] (see Theorem 8.3.1).

In particular, there also exist several *tools* to analyze AST, PAST, and expected costs for imperative probabilistic programs, e.g., Amber [MBKK21a, MBKK21b], KoAT [LMG26, MHG21, LMG24], Eco-Imp [AMS20], Absynth [NCH18], and Pastry [NYC+25]. Moreover, higher moments for a loop's variables are analyzed automatically with the tool Polar [MSBK22, MMBK25]. These tools mainly consider imperative programs with an innermost evaluation strategy. Hence, our results on the relation between the different evaluation strategies (Chapter 4) cannot be directly used for these tools, while our results on modularity (Chapter 5) may in principle be of interest. However, our results concern the functional recursive nature of term rewriting, where one does not have a fixed control flow as in an imperative program.

Compared to these tools, our implementation in AProVE currently can prove AST (Chapter 7) and SAST, and analyze the expected complexity of PTRSs (Chapter 8). For algorithms whose termination behavior relies only on numbers, techniques for imperative programs with built-in support for arithmetic are usually more powerful

than approaches based on term rewriting. The reason is that for term rewriting, numbers have to be represented via terms, e.g., 2 can be represented by the term $\mathsf{s}(\mathsf{s}(\mathsf{0}))$. On the other hand, term rewriting can handle programs with non-trivial recursive structure and arbitrary user-defined data structures, as these structures can easily be represented as terms. For example, the list $[2, 1]$ can be represented by the term $\mathsf{cons}(\mathsf{s}(\mathsf{s}(\mathsf{0})), \mathsf{cons}(\mathsf{s}(\mathsf{0}), \mathsf{nil}))$. Thus, tools based on term rewriting are particularly suitable when analyzing programs whose termination depends on, e.g., lists, trees, or graphs. In the non-probabilistic setting, there are also techniques and tools for term rewriting with integrated built-in numbers, e.g., for *integer term rewrite systems* [FGP+09] or *logically constrained term rewrite systems* [KN13]. Lifting these approaches to the probabilistic setting is an interesting direction for future work.

In addition to the related work on probabilistic loop programs, there are also several approaches for probabilistic recursive programs, e.g., to analyze the probabilistic lambda calculus or other higher-order functional languages based on types, martingales, or separation logic [ADLG19, DLG19, BO21, KO21, DLFDR21, RBG24, DLP24, GAH+24]. There has also been work on functional languages where `AST` or `PAST` is (partly) decidable, e.g., probabilistic higher-order recursion schemes [KDLG20], probabilistic pushdown automata [BEKK13, BKKV15, WK23], and restricted probabilistic tree-stack automata [LMO22]. Many of the results on recursive languages fix a leftmost-innermost evaluation strategy to avoid non-determinism. Here, our results (Chapter 4) may be helpful to extend these techniques to different evaluation strategies, and also our modularity results (Chapter 5) may be of interest for the different recursive probabilistic languages.

Finally, probabilistic programs that allow for *data structures* are analyzed in [WKH20, LMZ22, BKK+23]. While [BKK+23] uses pointers to represent data structures like tables and lists, [WKH20, LMZ22] consider a probabilistic programming language with matching similar to term rewriting and develop an automatic amortized resource analysis via fixed template potential functions. However, these latter works are mostly targeted towards specific data structures, and we consider general term rewrite systems that can model arbitrary data structures.

# 11 Conclusion

At the end of this thesis, we shortly recapitulate all of its contributions, the impact of these contributions on the current state-of-the-art, and we give an outlook on what may be the next research goals in general.

In the literature, formal verification of probabilistic programs mostly focuses on programs dealing with numbers as the main data structure, and only some works include additional pre-defined data structures. The language of interest is usually not specifically designed to be able to express arbitrary data structures. However, term rewriting is a programming language that can easily express lists, trees, graphs, etc., and it has been successfully used to analyze programs written in languages like Java or C. Thus, with this thesis, we made a significant step towards the formal verification of probabilistic programs dealing with arbitrary data structures. To be precise, we developed the underlying theory of probabilistic term rewrite systems (PTRSs) and a comparison of their different evaluation strategies. Moreover, we presented several transformational approaches for almost-sure termination (AST) and expected complexity analysis, modularity results, and decidability results. Finally, we developed modular frameworks for proving AST, as well as for inferring bounds on the expected runtime complexity, and techniques for disproving termination. Everything is implemented in our tool AProVE and has been evaluated from several points of view.

SECTION 11.1

## Summary of the Contributions

We started in Chapter 1 and Chapter 2 by introducing the general problem of automatic termination and runtime analysis and the idea of using term rewriting as a backend language. Moreover, we introduced all necessary notions from probability theory. This includes the fundamental stochastic systems of random walks, random walk programs, Markov chains, and Markov decision processes that have been used throughout the whole thesis.

In Chapter 3, we introduced the programming formalism of interest. Starting from abstract reduction systems and (ordinary) term rewrite systems, we integrated probabilistic choices after the rule selection, yielding probabilistic abstract reduction systems, and finally, probabilistic term rewrite systems (PTRSs). Then, we gave a unified treatment of three different ways to describe stochastic reductions (based on rewriting distributions, reduction trees, and stochastic processes). We established their equivalence for all three notions of termination, namely AST, PAST, and SAST, and for

expected worst-case complexity in general. Moreover, we proved that PAST and SAST coincide for essentially all PTRSs occurring in practice, and we presented the first proof techniques for AST based on algebraic interpretations. Using analogous approaches, one can also obtain upper runtime bounds and prove PAST and SAST. This chapter forms the backbone of the whole thesis, since it fixes the semantics and the different termination and complexity properties that all remaining chapters analyze. Moreover, its PAST-versus-SAST result already justifies why we can focus on SAST instead of PAST throughout the whole termination analysis.

In Chapter 4, we compared the different evaluation strategies of PTRSs. It is a well-known phenomenon that innermost evaluation is the easiest evaluation strategy to analyze w.r.t. proving termination, while allowing an arbitrary evaluation strategy is the easiest to analyze w.r.t. disproving termination. Hence, we developed the first sufficient criteria for PTRSs which ensure that innermost AST coincides with full AST (namely non-overlappingness and linearity). Moreover, we showed that the same criteria carry over to PAST, SAST, the expected worst-case complexity, and to further strategies such as leftmost-innermost and outermost rewriting. Then, we weakened the linearity requirement to spareness for basic start terms, and we complemented these criteria by transformational approaches that lift runtime complexity bounds back to derivational complexity bounds. Additionally, working in the more complex probabilistic setting even yielded the first results relating derivational and innermost derivational complexity for ordinary, non-probabilistic TRSs.

In Chapter 5, we investigated the composability of PTRSs. We presented the first probabilistic composability results for sequential and parallel execution, and the first modularity results for disjoint, shared constructor, and hierarchical unions, as well as for signature extensions. Moreover, we analyzed witnesses for non-AST showing that important properties fail in the probabilistic setting, but that certain structural guarantees can still be recovered. These results are both theoretically and practically important, as they describe which decompositions are sound and indicate how powerful a corresponding modular framework for the analysis of, e.g., AST, can be. However, they are subsumed by the dependency graph and usable rules processors developed later in the modular annotated dependency pair framework.

In Chapter 6, we studied how difficult termination analysis of PTRSs is in general and identified subclasses for which AST becomes decidable. After explaining why finite Markov decision processes and random walks do not yield suitable subclasses for our purposes, we introduced stochastic context-free grammars (SCFGs) as a first target formalism. Then, we gave a transformation that preserves AST from ground, non-overlapping, and tail-recursive PTRSs (GNT) to SCFGs, and extended it to the broader right-ground class (RNT). This yields the first decision procedures for AST of non-trivial subclasses of PTRSs. This is of theoretical interest, as it narrows down the gap between decidability and undecidability of AST of PTRSs. Additionally, in practice it also helps, as the resulting decision procedure can be integrated directly into our analysis.

In Chapter 7, we developed our main technique for proving AST: The *annotated dependency pair (ADP) framework*. As in the non-probabilistic setting, this is a modular divide-and-conquer framework that uses *processors* to transform a complicated problem into simpler sub-problems. Thus, different termination arguments can be combined and applied locally instead of orienting the whole system at once. Since the probabilistic setting forces us to consider all recursive calls of a rule simultaneously,

we introduced annotated dependency pairs and several structural, algebra-based, and transformational processors. Moreover, we showed that the framework for innermost rewriting is drastically more powerful than the one for full rewriting. Thus, this chapter provides the most powerful automated technique for proving AST of PTRSs up to now.

In Chapter 8, we lifted the ADP framework from the qualitative property AST to the quantitative analysis of expected runtime complexity. By combining our annotated dependency pairs with the dependency pair framework for runtime complexity from [NEG13], we obtained the first modular framework for analyzing the expected runtime of PTRSs. A finite bound on the expected runtime complexity proves SAST w.r.t. basic start terms. Hence, this framework proves SAST w.r.t. basic start terms (innermost) and additionally yields concrete upper bounds on the expected number of rewrite steps.

In Chapter 9, we complemented all the proving techniques by techniques to *disprove* termination, i.e., to disprove AST or SAST, which corresponds to finding bugs instead of proving their absence. We showed that, in contrast to the non-probabilistic setting, searching for a single loop no longer suffices. So instead, we embed non-terminating random walks into the rewrite sequences of a PTRS. The crucial difference to the qualitative loop search for TRSs is that the probabilistic setting requires a quantitative analysis. To be precise, we have to count how many non-overlapping occurrences of a loop or looping pattern term can be executed, which we solve with a dynamic programming algorithm.

Finally, in Chapter 10, we described how the above techniques are realized in our tool AProVE. This is crucial, since the central goal of this thesis is the *fully automatic* analysis of termination and expected runtime complexity of probabilistic term rewriting. A technique that cannot be automated is of little use for the verification of real-world software. We presented AProVE's capabilities and input formats, and the five ways of running the now open-source tool (web interface, command line, API, Eclipse plug-in, and Docker). Moreover, we described its modular architecture built around the four central notions of *problems*, *processors*, *proofs*, and *strategies*. This architecture mirrors the theoretical structure of our (annotated) dependency pair frameworks almost one-to-one.

SECTION 11.2

## Impact on the State-of-the-Art

The individual chapters contribute to several complementary aspects of the automatic verification of PTRSs and probabilistic programs in general. Regarding the *underlying theory*, Chapter 3 and Chapter 5 established the semantic foundations, the different notions of termination and their relationships, and the composability of PTRSs. Regarding the *comparison of the different rewrite relations*, Chapter 4 showed when the various evaluation strategies coincide, which even led to new insights for ordinary TRSs. Regarding *decidability*, Chapter 6 showed the border of decidable AST analysis via stochastic context-free grammars. Regarding the *practical application*, Chapter 7 and Chapter 8 provided modular ADP frameworks for AST as well as for expected runtime complexity and SAST w.r.t. basic start terms (innermost), and Chapter 9 added the corresponding *disproving* techniques.

SECTION 11.3

## Outlook and Future Work

There are several promising directions for future research, which range from improvements of individual techniques to fundamental extensions of the whole formalism. The concrete open problems that we encountered are stated throughout the thesis. Hence, we only mention the most important directions here.

On the *micro level*, one can improve each of the presented techniques individually. For the ADP frameworks, one can develop further processors. Moreover, one can lift more ranking functions from the analysis of TRSs and from imperative probabilistic programs to PTRSs, e.g., by transferring simplification orders, matrix and tuple interpretations, or additional proof rules to the probabilistic setting (Open Problems 3.5.2 to 3.5.4). Similarly, one can look for finer modularity criteria that also cover unions of systems sharing rewrite rules (Open Problem 5.6.1). Moreover, one can extend the decidability results beyond the current RNT restriction and towards `SAST` and expected complexity (Open Problems 6.5.2 and 6.5.3), and adapt further disproving techniques such as repulsing supermartingales and Park induction (Open Problems 9.5.1 and 9.5.2). Another direction is to infer lower bounds on the expected complexity, which is considerably harder than inferring upper bounds (Open Problem 9.5.3). Finally, one can optimize the strategy that decides when to apply which processor, possibly with the help of machine learning (Open Problem 7.9.1).

On the *macro level*, one can target the formalism itself. The current computational model of PTRSs, while Turing complete, is still quite restricted. One could allow more complex, non-constant probability distributions in the right-hand sides of rules (Open Problem 3.5.5). More importantly, one could integrate integers and further SMT-LIB theories directly into the formalism, yielding probabilistic integer term rewrite systems or probabilistic logically constrained term rewrite systems (Open Problem 3.5.6). Thus, such an extension would combine the strengths of term-based formalisms (arbitrary data structures) with those of integer-based formalisms (efficient arithmetic reasoning), which are currently complementary.

Finally, and most importantly for real-world applicability, all of our results assume that the program of interest is already given as a PTRS. However, while computable transformations from languages such as Java and C to rewriting-based formalisms exist in the non-probabilistic setting, no such transformation exists yet in the probabilistic setting (Open Problem 3.5.1). Hence, an important direction for future work is to develop a termination-preserving and practically relevant translation from user-level probabilistic programming languages to PTRSs. Such a translation should keep the generated systems suitable for the automated analyses of this thesis. Then, it would connect all the techniques developed here to the analysis of probabilistic programs as they are actually written in practice.

# Bibliography


[AA20] Ariane Alves Almeida and Mauricio Ayala-Rincón. Formalizing the Dependency Pair Criterion for Innermost Termination. *Sci. Comput. Program.*, 195:102474, 2020. doi:10.1016/j.scico.2020.102474.

[AČJ+21] Roman Andriushchenko, Milan Češka, Sebastian Junges, Joost-Pieter Katoen, and Šimon Stupinský. PAYNT: A Tool for Inductive Synthesis of Probabilistic Programs. In *Proc. CAV '21*, LNCS 12759, pages 856–869, 2021. doi:10.1007/978-3-030-81685-8_40.

[AČJK21] Roman Andriushchenko, Milan Češka, Sebastian Junges, and Joost-Pieter Katoen. Inductive Synthesis for Probabilistic Programs Reaches New Horizons. In *Proc. TACAS '21*, LNCS 12651, pages 191–209, 2021. doi:10.1007/978-3-030-72016-2_11.

[ACN18] Sheshansh Agrawal, Krishnendu Chatterjee, and Petr Novotný. Lexicographic Ranking Supermartingales: An Efficient Approach to Termination of Probabilistic Programs. *Proc. ACM Program. Lang.*, 2(POPL):34:1–34:32, 2018. doi:10.1145/3158122.

[ADLG19] Martin Avanzini, Ugo Dal Lago, and Alexis Ghyselen. Type-Based Complexity Analysis of Probabilistic Functional Programs. In *Proc. LICS '19*, pages 1–13, 2019. doi:10.1109/LICS.2019.8785725.

[ADLY20] Martin Avanzini, Ugo Dal Lago, and Akihisa Yamada. On Probabilistic Term Rewriting. *Sci. Comput. Program.*, 185:102338, 2020. doi:10.1016/j.scico.2019.102338.

[AEF+08] Beatriz Alarcón, Fabian Emmes, Carsten Fuhs, Jürgen Giesl, Raúl Gutiérrez, Salvador Lucas, Peter Schneider-Kamp, and René Thiemann. Improving Context-Sensitive Dependency Pairs. In *Proc. LPAR '08*, LNCS 5330, pages 636–651, 2008. doi:10.1007/978-3-540-89439-1_44.

[AG00] Thomas Arts and Jürgen Giesl. Termination of Term Rewriting Using Dependency Pairs. *Theor. Comput. Sci.*, 236:133–178, 2000. doi:10.1016/S0304-3975(99)00207-8.

[AGL10] Beatriz Alarcón, Raúl Gutiérrez, and Salvador Lucas. Context-Sensitive Dependency Pairs. *Inf. Comput.*, 208:922–968, 2010. doi:10.1016/j.ic.2010.03.003.

[AGR21] Alessandro Abate, Mirco Giacobbe, and Diptarko Roy. Learning Probabilistic Termination Proofs. In *Proc. CAV '21*, LNCS 12760, pages 3–26, 2021. doi:10.1007/978-3-030-81688-9_1.

[AHK+23] Takahito Aoto, Nao Hirokawa, Dohan Kim, Misaki Kojima, Aart Middeldorp, Fabian Mitterwallner, Naoki Nishida, Teppei Saito, Jonas Schöpf, Kiraku Shintani, René Thiemann, and Akihisa Yamada. A New Format for Rewrite Systems. In *Proc. IWC '23*, pages 32–37, 2023. Website of the *ARI* format: https://project-coco.uibk.ac.at/ARI/. Accessed: 2026-08-03.

[AKGK25] Emma Ahrens, Jan-Christoph Kassing, Jürgen Giesl, and Joost-Pieter Katoen. Weighted Rewriting: Semiring Semantics for Abstract Reduction Systems. In *Proc. FSCD '25*, LIPIcs 337, pages 6:1–6:21, 2025. doi:10.4230/LIPIcs.FSCD.2025.6.

[ALM10] Beatriz Alarcón, Salvador Lucas, and José Meseguer. A Dependency Pair Framework for A ∨ C-Termination. In *Proc. WRLA '10*, LNCS 6381, pages 35–51, 2010. doi:10.1007/978-3-642-16310-4_4.

[ALNM09] Beatriz Alarcón, Salvador Lucas, and Rafael Navarro-Marset. Proving Termination with Matrix Interpretations Over The Reals. In *Proc. WST '09*, pages 12–15, 2009. Accessible via the *Termination Portal*: https://termination-portal.org/mediawiki/images/6/63/Proc_WST_2009.pdf. Accessed: 2026-08-03.

[AM16] Martin Avanzini and Georg Moser. A Combination Framework for Complexity. *Inf. Comput.*, 248:22–55, 2016. doi:10.1016/j.ic.2015.12.007.

[AMS06] Gul Agha, José Meseguer, and Koushik Sen. PMaude: Rewrite-based Specification Language for Probabilistic Object Systems. In *Proc. QAPL '05*, ENTCS 153, pages 213–239, 2006. doi:10.1016/j.entcs.2005.10.040.

[AMS16] Martin Avanzini, Georg Moser, and Michael Schaper. TcT: Tyrolean Complexity Tool. In *Proc. TACAS '16*, LNCS 9636, pages 407–423, 2016. doi:10.1007/978-3-662-49674-9_24.

[AMS20] Martin Avanzini, Georg Moser, and Michael Schaper. A Modular Cost Analysis for Probabilistic Programs. *Proc. ACM Program. Lang.*, 4(OOPSLA):172:1–172:30, 2020. doi:10.1145/3428240.

[APrR] Open Source Repository of AProVE: https://github.com/aprove-developers/aprove-open-source. Accessed: 2026-07-19.

[APrW] Website of AProVE: https://aprove.informatik.rwth-aachen.de/. Accessed: 2026-07-19.

[BB88] Gilles Brassard and Paul Bratley. *Algorithmics: Theory and Practice*. Prentice-Hall, 1988.

[BBKW24] Kevin Batz, Tom Jannik Biskup, Joost-Pieter Katoen, and Tobias Winkler. Programmatic Strategy Synthesis: Resolving Nondeterminism in Probabilistic Programs. *Proc. ACM Program. Lang.*, 8:93:2792–93:2820, 2024. doi:10.1145/3632935.

[BCJ+23] Kevin Batz, Mingshuai Chen, Sebastian Junges, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. Probabilistic Program Verification via Inductive Synthesis of Inductive Invariants. In *Proc. TACAS '23*, LNCS 13994, pages 410–429, 2023. doi:10.1007/978-3-031-30820-8_25.

[BCK+19] Tomás Brázdil, Krishnendu Chatterjee, Antonín Kucera, Petr Novotný, and Dominik Velan. Deciding Fast Termination for Probabilistic VASS with Nondeterminism. In *Proc. ATVA '19*, LNCS 11781, pages 462–478, 2019. doi:10.1007/978-3-030-31784-3_27.

[BCK+21] Kevin Batz, Mingshuai Chen, Benjamin Lucien Kaminski, Joost-Pieter Katoen, Christoph Matheja, and Philipp Schröer. Latticed K-Induction with an Application to Probabilistic Programs. In *Proc. CAV '21*, LNCS 12760, pages 524–549, 2021. doi:10.1007/978-3-030-81688-9_25.

[BCMT01] Guillaume Bonfante, Adam Cichon, Jean-Yves Marion, and Hélène Touzet. Algorithms with Polynomial Interpretation Termination Proof. *J. Funct. Program.*, 11(1):33–53, 2001. doi:10.1017/S0956796800003877.

[BEF+16] Marc Brockschmidt, Fabian Emmes, Stephan Falke, Carsten Fuhs, and Jürgen Giesl. Analyzing Runtime and Size Complexity of Integer Programs. *ACM Trans. Program. Lang. Syst.*, 38:13:1–13:50, 2016. doi:10.1145/2866575.

[BEFH16] Gilles Barthe, Thomas Espitau, Luis María Ferrer Fioriti, and Justin Hsu. Synthesizing Probabilistic Invariants via Doob's Decomposition. In *Proc. CAV '16*, LNCS 9779, pages 43–61, 2016. doi:10.1007/978-3-319-41528-4_3.

[BEKK13] Tomáš Brázdil, Javier Esparza, Stefan Kiefer, and Antonín Kučera. Analyzing Probabilistic Pushdown Automata. *Formal Methods Syst. Des.*, 43:124–163, 2013. doi:10.1007/s10703-012-0166-0.

[BFG22] Thaïs Baudon, Carsten Fuhs, and Laure Gonnord. Analysing Parallel Complexity of Term Rewriting. In *Proc. LOPSTR '22*, LNCS 13474, pages 3–23, 2022. doi:10.1007/978-3-031-16767-6_1.

[BFT25] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The SMT-LIB Standard: Version 2.7. Technical report, Department of Computer Science, The University of Iowa, 2025. Available at https://smt-lib.org/. Accessed: 2026-08-03.

[BG05] Olivier Bournez and Florent Garnier. Proving Positive Almost-Sure Termination. In *Proc. RTA '05*, LNCS 3467, pages 323–337, 2005. doi:10.1007/978-3-540-32033-3_24.

[BG06] Olivier Bournez and Florent Garnier. Proving Positive Almost Sure Termination Under Strategies. In *Proc. RTA '06*, LNCS 4098, pages 357–371, 2006. doi:10.1007/11805618_27.

[BK02] Olivier Bournez and Claude Kirchner. Probabilistic Rewrite Strategies. Applications to ELAN. In *Proc. RTA '02*, LNCS 2378, pages 252–266, 2002. doi:10.1007/3-540-45610-4_18.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, 2008.

[BK11] Frédéric Blanqui and Adam Koprowski. CoLoR: A Coq Library on Well-Founded Rewrite Relations and Its Application to the Automated Verification of Termination Certificates. *Math. Struct. Comput. Sci.*, 21:827–859, 2011. doi:10.1017/S0960129511000120.

[BKd03] Marc Bezem, JW Klop, and Roel de Vrijer. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2003.

[BKK+23] Kevin Batz, Benjamin Lucien Kaminski, Joost-Pieter Katoen, Christoph Matheja, and Lena Verscht. A Calculus for Amortized Expected Runtimes. *Proc. ACM Program. Lang.*, 7:1957–1986, 2023. doi:10.1145/3571260.

[BKKM18] Kevin Batz, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. How Long, O Bayesian Network, Will I Sample Thee? In *Proc. ESOP '18*, LNCS 10801, pages 186–213, 2018. doi:10.1007/978-3-319-89884-1_7.

[BKKM21] Kevin Batz, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. Relatively Complete Verification of Probabilistic Programs: An Expressive Language for Expectation-Based Reasoning. *Proc. ACM Program. Lang.*, 5:39:1–39:30, 2021. doi:10.1145/3434320.

[BKKV15] Tomás Brázdil, Stefan Kiefer, Antonín Kucera, and Ivana Hutarová Vareková. Runtime Analysis of Probabilistic Programs with Unbounded Recursion. *J. Comput. Syst. Sci.*, 81:288–310, 2015. doi:10.1016/J.JCSS.2014.06.005.

[Bla06] Frédéric Blanqui. Higher-Order Dependency Pairs. In *Proc. WST '06*, 2006. Accessible via the *Termination Portal*: https://termination-portal.org/mediawiki/images/c/c7/Proc_WST_2006.pdf. Accessed: 2026-08-03.

[BMOG12] Marc Brockschmidt, Richard Musiol, Carsten Otto, and Jürgen Giesl. Automated Termination Proofs for Java Programs with Cyclic Data. In *Proc. CAV '12*, LNCS 7358, pages 105–122, 2012. doi:10.1007/978-3-642-31424-7_13.

[BN98] Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998. doi:10.1017/CBO9781139172752.

[BO21] Raven Beutner and Luke Ong. On Probabilistic Termination of Functional Programs with Continuous Distributions. In *Proc. PLDI '21*, pages 1312–1326, 2021. doi:10.1145/3453483.3454111.

[BOG11] Marc Brockschmidt, Carsten Otto, and Jürgen Giesl. Modular Termination Proofs of Recursive Java Bytecode Programs by Term Rewriting. In *Proc. RTA '11*, LIPIcs 10, pages 155–170, 2011. doi:10.4230/LIPIcs.RTA.2011.155.

[BOvG10] Marc Brockschmidt, Carsten Otto, Christian von Essen, and Jürgen Giesl. Termination Graphs for Java Bytecode. In Simon Siegler and Nathan Wasser, editors, *Verification, Induction, Termination Analysis*, LNCS 6463, pages 17–37. Springer, 2010. doi:10.1007/978-3-642-17172-7_2.

[Bra06] Mark Braverman. Termination of Integer Linear Programs. In *Proc. CAV '06*, LNCS 4144, pages 372–385, 2006. doi:10.1007/11817963_34.

[BSOG12] Marc Brockschmidt, Thomas Ströder, Carsten Otto, and Jürgen Giesl. Automated Detection of Non-Termination and NullPointerExceptions for Java Bytecode. In *Proc. FoVeOOS '11*, LNCS 7421, pages 123–141, 2012. doi:10.1007/978-3-642-31762-0_9.

[BTP+22] Jialu Bao, Nitesh Trivedi, Drashti Pathak, Justin Hsu, and Subhajit Roy. Data-Driven Invariant Learning for Probabilistic Programs. In *Proc. CAV '22*, LNCS 13371, pages 33–54, 2022. doi:10.1007/978-3-031-13185-1_3.

[CCF+07] Evelyne Contejean, Pierre Courtieu, Julien Forest, Olivier Pons, and Xavier Urbain. Certification of Automated Termination Proofs. In *Proc. FroCoS '07*, LNCS 4720, pages 148–162, 2007. doi:10.1007/978-3-540-74621-8_10.

[CCF+14] Hong-Yi Chen, Byron Cook, Carsten Fuhs, Kaustubh Nimkar, and Peter O'Hearn. Proving Nontermination via Safety. In *Proc. TACAS '14*, LNCS 8413, pages 156–171, 2014. doi:10.1007/978-3-642-54862-8_11.

[CCT09] James Cheney, Laura Chiticariu, and Wang Chiew Tan. Provenance in Databases: Why, How, and Where. *Found. Trends Databases*, 1(4):379–474, 2009. doi:10.1561/1900000006.

[CEM25] Barış Can Esmer and Dániel Marx. Generalized Graph Packing Problems Parameterized by Treewidth. In *Proc. ESA '25*, LIPIcs 351, pages 3:1–3:15, 2025. doi:10.4230/LIPIcs.ESA.2025.3.

[CFG16] Krishnendu Chatterjee, Hongfei Fu, and Amir Kafshdar Goharshady. Termination Analysis of Probabilistic Programs through Positivstellensatz's. In *Proc. CAV '16*, LNCS 9779, pages 3–22, 2016. doi:10.1007/978-3-319-41528-4_1.

[CFM17] Krishnendu Chatterjee, Hongfei Fu, and Aniket Murhekar. Automated Recurrence Analysis for Almost-Linear Expected-Runtime Bounds. In *Proc. CAV '17*, LNCS 10426, pages 118–139, 2017. doi:10.1007/978-3-319-63387-9_6.

[CFN20] Krishnendu Chatterjee, Hongfei Fu, and Petr Novotný. Termination Analysis of Probabilistic Programs with Martingales. In Gilles Barthe, Joost-Pieter Katoen, and Alexandra Silva, editors, *Foundations of Probabilistic Programming*, pages 221–258. Cambridge University Press, 2020. doi:10.1017/9781108770750.008.

[CFNH18] Krishnendu Chatterjee, Hongfei Fu, Petr Novotný, and Rouzbeh Hasheminezhad. Algorithmic Analysis of Qualitative and Quantitative Termination Problems for Affine Probabilistic Programs. *ACM Trans. Program. Lang. Syst.*, 40(2):7:1–7:45, 2018. doi:10.1145/3174800.

[CGMŽ22] Krishnendu Chatterjee, Amir Kafshdar Goharshady, Tobias Meggendorfer, and Đorđe Žikelić. Sound and Complete Certificates for Quantitative Termination Analysis of Probabilistic Programs. In *Proc. CAV '22*, LNCS 13371, pages 55–78, 2022. doi:10.1007/978-3-031-13185-1_4.

[CGMŽ24] Krishnendu Chatterjee, Amir Kafshdar Goharshady, Tobias Meggendorfer, and Đorđe Žikelić. Quantitative Bounds on Resource Usage of Probabilistic Programs. *Proc. ACM Program. Lang.*, 8:107:362–107:391, 2024. doi:10.1145/3649824.

[CGN+23] Krishnendu Chatterjee, Ehsan Kafshdar Goharshady, Petr Novotný, Jiri Zárevúcky, and Đorđe Žikelić. On Lexicographic Proof Rules for Probabilistic Termination. *Formal Aspects Comput.*, 35(2):11:1–11:25, 2023. doi:10.1145/3585391.

[CGP10] Pierre Courtieu, Gladys Gbedo, and Olivier Pons. Improved Matrix Interpretation. In *Proc. SOFSEM '10*, LNCS 5901, pages 283–295, 2010. doi:10.1007/978-3-642-11266-9_24.

[CKJ+15] Florian Corzilius, Gereon Kremer, Sebastian Junges, Stefan Schupp, and Erika Ábrahám. SMT − RAT: An Open Source C++ Toolbox for Strategic and Parallel SMT Solving. In *Proc. SAT '15*, LNCS 9340, pages 360–368, 2015. doi:10.1007/978-3-319-24318-4_26.

[CKKW22] Mingshuai Chen, Joost-Pieter Katoen, Lutz Klinkenberg, and Tobias Winkler. Does a Program Yield the Right Distribution? In *Proc. CAV '22*, LNCS 13371, pages 79–101, 2022. doi:10.1007/978-3-031-13185-1_5.

[CNŽ17] Krishnendu Chatterjee, Petr Novotný, and Đorđe Žikelić. Stochastic Invariants for Probabilistic Termination. *SIGPLAN Not.*, 52:145–160, 2017. doi:10.1145/3093333.3009873.

[CS13] Aleksandar Chakarov and Sriram Sankaranarayanan. Probabilistic Program Analysis with Martingales. In *Proc. CAV '13*, LNCS 8044, pages 511–526, 2013. doi:10.1007/978-3-642-39799-8_34.

[Dau92] Max Dauchet. Simulation of Turing Machines by a Regular Rewrite Rule. *Theor. Comput. Sci.*, 103:409–420, 1992. doi:10.1016/0304-3975(92)90022-8.

[dB08] Leonardo de Moura and Nikolaj Bjørner. Z3: An Efficient SMT Solver. In *Proc. TACAS '08*, LNCS 4963, pages 337–340, 2008. doi:10.1007/978-3-540-78800-3_24.

[DCM18] Alejandro Díaz-Caro and Guido Martínez. Confluence in Probabilistic Rewriting. In *Proc. LSFA '17*, ENTCS 338, pages 115–131, 2018. doi:10.1016/j.entcs.2018.10.008.

[DE11] Vincent Danos and Thomas Ehrhard. Probabilistic Coherence Spaces as a Model of Higher-Order Probabilistic Computation. *Inf. Comput.*, 209:966–991, 2011. doi:10.1016/j.ic.2011.02.001.

[Der81] Nachum Dershowitz. Termination of Linear Rewriting Systems. In *Proc. ICALP '81*, LNCS 115, pages 448–458, 1981. doi:10.1007/3-540-10843-2_36.

[Der82] Nachum Dershowitz. Orderings for Term-Rewriting Systems. *Theor. Comput. Sci.*, 17:279–301, 1982. doi:10.1016/0304-3975(82)90026-3.

[Der87] Nachum Dershowitz. Termination of Rewriting. *J. Symb. Comput.*, 3:69–115, 1987. doi:10.1016/S0747-7171(87)80022-6.

[DG19] Katrin M. Dannert and Erich Grädel. Provenance Analysis: A Perspective for Description Logics? In *Description Logic, Theory Combination, and All That: Essays Dedicated to Franz Baader on the Occasion of His 60th Birthday*, LNCS 11560, pages 266–285, 2019. doi:10.1007/978-3-030-22102-7_12.

[DL20] Ugo Dal Lago. On Probabilistic $\lambda$-Calculi. In Gilles Barthe, Joost-Pieter Katoen, and Alexandra Silva, editors, *Foundations of Probabilistic Programming*, pages 121–144. Cambridge University Press, 2020. doi:10.1017/9781108770750.005.

[DLFDR21] Ugo Dal Lago, Claudia Faggian, and Simona Ronchi Della Rocca. Intersection Types and (Positive) Almost-Sure Termination. *Proc. ACM Program. Lang.*, 5:1–32, 2021. doi:10.1145/3434313.

[DLFP26] Ugo Dal Lago, Guido Fiorillo, and Paolo Pistone. On Higher-Order Probabilistic Verification via the Weighted Relational Model of Linear Logic. In *Proc. LICS '26*, LIPIcs 380, pages 34:1–34:27, 2026. doi:10.4230/LIPIcs.LICS.2026.34.

[DLG19] Ugo Dal Lago and Charles Grellois. Probabilistic Termination by Monadic Affine Sized Typing. *ACM Trans. Program. Lang. Syst.*, 41(2):10:1–10:65, 2019. doi:10.1145/3293605.

[DLM12] Ugo Dal Lago and Simone Martini. On Constructor Rewrite Systems and the Lambda Calculus. *Log. Methods Comput. Sci.*, 8(3), 2012. doi:10.2168/LMCS-8(3:12)2012.

[DLP24] Ugo Dal Lago and Luca Padovani. On the Almost-Sure Termination of Binary Sessions. In *Proc. PPDP '24*, pages 9:1–9:12, 2024. doi:10.1145/3678232.3678239.

[DLV20] Ugo Dal Lago and Gabriele Vanoni. On Randomised Strategies in the $\lambda$-Calculus. *Theor. Comput. Sci.*, 813:100–116, 2020. doi:10.1016/j.tcs.2019.09.033.

[DLZ12] Ugo Dal Lago and Margherita Zorzi. Probabilistic Operational Semantics for the Lambda Calculus. *RAIRO Theor. Informatics Appl.*, 46:413–450, 2012. doi:10.1051/ita/2012012.

[DLZG14] Ugo Dal Lago, Sara Zuppiroli, and Maurizio Gabbrielli. Probabilistic Recursion Theory and Implicit Computational Complexity. *Sci. Ann. Comput. Sci.*, 24(2):177–216, 2014. doi:10.7561/SACS.2014.2.177.

[DT] Nachum Dershowitz and Ralf Treinen. The RTA list of open problems. URL: https://www.win.tue.nl/rtaloop/. Accessed 2023-04-11. Currently moving to https://rewriting.inria.fr/open-problems.

[Dut14] Bruno Dutertre. Yices 2.2. In *Proc. CAV '14*, LNCS 8559, pages 737–744, 2014. doi:10.1007/978-3-319-08867-9_49.

[DWH23] Ankush Das, Di Wang, and Jan Hoffmann. Probabilistic Resource-Aware Session Types. *Proc. ACM Program. Lang.*, 7:66:1925–66:1956, 2023. doi:10.1145/3571259.

[ECM26] ECMA-262: ECMAScript 2026 Language Specification, 2026. Ecma International, 17th edition. https://262.ecma-international.org/. Accessed: 2026-07-20.

[EEG12] Fabian Emmes, Tim Enger, and Jürgen Giesl. Proving Non-Looping Non-Termination Automatically. In *Proc. IJCAR '12*, LNCS 7364, pages 225–240, 2012. doi:10.1007/978-3-642-31365-3_19.

[EGK13] Javier Esparza, Andreas Gaiser, and Stefan Kiefer. A Strongly Polynomial Algorithm for Criticality of Branching Processes and Consistency of Stochastic Context-Free Grammars. *Inf. Process. Lett.*, 113:381–385, 2013. doi:10.1016/j.ipl.2013.02.015.

[EGSZ11] Jörg Endrullis, Herman Geuvers, Jakob Grue Simonsen, and Hans Zantema. Levels of Undecidability in Rewriting. *Inf. Comput.*, 209:227–245, 2011. doi:10.1016/j.ic.2010.09.003.

[EH10] Jörg Endrullis and Dimitri Hendriks. Transforming Outermost into Context-Sensitive Rewriting. *Log. Methods Comput. Sci.*, 6(2), 2010. doi:10.2168/LMCS-6(2:5)2010.

[EKM06] Javier Esparza, Antonín Kučera, and Richard Mayr. Model Checking Probabilistic Pushdown Automata. *Log. Methods Comput. Sci.*, 2(1), 2006. doi:10.2168/LMCS-2(1:2)2006.

[ES04] Niklas Eén and Niklas Sörensson. An Extensible SAT-solver. In *Proc. SAT '03*, LNCS 2919, pages 502–518, 2004. doi:10.1007/978-3-540-24605-3_37.

[Eso] Esolang, the Esoteric Programming Languages Wiki: https://esolangs.org/wiki/Main_Page. Accessed 2026-07-19.

[EWZ06] Jörg Endrullis, Johannes Waldmann, and Hans Zantema. Matrix Interpretations for Proving Termination of Term Rewriting. In *Proc. IJCAR '06*, LNCS 4130, pages 574–588, 2006. doi:10.1007/11814771_47.

[EWZ08] Jörg Endrullis, Johannes Waldmann, and Hans Zantema. Matrix Interpretations for Proving Termination of Term Rewriting. *J. Autom. Reason.*, 40:195–220, 2008. doi:10.1007/s10817-007-9087-9.

[EY09] Kousha Etessami and Mihalis Yannakakis. Recursive Markov Chains, Stochastic Grammars, and Monotone Systems of Nonlinear Equations. *J. ACM*, 56:1:1–1:66, 2009. doi:10.1145/1462153.1462154.

[EY15] Kousha Etessami and Mihalis Yannakakis. Recursive Markov Decision Processes and Recursive Stochastic Games. *J. ACM*, 62:11:1–11:69, 2015. doi:10.1145/2699431.

[EZ15] Jörg Endrullis and Hans Zantema. Proving Non-Termination by Finite Automata. In *Proc. RTA '15*, LIPIcs 36, pages 160–176, 2015. doi: 10.4230/LIPICS.RTA.2015.160.

[Fag22] Claudia Faggian. Probabilistic Rewriting and Asymptotic Behaviour: On Termination and Unique Normal Forms. *Log. Methods Comput. Sci.*, 18(2), 2022. doi:10.46298/lmcs-18(2:5)2022.

[FC19] Hongfei Fu and Krishnendu Chatterjee. Termination of Nondeterministic Probabilistic Programs. In *Proc. VMCAI '19*, LNCS 11388, pages 468–490, 2019. doi:10.1007/978-3-030-11245-5_22.

[FCS+23] Shenghua Feng, Mingshuai Chen, Han Su, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Naijun Zhan. Lower Bounds for Possibly Divergent Probabilistic Programs. *Proc. ACM Program. Lang.*, 7:696–726, 2023. doi:10.1145/3586051.

[FDR19] Claudia Faggian and Simona Ronchi Della Rocca. Lambda Calculus and Probabilistic Computation. In *Proc. LICS '19*, pages 1–13, 2019. doi:10.1109/LICS.2019.8785699.

[FF22] Florian Frohn and Carsten Fuhs. A Calculus for Modular Loop Acceleration and Non-Termination Proofs. *Int. J. Softw. Tools Technol. Transf.*, 24:691–715, 2022. doi:10.1007/s10009-022-00670-2.

[FFH15] Luis María Ferrer Fioriti and Holger Hermanns. Probabilistic Termination: Soundness, Completeness, and Compositionality. In *Proc. POPL '15*, pages 489–501, 2015. doi:10.1145/2676726.2677001.

[FG17a] Florian Frohn and Jürgen Giesl. Analyzing Runtime Complexity via Innermost Runtime Complexity. In *Proc. LPAR '17*, EPiC 46, pages 249–268, 2017. doi:10.29007/1nbh.

[FG17b] Florian Frohn and Jürgen Giesl. Complexity Analysis for Java with AProVE. In *Proc. IFM '17*, LNCS 10510, pages 85–101, 2017. doi: 10.1007/978-3-319-66845-1_6.

[FG19] Florian Frohn and Jürgen Giesl. Termination of Triangular Integer Loops is Decidable. In *Proc. CAV '19*, LNCS 11562, pages 426–444, 2019. doi:10.1007/978-3-030-25543-5_24.

[FG22a] Claudia Faggian and Giulio Guerrieri. Strategies for Asymptotic Normalization. In *Proc. FSCD '22*, LIPIcs 228, pages 17:1–17:24, 2022. doi:10.4230/LIPIcs.FSCD.2022.17.

[FG22b] Florian Frohn and Jürgen Giesl. Proving Non-Termination and Lower Runtime Bounds with LoAT (System Description). In *Proc. IJCAR '22*, LNCS 13385, pages 712–722, 2022. doi:10.1007/978-3-031-10769-6_41.

[FG23] Florian Frohn and Jürgen Giesl. Proving Non-Termination by Acceleration Driven Clause Learning (Short Paper). In *Proc. CADE '23*, LNCS 14132, pages 220–233, 2023. doi:10.1007/978-3-031-38499-8_13.

[FGGL26] Florian Frohn, Jürgen Giesl, Peter Giesl, and Nils Lommen. On Deciding Constant Runtime of Linear Loops. In *Proc. TACAS '26*, LNCS 16506, pages 131–150, 2026. doi:10.1007/978-3-032-22749-2_7.

[FGH+17] Florian Frohn, Jürgen Giesl, Jera Hensel, Cornelius Aschermann, and Thomas Ströder. Lower Bounds for Runtime Complexity of Term Rewriting. *J. Autom. Reason.*, 59:121–163, 2017. doi:10.1007/s10817-016-9397-x.

[FGP+09] Carsten Fuhs, Jürgen Giesl, Martin Plücker, Peter Schneider-Kamp, and Stephan Falke. Proving Termination of Integer Term Rewriting. In *Proc. RTA '09*, LNCS 5595, pages 32–47, 2009. doi:10.1007/978-3-642-02348-4_3.

[FHG20] Florian Frohn, Marcel Hark, and Jürgen Giesl. Termination of Polynomial Loops. In *Proc. SAS '20*, LNCS 12389, pages 89–112, 2020. doi:10.1007/978-3-030-65474-0_5.

[FK19] Carsten Fuhs and Cynthia Kop. A Static Higher-Order Dependency Pair Framework. In *Proc. ESOP '19*, LNCS 11423, pages 752–782, 2019. doi:10.1007/978-3-030-17184-1_27.

[Fuh19] Carsten Fuhs. Transforming Derivational Complexity of Term Rewriting to Runtime Complexity. In *Proc. FroCoS '19*, LNCS 11715, pages 348–364, 2019. doi:10.1007/978-3-030-29007-8_20.

[GA01] Jürgen Giesl and Thomas Arts. Verification of Erlang Processes by Dependency Pairs. *Appl. Algebra Eng. Commun. Comput.*, 12(1/2):39–72, 2001. doi:10.1007/s002000100063.

[GAB+17] Jürgen Giesl, Cornelius Aschermann, Marc Brockschmidt, Fabian Emmes, Florian Frohn, Carsten Fuhs, Jera Hensel, Carsten Otto, Martin Plücker, Peter Schneider-Kamp, Thomas Ströder, Stephanie Swiderski, and René Thiemann. Analyzing Program Termination and Complexity Automatically with AProVE. *J. Autom. Reason.*, 58:3–31, 2017. doi:10.1007/s10817-016-9388-y.

[GAH+24] Simon Oddershede Gregersen, Alejandro Aguirre, Philipp G. Haselwarter, Joseph Tassarotti, and Lars Birkedal. Almost-Sure Termination by Guarded Refinement. *Proc. ACM Program. Lang.*, 8:203–233, 2024. doi:10.1145/3674632.

[GAO02] Jürgen Giesl, Thomas Arts, and Enno Ohlebusch. Modular Termination Proofs for Rewriting Using Dependency Pairs. *J. Symb. Comput.*, 34:21–58, 2002. doi:10.1006/jsco.2002.0541.

[Ges90] Alfons Geser. *Relative Termination*. PhD thesis, University of Passau, Germany, 1990.

[GGH19] Jürgen Giesl, Peter Giesl, and Marcel Hark. Computing Expected Runtimes for Constant Probability Programs. In *Proc. CADE '19*, LNCS 11716, pages 269–286, 2019. doi:10.1007/978-3-030-29436-6_16.

[GHM+08] Ashutosh Gupta, Thomas A. Henzinger, Rupak Majumdar, Andrey Rybalchenko, and Ru-Gang Xu. Proving Non-Termination. In *Proc. POPL '08*, pages 147–158, 2008. doi:10.1145/1328438.1328459.

[GHNR14] Andrew D. Gordon, Thomas A. Henzinger, Aditya V. Nori, and Sriram K. Rajamani. Probabilistic Programming. In *Proc. FOSE '14*, pages 167–181, 2014. doi:10.1145/2593882.2593900.

[Gie95] Jürgen Giesl. Generating Polynomial Orderings for Termination Proofs. In *Proc. RTA '95*, LNCS 914, pages 426–431, 1995. doi:10.1007/3-540-59200-8_77.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[GJS+25] James Gosling, Bill Joy, Guy Steele, Gilad Bracha, Alex Buckley, Daniel Smith, and Gavin Bierman. The Java Language Specification, Java SE 25 Edition, 2025. Oracle America, JSR 400. https://docs.oracle.com/javase/specs/jls/se25/html/. Accessed: 2026-07-20.

[GK01] Jürgen Giesl and Deepak Kapur. Dependency Pairs for Equational Rewriting. In *Proc. RTA '01*, LNCS 2051, pages 93–107, 2001. doi:10.1007/3-540-45127-7_9.

[GKT07] Todd J. Green, Grigoris Karvounarakis, and Val Tannen. Provenance Semirings. In *Proc. PODS '07*, pages 31–40, 2007. doi:10.1145/1265530.1265535.

[GL20a] Raúl Gutiérrez and Salvador Lucas. Automatically Proving and Disproving Feasibility Conditions. In *Proc. IJCAR '20*, LNCS 12167, pages 416–435, 2020. doi:10.1007/978-3-030-51054-1_27.

[GL20b] Raúl Gutiérrez and Salvador Lucas. MU – TERM: Verify Termination Properties Automatically (System Description). In *Proc. IJCAR '20*, LNCS 12167, pages 436–447, 2020. doi:10.1007/978-3-030-51054-1_28.

[GM03] Jürgen Giesl and Aart Middeldorp. Innermost Termination of Context-Sensitive Rewriting. In *Proc. DLT '02*, LNCS 2450, pages 231–244, 2003. doi:10.1007/3-540-45005-X_20.

[Gna07] Isabelle Gnaedig. Induction for Positive Almost Sure Termination. In *Proc. PPDP '07*, pages 167–178, 2007. doi:10.1145/1273920.1273943.

[GR14] Jan Gorodkin and Walter L. Ruzzo, editors. *RNA Sequence, Structure, and Function: Computational and Bioinformatic Methods*, volume 1097 of *Methods in Molecular Biology*. Humana Press, 2014. doi:10.1007/978-1-62703-709-9.

[Gra94] Bernhard Gramlich. Generalized Sufficient Conditions for Modular Termination of Rewriting. *Appl. Algebra Eng. Commun. Comput.*, 5:131–158, 1994. doi:10.1007/BF01190827.

[Gra95] Bernhard Gramlich. Abstract Relations between Restricted Termination and Confluence Properties of Rewrite Systems. *Fundam. Informaticae*, 24:3–23, 1995. doi:10.3233/FI-1995-24121.

[Gra96] Bernhard Gramlich. *Termination and Confluence: Properties of Structured Rewrite Systems*. PhD thesis, University of Kaiserslautern, 1996.

[Gra12] Bernhard Gramlich. Modularity in Term Rewriting Revisited. *Theor. Comput. Sci.*, 464:3–19, 2012. doi:10.1016/J.TCS.2012.09.008.

[GRS+11] Jürgen Giesl, Matthias Raffelsieper, Peter Schneider-Kamp, Stephan Swiderski, and René Thiemann. Automated Termination Proofs for Haskell by Term Rewriting. *ACM Trans. Program. Lang. Syst.*, 33:7:1–7:39, 2011. doi:10.1145/1890028.1890030.

[GRS+19] Jürgen Giesl, Albert Rubio, Christian Sternagel, Johannes Waldmann, and Akihisa Yamada. The Termination and Complexity Competition. In *Proc. TACAS '19*, LNCS 11429, pages 156–166, 2019. Website of *TermComp*: https://termination-portal.org/wiki/Termination_Competition. Accessed: 2026-07-19. doi:10.1007/978-3-030-17502-3_10.

[GSST06] Jürgen Giesl, Stephan Swiderski, Peter Schneider-Kamp, and René Thiemann. Automated Termination Analysis for Haskell: From Term Rewriting to Programming Languages. In *Proc. RTA '06*, LNCS 4098, pages 297–312, 2006. doi:10.1007/11805618_23.

[GTS04] Jürgen Giesl, René Thiemann, and Peter Schneider-Kamp. The Dependency Pair Framework: Combining Techniques for Automated Termination Proofs. In *Proc. LPAR '04*, LNCS 3452, pages 301–331, 2004. doi:10.1007/978-3-540-32275-7_21.

[GTS05] Jürgen Giesl, René Thiemann, and Peter Schneider-Kamp. Proving and Disproving Termination of Higher-Order Functions. In *Proc. FroCoS '05*, LNCS 3717, pages 216–231, 2005. doi:10.1007/11559306_12.

[GTSF06] Jürgen Giesl, René Thiemann, Peter Schneider-Kamp, and Stephan Falke. Mechanizing and Improving Dependency Pairs. *J. Autom. Reason.*, 37:155–203, 2006. doi:10.1007/s10817-006-9057-7.

[GZ99] Alfons Geser and Hans Zantema. Non-looping String Rewriting. *RAIRO Theor. Informatics Appl.*, 33:279–301, 1999. doi:10.1051/ita:1999118.

[HÅ01] Karl Hasselström and Jon Åslund. The Shakespeare Programming Language, 2001. KTH Royal Institute of Technology, Stockholm. https://krayon.github.io/shakespearelang/docs/shakespeare.html. Accessed: 2026-07-20.

[HFC18] Mingzhang Huang, Hongfei Fu, and Krishnendu Chatterjee. New Approaches for Almost-Sure Termination of Probabilistic Programs. In *Proc. APLAS '18*, LNCS 11275, pages 181–201, 2018. doi:10.1007/978-3-030-02768-1_11.

[HFCG19] Mingzhang Huang, Hongfei Fu, Krishnendu Chatterjee, and Amir Kafshdar Goharshady. Modular Verification for Almost-Sure Termination of Probabilistic Programs. *Proc. ACM Program. Lang.*, 3(OOPSLA):129:1–129:29, 2019. doi:10.1145/3360555.

[HFG20] Marcel Hark, Florian Frohn, and Jürgen Giesl. Polynomial Loops: Beyond Termination. In *Proc. LPAR '20*, EPiC 73, pages 279–297, 2020. doi:10.29007/nxv1.

[HG23] Jera Hensel and Jürgen Giesl. Proving Termination of C Programs with Lists. In *Proc. CADE '23*, LNCS 14132, pages 266–285, 2023. doi:10.1007/978-3-031-38499-8_16.

[HGFS16] Jera Hensel, Jürgen Giesl, Florian Frohn, and Thomas Ströder. Proving Termination of Programs with Bitvector Arithmetic by Symbolic Execution. In *Proc. SEFM '16*, LNCS 9763, pages 234–252, 2016. doi:10.1007/978-3-319-41591-8_16.

[HGFS18] Jera Hensel, Jürgen Giesl, Florian Frohn, and Thomas Ströder. Termination and Complexity Analysis for Programs with Bitvector Arithmetic by Symbolic Execution. *J. Log. Algebr. Methods Program.*, 97:105–130, 2018. doi:10.1016/j.jlamp.2018.02.004.

[HJK+22] Christian Hensel, Sebastian Junges, Joost-Pieter Katoen, Tim Quatmann, and Matthias Volk. The Probabilistic Model Checker Storm. *Int. J. Softw. Tools Technol. Transf.*, 24:589–610, 2022. doi:10.1007/s10009-021-00633-z.

[HKGK20] Marcel Hark, Benjamin Lucien Kaminski, Jürgen Giesl, and Joost-Pieter Katoen. Aiming Low Is Harder: Induction for Lower Bounds in Probabilistic Program Verification. *Proc. ACM Program. Lang.*, 4(POPL):37:1–37:28, 2020. doi:10.1145/3371105.

[HL78] Gérard Huet and Dallas S. Lankford. On the Uniform Halting Problem for Term Rewriting Systems. Technical Report Rapport Laboria 283, INRIA, 1978.

[HL89] Dieter Hofbauer and Clemens Lautemann. Termination Proofs and the Length of Derivations. In *Proc. RTA '89*, LNCS 355, pages 167–177, 1989. doi:10.1007/3-540-51081-8_107.

[HM04] Nao Hirokawa and Aart Middeldorp. Dependency Pairs Revisited. In *Proc. RTA '04*, LNCS 3091, pages 249–268, 2004. doi:10.1007/978-3-540-25979-4_18.

[HM05] Nao Hirokawa and Aart Middeldorp. Automating the Dependency Pair Method. *Inf. Comput.*, 199:172–199, 2005. doi:10.1016/j.ic.2004.10.004.

[HM07] Nao Hirokawa and Aart Middeldorp. Tyrolean Termination Tool: Techniques and Features. *Inf. Comput.*, 205(4):474–511, 2007. doi:10.1016/J.IC.2006.08.010.

[HM08] Nao Hirokawa and Georg Moser. Automated Complexity Analysis Based on the Dependency Pair Method. In *Proc. IJCAR '08*, LNCS 5195, pages 364–379, 2008. doi:10.1007/978-3-540-71070-7_32.

[HM11] Nao Hirokawa and Aart Middeldorp. Decreasing Diagrams and Relative Termination. *J. Autom. Reason.*, 47(4):481–501, 2011. doi:10.1007/S10817-011-9238-X.

[Hof90] Dieter Hofbauer. Termination Proofs by Multiset Path Orderings Imply Primitive Recursive Derivation Lengths. In *Proc. ALP '90*, LNCS 463, pages 347–358, 1990. doi:10.1007/3-540-53162-9_50.

[Hof01] Dieter Hofbauer. Termination Proofs for Ground Rewrite Systems – Interpretations and Derivational Complexity. *Appl. Algebra Eng. Commun. Comput.*, 12:21–38, 2001. doi:10.1007/s002000100062.

[Hof03] Dieter Hofbauer. An Upper Bound on the Derivational Complexity of Knuth–Bendix Orderings. *Inf. Comput.*, 183:43–56, 2003. doi:10.1016/S0890-5401(03)00008-7.

[HOW19] Mehran Hosseini, Joël Ouaknine, and James Worrell. Termination of Linear Loops over the Integers. In *Proc. ICALP '19*, LIPIcs 132, pages 118:1–118:13, 2019. doi:10.4230/LIPICS.ICALP.2019.118.

[HSP83] Sergiu Hart, Micha Sharir, and Amir Pnueli. Termination of Probabilistic Concurrent Programs. *ACM Trans. Program. Lang. Syst.*, 5:356–380, 1983. doi:10.1145/2166.357214.

[HW06] Dieter Hofbauer and Johannes Waldmann. Termination of String Rewriting with Matrix Interpretations. In *Proc. RTA '06*, LNCS 4098, pages 328–342, 2006. doi:10.1007/11805618_25.

[INVY17] José Iborra, Naoki Nishida, Germán Vidal, and Akihisa Yamada. Relative Termination via Dependency Pairs. *J. Autom. Reason.*, 58(3):391–411, 2017. doi:10.1007/S10817-016-9373-5.

[KB70] Donald E. Knuth and Peter B. Bendix. Simple Word Problems in Universal Algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, 1970. doi:10.1016/B978-0-08-012975-4.50028-X.

[KBC+24] Lutz Klinkenberg, Christian Blumenthal, Mingshuai Chen, Darion Haase, and Joost-Pieter Katoen. Exact Bayesian Inference for Loopy Probabilistic Programs Using Generating Functions. *Proc. ACM Program. Lang.*, 8:127:923–127:953, 2024. doi:10.1145/3649844.

[KC18] Maja H. Kirkeby and Henning Christiansen. Confluence and Convergence in Probabilistically Terminating Reduction Systems. In *Proc. LOPSTR '17*, LNCS 10855, pages 164–179, 2018. doi:10.1007/978-3-319-94460-9_10.

[KDG24] Jan-Christoph Kassing, Stefan Dollase, and Jürgen Giesl. A Complete Dependency Pair Framework for Almost-Sure Innermost Termination of Probabilistic Term Rewriting. In *Proc. FLOPS '24*, LNCS 14659, pages 62–80, 2024. doi:10.1007/978-981-97-2300-3_4.

[KDLG20] Naoki Kobayashi, Ugo Dal Lago, and Charles Grellois. On the Termination Problem for Probabilistic Higher-Order Recursive Programs. *Log. Methods Comput. Sci.*, 16(4), 2020. doi:10.23638/LMCS-16(4:2)2020.

[KFG24] Jan-Christoph Kassing, Florian Frohn, and Jürgen Giesl. From Innermost to Full Almost-Sure Termination of Probabilistic Term Rewriting. In *Proc. FoSSaCS '24*, LNCS 14575, pages 206–228, 2024. doi:10.1007/978-3-031-57231-9_10.

[KG23] Jan-Christoph Kassing and Jürgen Giesl. Proving Almost-Sure Innermost Termination of Probabilistic Term Rewriting Using Dependency Pairs. In *Proc. CADE '23*, LNCS 14132, pages 344–364, 2023. doi:10.1007/978-3-031-38499-8_20.

[KG24] Jan-Christoph Kassing and Jürgen Giesl. Annotated Dependency Pairs for Full Almost-Sure Termination of Probabilistic Term Rewriting. In *Principles of Verification: Cycling the Probabilistic Landscape: Essays Dedicated to Joost-Pieter Katoen on the Occasion of His 60th Birthday*, LNCS 15260, pages 339–366, 2024. doi:10.1007/978-3-031-75783-9_14.

[KG25] Jan-Christoph Kassing and Jürgen Giesl. From Innermost to Full Probabilistic Term Rewriting: Almost-Sure Termination, Complexity, and Modularity. *Log. Methods Comput. Sci.*, 21(4), 2025. doi:10.46298/lmcs-21(4:28)2025.

[KG26] Jan-Christoph Kassing and Jürgen Giesl. The Annotated Dependency Pair Framework for Almost-Sure Termination of Probabilistic Term Rewriting. *Sci. Comput. Program.*, 251:103417, 2026. doi:10.1016/j.scico.2025.103417.

[KH99] B Knudsen and J Hein. RNA Secondary Structure Prediction Using Stochastic Context-Free Grammars and Evolutionary History. *Bioinformatics*, 15(6):446–454, 1999. doi:10.1093/bioinformatics/15.6.446.

[KH12] Dominik Klein and Nao Hirokawa. Confluence of Non-Left-Linear TRSs via Relative Termination. In *Proc. LPAR '12*, LNCS 7180, pages 258–273, 2012. doi:10.1007/978-3-642-28717-6_21.

[KK15] Benjamin Lucien Kaminski and Joost-Pieter Katoen. On the Hardness of Almost-Sure Termination. In *Proc. MFCS '15*, LNCS 9234, pages 307–318, 2015. doi:10.1007/978-3-662-48057-1_24.

[KKM19] Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. On the Hardness of Analyzing Probabilistic Programs. *Acta Inf.*, 56:255–285, 2019. doi:10.1007/s00236-018-0321-1.

[KKM20] Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. Expected Runtime Analysis by Program Verification. In Gilles Barthe, Joost-Pieter Katoen, and Alexandra Silva, editors, *Foundations of Probabilistic Programming*, pages 185–220. Cambridge University Press, 2020. doi:10.1017/9781108770750.007.

[KKMO16] Benjamin Lucien Kaminski, Joost-Pieter Katoen, Christoph Matheja, and Federico Olmedo. Weakest Precondition Reasoning for Expected Run-Times of Probabilistic Programs. In *Proc. ESOP '16*, LNCS 9632, pages 364–389, 2016. doi:10.1007/978-3-662-49498-1_15.

[KKMO18] Benjamin Lucien Kaminski, Joost-Pieter Katoen, Christoph Matheja, and Federico Olmedo. Weakest Precondition Reasoning for Expected Runtimes of Randomized Algorithms. *J. ACM*, 65:1–68, 2018. doi: 10.1145/3208102.

[KL80] S. Kamin and J.-J. Lévy. Two Generalizations of the Recursive Path Ordering. Unpublished Manuscript, University of Illinois, Urbana, IL, 1980.

[KMMM10] Joost-Pieter Katoen, Annabelle McIver, Larissa Meinicke, and Carroll C. Morgan. Linear-Invariant Generation for Probabilistic Programs: Automated Support for Proof-Based Methods. In *Proc. SAS '10*, LNCS 6337, pages 390–406, 2010. doi:10.1007/978-3-642-15769-1_24.

[KMS+22] Andrey Kofnov, Marcel Moosbrugger, Miroslav Stankovič, Ezio Bartocci, and Efstathia Bura. Moment-Based Invariants for Probabilistic Loops with Non-Polynomial Assignments. In *Proc. QEST '22*, LNCS 13479, pages 3–25, 2022. doi:10.1007/978-3-031-16336-4_1.

[KN13] Cynthia Kop and Naoki Nishida. Term Rewriting with Logical Constraints. In *Proc. FroCoS '13*, LNCS 8152, pages 343–358, 2013. doi:10.1007/978-3-642-40885-4_24.

[KNP11] Marta Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of Probabilistic Real-Time Systems. In *Proc. CAV '11*, LNCS 6806, pages 585–591, 2011. doi:10.1007/978-3-642-22110-1_47.

[KNSG26] Jan-Christoph Kassing, Henri Nagel, Alexander Schlecht, and Jürgen Giesl. Disproving (Positive) Almost-Sure Termination of Probabilistic Term Rewriting via Random Walks. In *Proc. IJCAR '26*, LNCS 16689, pages 304–323, 2026. doi:10.1007/978-3-032-32592-1_18.

[KO92] Masahito Kurihara and Azuma Ohuchi. Modularity of Simple Termination of Term Rewriting Systems with Shared Constructors. *Theor. Comput. Sci.*, 103:273–282, 1992. doi:10.1016/0304-3975(92)90015-8.

[KO21] Andrew Kenyon-Roberts and Luke Ong. Supermartingales, Ranking Functions and Probabilistic Lambda Calculus. In *Proc. LICS '21*, pages 1–13, 2021. doi:10.1109/LICS52264.2021.9470550.

[Koz81] Dexter Kozen. Semantics of Probabilistic Programs. *J. Comput. Syst. Sci.*, 22(3):328–350, 1981. doi:10.1016/0022-0000(81)90036-2.

[Koz85] Dexter Kozen. A Probabilistic PDL. *J. Comput. Syst. Sci.*, 30(2):162–178, 1985. doi:10.1016/0022-0000(85)90012-1.

[KR88] Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language*. Prentice Hall, second edition, 1988.

[KR00] M.R.K. Krishna Rao. Some Characteristics of Strong Innermost Normalization. *Theor. Comput. Sci.*, 239(1):141–164, 2000. doi: 10.1016/S0304-3975(99)00215-7.

[KSG25] Jan-Christoph Kassing, Leon Spitzer, and Jürgen Giesl. Dependency Pairs for Expected Innermost Runtime Complexity and Strong Almost-Sure Termination of Probabilistic Term Rewriting. In *Proc. PPDP '25*, pages 10:1–10:14, 2025. doi:10.1145/3756907.3756917.

[KSZM09] Martin Korp, Christian Sternagel, Harald Zankl, and Aart Middeldorp. Tyrolean Termination Tool 2. In *Proc. RTA '09*, LNCS 5595, pages 295–304, 2009. doi:10.1007/978-3-642-02348-4_21.

[KUH19] Satoshi Kura, Natsuki Urabe, and Ichiro Hasuo. Tail Probabilities for Randomized Program Runtimes via Martingales for Higher Moments. In *Proc. TACAS '19*, LNCS 11428, pages 135–153, 2019. doi:10.1007/978-3-030-17465-1_8.

[Kv12] Cynthia Kop and Femke van Raamsdonk. Dynamic Dependency Pairs for Algebraic Functional Systems. *Log. Methods Comput. Sci.*, 8(2), 2012. doi:10.2168/LMCS-8(2:10)2012.

[KV21] Cynthia Kop and Deivid Vale. Tuple Interpretations for Higher-Order Complexity. In *Proc. FSCD '21*, LIPIcs 195, pages 31:1–31:22, 2021. doi:10.4230/LIPIcs.FSCD.2021.31.

[KVG24] Jan-Christoph Kassing, Grigory Vartanyan, and Jürgen Giesl. A Dependency Pair Framework for Relative Termination of Term Rewriting. In *Proc. IJCAR '24*, LNCS 14740, pages 360–380, 2024. doi:10.1007/978-3-031-63501-4_19.

[Kvv07] Jan Willem Klop, Vincent van Oostrom, and Femke van Raamsdonk. Reduction Strategies and Acyclicity. In Hubert Comon-Lundh, Claude Kirchner, and Hélène Kirchner, editors, *Rewriting, Computation and Proof: Essays Dedicated to Jean-Pierre Jouannaud on the Occasion of His 60th Birthday*, LNCS 4600, pages 89–112. Springer, 2007. doi:10.1007/978-3-540-73147-4_5.

[KW08] Adam Koprowski and Johannes Waldmann. Arctic Termination ...Below Zero. In *Proc. RTA '08*, LNCS 5117, pages 202–216, 2008. doi:10.1007/978-3-540-70590-1_14.

[KWCK23] Lutz Klinkenberg, Tobias Winkler, Mingshuai Chen, and Joost-Pieter Katoen. Exact Probabilistic Inference Using Generating Functions, 2023. doi:10.48550/arXiv.2302.00513.

[KY76] Donald E. Knuth and A. Yao. The Complexity of Nonuniform Random Number Generation. In J. F. Traub, editor, *Algorithms and Complexity: New Directions and Recent Results*, pages 357–428. Academic Press, 1976.

[KZ05] Adam Koprowski and Hans Zantema. Proving Liveness with Fairness Using Rewriting. In *Proc. FroCoS '05*, LNCS 3717, pages 232–247, 2005. doi:10.1007/11559306_13.

[Lan75] Dallas S. Lankford. Canonical Algebraic Simplification in Computational Logic. Technical Report ATP-25, University of Texas at Austin, 1975.

[Lan79] Dallas S. Lankford. On Proving Term Rewriting Systems Are Noetherian. Memo MTP-3, Math.Dept.„ Louisiana Technical University, Ruston, LA, 1979.

[Lep01] Ingo Lepper. Derivation Lengths and Order Types of Knuth-Bendix Orders. *Theor. Comput. Sci.*, 269:433–450, 2001. doi:10.1016/S0304-3975(01)00015-9.

[LG18] Salvador Lucas and Raúl Gutiérrez. Use of Logical Models for Proving Infeasibility in Term Rewriting. *Inf. Process. Lett.*, 136:90–95, 2018. doi:10.1016/j.ipl.2018.04.002.

[LH18] Jan Leike and Matthias Heizmann. Geometric Nontermination Arguments. In *Proc. TACAS '18*, LNCS 10806, pages 266–283, 2018. doi:10.1007/978-3-319-89963-3_16.

[LJB01] Chin Soon Lee, Neil D. Jones, and Amir M. Ben-Amram. The Size-Change Principle for Program Termination. In *Proc. POPL '01*, pages 81–92, 2001. doi:10.1145/360204.360210.

[LL10] Gregory F. Lawler and Vlada Limic. *Random Walk: A Modern Introduction*, volume 123 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, 2010.

[LMG24] Nils Lommen, Éléanore Meyer, and Jürgen Giesl. Control-Flow Refinement for Complexity Analysis of Probabilistic Programs in KoAT (Short Paper). In *Proc. IJCAR '24*, LNCS 14739, pages 233–243, 2024. doi:10.1007/978-3-031-63498-7_14.

[LMG26] Nils Lommen, Éléanore Meyer, and Jürgen Giesl. KoAT: Automatic Complexity and Termination Analysis of Integer Programs. In *Proc. CAV '26*, LNCS 16684, pages 54–69, 2026. doi:10.1007/978-3-032-32537-2_3.

[LMO22] Guanyan Li, Andrzej S. Murawski, and Luke Ong. Probabilistic Verification beyond Context-Freeness. In *Proc. LICS '22*, pages 33:1–33:13, 2022. doi:10.1145/3531130.3533351.

[LMZ22] Lorenz Leutgeb, Georg Moser, and Florian Zuleger. Automated Expected Amortised Cost Analysis of Probabilistic Data Structures. In *Proc. CAV '22*, LNCS 13372, pages 70–91, 2022. doi:10.1007/978-3-031-13188-2_4.

[LNO+14] Daniel Larraz, Kaustubh Nimkar, Albert Oliveras, Enric Rodríguez-Carbonell, and Albert Rubio. Proving Non-termination Using Max-SMT. In *Proc. CAV '14*, LNCS 8559, pages 779–796, 2014. doi:10.1007/978-3-319-08867-9_52.

[LT85] P. Lancaster and M. Tismenetsky. *The Theory of Matrices: With Applications*. Computer Science and Scientific Computing. Elsevier, 1985. URL: https://books.google.de/books?id=2c011Aptsa8C.

[Luc05] Salvador Lucas. Polynomials over the Reals in Proofs of Termination: From Theory to Practice. *RAIRO Theor. Informatics Appl.*, 39:547–586, 2005. doi:10.1051/ita:2005029.

[Luc07] Salvador Lucas. Practical Use of Polynomials over the Reals in Proofs of Termination. In *Proc. PPDP '07*, pages 39–50, 2007. doi:10.1145/1273920.1273927.

[Luc10] Salvador Lucas. From Matrix Interpretations over the Rationals to Matrix Interpretations over the Naturals. In *Proc. CICM '10*, LNCS 6167, pages 116–131, 2010. doi:10.1007/978-3-642-14128-7_11.

[MAM+23] César A. Muñoz, Mauricio Ayala-Rincón, Mariano M. Moscato, Aaron Dutle, Anthony J. Narkawicz, Ariane Alves Almeida, Andréia B. Avelar da Silva, and Thiago Mendonça Ferreira Ramos. Formal Verification of Termination Criteria for First-Order Recursive Functions. *J. Autom. Reason.*, 67:40, 2023. doi:10.1007/S10817-023-09669-Z.

[Mas92] Shigeru Masuyama. On the Tree Packing Problem. *Discrete Applied Mathematics*, 35:163–166, 1992. doi:10.1016/0166-218X(92)90040-H.

[MBKK21a] Marcel Moosbrugger, Ezio Bartocci, Joost-Pieter Katoen, and Laura Kovács. Automated Termination Analysis of Polynomial Probabilistic Programs. In *Proc. ESOP '21*, LNCS 12648, pages 491–518, 2021. doi:10.1007/978-3-030-72019-3_18.

[MBKK21b] Marcel Moosbrugger, Ezio Bartocci, Joost-Pieter Katoen, and Laura Kovács. The Probabilistic Termination Tool Amber. In *Proc. FM '21*, LNCS 13047, pages 667–675, 2021. doi:10.1007/978-3-030-90870-6_36.

[MG95] Aart Middeldorp and Bernhard Gramlich. Simple Termination Is Difficult. *Appl. Algebra Eng. Commun. Comput.*, 6:115–128, 1995. doi:10.1007/BF01225647.

[MG25] Éléanore Meyer and Jürgen Giesl. Deciding Termination of Simple Randomized Loops. In *Proc. MFCS '25*, LIPIcs 345, pages 76:1–76:19, 2025. doi:10.4230/LIPIcs.MFCS.2025.76.

[MHG21] Fabian Meyer, Marcel Hark, and Jürgen Giesl. Inferring Expected Runtimes of Probabilistic Integer Programs Using Expected Sizes. In *Proc. TACAS '21*, LNCS 12651, pages 250–269, 2021. doi:10.1007/978-3-030-72016-2_14.

[Mid90] Aart Middeldorp. *Modular Properties of Term Rewriting Systems*. PhD thesis, Vrije Universiteit Amsterdam, The Netherlands, 1990.

[MM05] Annabelle McIver and Carroll Morgan. *Abstraction, Refinement and Proof for Probabilistic Systems*. Springer, 2005. doi:10.1007/B138392.

[MMBK25] Marcel Moosbrugger, Julian Müllner, Ezio Bartocci, and Laura Kovács. Polar: An Algebraic Analyzer for (Probabilistic) Loops. In Nils Jansen, Sebastian Junges, Benjamin Lucien Kaminski, Christoph Matheja, Thomas Noll, Tim Quatmann, Mariëlle Stoelinga, and Matthias Volk, editors, *Principles of Verification: Cycling the Probabilistic Landscape: Essays Dedicated to Joost-Pieter Katoen on the Occasion of His 60th Birthday*, LNCS 15260, pages 179–200. Springer, 2025. doi:10.1007/978-3-031-75783-9_8.

[MMKK18] Annabelle McIver, Carroll Morgan, Benjamin Lucien Kaminski, and Joost-Pieter Katoen. A New Proof Rule for Almost-Sure Termination. *Proc. ACM Program. Lang.*, 2(POPL):33:1–33:28, 2018. doi:10.1145/3158121.

[MMT24] Fabian Mitterwallner, Aart Middeldorp, and René Thiemann. Linear Termination Is Undecidable. In *Proc. LICS '24*, pages 1–12, 2024. doi:10.1145/3661814.3662081.

[MN70] Z. Manna and S. Ness. On the Termination of Markov Algorithms. In *Proc. HICSS '70*, pages 789–792, 1970.

[Mor18] David Morgan-Mar. Piet, 2018. Esoteric programming language specification, last updated 2018. https://www.dangermouse.net/esoteric/piet.html. Accessed: 2026-07-20.

[Mos06] Georg Moser. Derivational Complexity of Knuth-Bendix Orders Revisited. In *Proc. LPAR '06*, LNCS 4246, pages 75–89, 2006. doi:10.1007/11916277_6.

[MP15] Georg Moser and Thomas Powell. On the Computational Content of Termination Proofs. In *Proc. CiE '15*, LNCS 9136, pages 276–285, 2015. doi:10.1007/978-3-319-20028-6_28.

[MP23] Frédéric Mesnard and Étienne Payet. NTI+cTI: A Logic Programming Termination Analyzer. In *Proc. WST '23*, pages 5–7, 2023. Accessible via the *Termination Portal*: https://termination-portal.org/mediawiki/images/b/b2/Proc_WST_2023.pdf. Accessed: 2026-08-03.

[MR95] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[MS24] Rupak Majumdar and V. R. Sathiyanarayana. Positive Almost-Sure Termination: Complexity and Proof Rules. *Proc. ACM Program. Lang.*, 8:1089–1117, 2024. doi:10.1145/3632879.

[MS25] Rupak Majumdar and V. R. Sathiyanarayana. Sound and Complete Proof Rules for Probabilistic Termination. *Proc. ACM Program. Lang.*, 9:1871–1902, 2025. doi:10.1145/3704899.

[MSBK22] Marcel Moosbrugger, Miroslav Stankovič, Ezio Bartocci, and Laura Kovács. This Is the Moment for Probabilistic Loops. *Proc. ACM Program. Lang.*, 6:178:1497–178:1525, 2022. doi:10.1145/3563341.

[MV06] Panagiotis Manolios and Daron Vroon. Termination Analysis with Calling Context Graphs. In *Proc. CAV '06*, LNCS 4144, pages 401–414, 2006. doi:10.1007/11817963_36.

[NCH18] Van Chan Ngo, Quentin Carbonneaux, and Jan Hoffmann. Bounded Expectations: Resource Analysis for Probabilistic Programs. In *Proc. PLDI '18*, pages 496–512, 2018. doi:10.1145/3192366.3192394.

[NEG13] Lars Noschinski, Fabian Emmes, and Jürgen Giesl. Analyzing Innermost Runtime Complexity of Term Rewriting by Dependency Pairs. *J. Autom. Reason.*, 51:27–56, 2013. doi:10.1007/s10817-013-9277-6.

[New42] Newman, Maxwell Herman Alexander. On Theories with a Combinatorial Definition of Equivalence. *Annals of Mathematics*, 43:223–242, 1942. doi:10.2307/1968867.

[NFB+17] Matthias Naaf, Florian Frohn, Marc Brockschmidt, Carsten Fuhs, and Jürgen Giesl. Complexity Analysis for Term Rewriting by Integer Transition Systems. In *Proc. FroCoS '17*, LNCS 10483, pages 132–150, 2017. doi:10.1007/978-3-319-66167-4_8.

[NFZ17] Julian Nagele, Bertram Felgenhauer, and Harald Zankl. Certifying Confluence Proofs via Relative Termination and Rule Labeling. *Log. Methods Comput. Sci.*, 13(2), 2017. doi:10.23638/LMCS-13(2:4)2017.

[NM12] Friedrich Neurauter and Aart Middeldorp. On the Domain and Dimension Hierarchy of Matrix Interpretations. In *Proc. LPAR '12*, LNCS 7180, pages 320–334, 2012. doi:10.1007/978-3-642-28717-6_25.

[NYC+25] Sergei Novozhilov, Mingqi Yang, Mingshuai Chen, Zhiyang Li, and Jianwei Yin. On the Almost-Sure Termination of Probabilistic Counter Programs. In *Proc. CAV '25*, LNCS 15932, pages 82–104, 2025. doi:10.1007/978-3-031-98679-6_4.

[OBvG10] Carsten Otto, Marc Brockschmidt, Christian von Essen, and Jürgen Giesl. Automated Termination Analysis of Java Bytecode by Term Rewriting. In *Proc. RTA '10*, LIPIcs 6, pages 259–276, 2010. doi:10.4230/LIPIcs.RTA.2010.259.

[O'D77] M. J. O'Donnell. *Computing in Systems Described by Equations*. LNCS 58. Springer, 1977. doi:10.1007/3-540-08531-9.

[Ohl94] Enno Ohlebusch. On the Modularity of Termination of Term Rewriting Systems. *Theor. Comput. Sci.*, 136:333–360, 1994. doi:10.1016/0304-3975(94)00039-L.

[Ohl95] Enno Ohlebusch. Modular Properties of Composable Term Rewriting Systems. *J. Symb. Comput.*, 20:1–41, 1995. doi:10.1006/jsco.1995.1036.

[Ohl02a] Enno Ohlebusch. *Advanced Topics in Term Rewriting*. Springer, 2002. doi:10.1007/978-1-4757-3661-8.

[Ohl02b] Enno Ohlebusch. Hierarchical Termination Revisited. *Inf. Process. Lett.*, 84:207–214, 2002. doi:10.1016/S0020-0190(02)00272-7.

[OPW15] Joël Ouaknine, João Sousa Pinto, and James Worrell. On Termination of Integer Linear Loops. In *Proc. SODA '15*, pages 957–969, 2015. doi:10.1137/1.9781611973730.65.

[Pay08] Étienne Payet. Loop Detection in Term Rewriting Using the Eliminating Unfoldings. *Theor. Comput. Sci.*, 403:307–327, 2008. doi:10.1016/j.tcs.2008.05.013.

[Pay24] Étienne Payet. Non-termination in Term Rewriting and Logic Programming. *J. Autom. Reason.*, 68(1):4, 2024. doi:10.1007/s10817-023-09693-z.

[PBC25] Francesco Pontiggia, Ezio Bartocci, and Michele Chiari. POPACheck: A Model Checker for Probabilistic Pushdown Automata. In *Proc. CAV '25*, LNCS 15932, pages 105–121, 2025. doi:10.1007/978-3-031-98679-6_5.

[Pey03] Simon Peyton Jones, editor. *Haskell 98 Language and Libraries: The Revised Report*. Cambridge University Press, 2003.

[Put94] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 1994.

[Rao94] M. R. K. Krishna Rao. Simple Termination of Hierarchical Combinations of Term Rewriting Systems. In *Proc. TACS '94*, LNCS 789, pages 203–223, 1994. doi:10.1007/3-540-57887-0_97.

[Rao95] M. R. K. Krishna Rao. Modular Proofs for Completeness of Hierarchical Term Rewriting Systems. *Theor. Comput. Sci.*, 151:487–512, 1995. doi:10.1016/0304-3975(95)00075-8.

[RBG24] Vineet Rajani, Gilles Barthe, and Deepak Garg. A Modal Type Theory of Expected Cost in Higher-Order Probabilistic Programs. *Proc. ACM Program. Lang.*, 8:389–414, 2024. doi:10.1145/3689725.

[RF25] Chet Ramey and Brian Fox. Bash Reference Manual, 2025. Edition 5.3, Free Software Foundation. https://www.gnu.org/software/bash/manual/bash.html. Accessed: 2026-07-20.

[Ric53] Henry Gordon Rice. Classes of Recursively Enumerable Sets and Their Decision Problems. *Trans. Am. Math. Soc.*, 74(2):358–366, 1953. doi:10.1090/S0002-9947-1953-0053041-6.

[Rog87] Hartley Rogers. *Theory of Recursive Functions and Effective Computability*. MIT Press, 1987.

[Rus87] Michaël Rusinowitch. On Termination of the Direct Sum of Term-Rewriting Systems. *Inf. Process. Lett.*, 26:65–70, 1987. doi:10.1016/0020-0190(87)90039-1.

[RZ09] Matthias Raffelsieper and Hans Zantema. A Transformational Approach to Prove Outermost Termination Automatically. *Electron. Notes Theor. Comput. Sci.*, 237:3–21, 2009. doi:10.1016/j.entcs.2009.03.032.

[SBK+23] Philipp Schröer, Kevin Batz, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. A Deductive Verification Infrastructure for Probabilistic Programs. *Proc. ACM Program. Lang.*, 7:2052–2082, 2023. doi:10.1145/3622870.

[SFCG23] Yican Sun, Hongfei Fu, Krishnendu Chatterjee, and Amir Kafshdar Goharshady. Automated Tail Bound Analysis for Probabilistic Recurrence Relations. In *Proc. CAV '23*, LNCS 13966, pages 16–39, 2023. doi:10.1007/978-3-031-37709-9_2.

[SGB+14] Thomas Ströder, Jürgen Giesl, Marc Brockschmidt, Florian Frohn, Carsten Fuhs, Jera Hensel, and Peter Schneider-Kamp. Proving Termination and Memory Safety for Programs with Pointer Arithmetic. In *Proc. IJCAR '14*, LNCS 8562, pages 208–223, 2014. doi:10.1007/978-3-319-08587-6_15.

[SGB+17] Thomas Ströder, Jürgen Giesl, Marc Brockschmidt, Florian Frohn, Carsten Fuhs, Jera Hensel, Peter Schneider-Kamp, and Cornelius Aschermann. Automatically Proving Termination and Memory Safety for Programs with Pointer Arithmetic. *J. Autom. Reason.*, 58:33–65, 2017. doi:10.1007/s10817-016-9389-x.

[SGST07] Peter Schneider-Kamp, Jürgen Giesl, Alexander Serebrenik, and René Thiemann. Automated Termination Analysis for Logic Programs by Term Rewriting. In *Proc. LOPSTR '06*, LNCS 4407, pages 177–193, 2007. doi:10.1007/978-3-540-71410-1_13.

[SGST09] Peter Schneider-Kamp, Jürgen Giesl, Alexander Serebrenik, and René Thiemann. Automated Termination Proofs for Logic Programs by Term Rewriting. *ACM Trans. Comput. Log.*, 11:2:1–2:52, 2009. doi:10.1145/1614431.1614433.

[Spi01] Frank Spitzer. *Principles of Random Walk*, volume 34 of *Graduate Texts in Mathematics*. Springer, 2001.

[SS11] Andreas Schnabl and Jakob Grue Simonsen. The Exact Hardness of Deciding Derivational and Runtime Complexity. In *Proc. CSL '11*, LIPIcs 12, pages 481–495, 2011. doi:10.4230/LIPIcs.CSL.2011.481.

[SY19] Christian Sternagel and Akihisa Yamada. Reachability Analysis for Termination and Confluence of Rewriting. In *Proc. TACAS '19*, LNCS 11427, pages 262–278, 2019. doi:10.1007/978-3-030-17462-0_15.

[Tao15] Terence Tao. 275A, Notes 0: Foundations of probability theory, 2015. URL: https://terrytao.wordpress.com/2015/09/29/275a-notes-0-foundations-of-probability-theory/. Accessed 2026-07-19.

[TG05] René Thiemann and Jürgen Giesl. The Size-Change Principle and Dependency Pairs for Termination of Term Rewriting. *Appl. Algebra Eng. Commun. Comput.*, 16:229–270, 2005. doi:10.1007/S00200-005-0179-7.

[TGS08] René Thiemann, Jürgen Giesl, and Peter Schneider-Kamp. Deciding Innermost Loops. In *Proc. RTA '08*, LNCS 5117, pages 366–380, 2008. doi:10.1007/978-3-540-70590-1_25.

[Thi07] René Thiemann. *The DP Framework for Proving Termination of Term Rewriting*. PhD thesis, RWTH Aachen University, 2007.

[Thi09] René Thiemann. From Outermost Termination to Innermost Termination. In *Proc. SOFSEM '09*, LNCS 5404, pages 533–545, 2009. doi:10.1007/978-3-540-95891-8_48.

[Tiw04] Ashish Tiwari. Termination of Linear Programs. In *Proc. CAV '04*, LNCS 3114, pages 70–82, 2004. doi:10.1007/978-3-540-27813-9_6.

[TKB95] Yoshihito Toyama, Jan Willem Klop, and Henk P. Barendregt. Termination for Direct Sums of Left-Linear Complete Term Rewriting Systems. *J. ACM*, 42:1275–1304, 1995. doi:10.1145/227683.227689.

[TOUH21] Toru Takisaka, Yuichiro Oyabu, Natsuki Urabe, and Ichiro Hasuo. Ranking and Repulsing Supermartingales for Reachability in Randomized Programs. *ACM Trans. Program. Lang. Syst.*, 43:5:1–5:46, 2021. doi:10.1145/3450967.

[Toy87a] Yoshihito Toyama. Counterexamples to Termination for the Direct Sum of Term Rewriting Systems. *Inf. Process. Lett.*, 25:141–143, 1987. doi:10.1016/0020-0190(87)90122-0.

[Toy87b] Yoshihito Toyama. On the Church-Rosser Property for the Direct Sum of Term Rewriting Systems. *J. ACM*, 34:128–143, 1987. doi:10.1145/7531.7534.

[TPDB] Termination Problem Data Base: https://github.com/TermCOMP/TPDB. Accessed: 2026-07-19.

[TS09a] René Thiemann and Christian Sternagel. Certification of Termination Proofs Using CeTA. In *Proc. TPHOLS '09*, LNCS 5674, pages 452–468, 2009. doi:10.1007/978-3-642-03359-9_31.

[TS09b] René Thiemann and Christian Sternagel. Loops under Strategies. In *Proc. RTA '09*, LNCS 5595, pages 17–31, 2009. doi:10.1007/978-3-642-02348-4_2.

[Urb04] Xavier Urbain. Modular and Incremental Automated Termination Proofs. *J. Autom. Reason.*, 32(4):315–355, 2004. doi:10.1007/BF03177743.

[Wal15] Johannes Waldmann. Matrix Interpretations on Polyhedral Domains. In *Proc. RTA '15*, LIPIcs 36, pages 318–333, 2015. doi:10.4230/LIPIcs.RTA.2015.318.

[Wei95] Andreas Weiermann. Termination Proofs for Term Rewriting Systems by Lexicographic Path Orderings Imply Multiply Recursive Derivation Lengths. *Theor. Comput. Sci.*, 139:355–362, 1995. doi:10.1016/0304-3975(94)00135-6.

[WFG+19] Peixin Wang, Hongfei Fu, Amir Kafshdar Goharshady, Krishnendu Chatterjee, Xudong Qin, and Wenjun Shi. Cost Analysis of Nondeterministic Probabilistic Programs. In *Proc. PLDI '19*, pages 204–220, 2019. doi:10.1145/3314221.3314581.

[WK23] Tobias Winkler and Joost-Pieter Katoen. On Certificates, Expected Runtimes, and Termination in Probabilistic Pushdown Automata. In *Proc. LICS '23*, pages 1–13, 2023. doi:10.1109/LICS56636.2023.10175714.

[WK26] Lorenz Winkler and Laura Kovács. Positive Almost-Sure Termination of Polynomial Random Walks. In *Proc. QEST+FORMATS '25*, LNCS 16143, pages 275–292, 2026. doi:10.1007/978-3-032-05792-1_15.

[WKH20] Di Wang, David M. Kahn, and Jan Hoffmann. Raising Expectations: Automating Expected Cost Analysis with Types. *Proc. ACM Program. Lang.*, 4(ICFP):110:1–110:31, 2020. doi:10.1145/3408992.

[WKMTd94] Jan Willem Klop, Aart Middeldorp, Yoshihito Toyama, and Roel de Vrijer. Modularity of Confluence: A Simplified Proof. *Inf. Process. Lett.*, 49:101–109, 1994. doi:10.1016/0020-0190(94)90034-5.

[WSF+21] Jinyi Wang, Yican Sun, Hongfei Fu, Krishnendu Chatterjee, and Amir Kafshdar Goharshady. Quantitative Analysis of Assertion Violations in Probabilistic Programs. In *Proc. PLDI '21*, pages 1171–1186, 2021. doi:10.1145/3453483.3454102.

[Yam14] Akihisa Yamada. *The Weighted Path Order for Termination of Term Rewriting*. PhD thesis, Nagoya University, 2014.

[Yam22a] Akihisa Yamada. Term Orderings for Non-reachability of (Conditional) Rewriting. In *Proc. IJCAR '22*, LNCS 13385, pages 248–267, 2022. doi:10.1007/978-3-031-10769-6_15.

[Yam22b] Akihisa Yamada. Tuple Interpretations for Termination of Term Rewriting. *J. Autom. Reason.*, 66:667–688, 2022. doi:10.1007/s10817-022-09640-4.

[YKS14] Akihisa Yamada, Keiichirou Kusakari, and Toshiki Sakabe. Nagoya Termination Tool. In *Proc. RTA-TLCA '14*, LNCS 8560, pages 466–475, 2014. doi:10.1007/978-3-319-08918-8_32.

[YKS15] Akihisa Yamada, Keiichirou Kusakari, and Toshiki Sakabe. A Unified Ordering for Termination Proving. *Sci. Comput. Program.*, 111:110–134, 2015. doi:10.1016/j.scico.2014.07.009.

[ZK14] Harald Zankl and Martin Korp. Modular Complexity Analysis for Term Rewriting. *Log. Methods Comput. Sci.*, 10(1), 2014. doi:10.2168/LMCS-10(1:19)2014.

## ABSTRACT

Probabilistic programs (i.e., programs that can make random choices during execution) are central to modern computing. They are used for randomized algorithms, Bayesian inference, cryptographic protocols, and many other applications. While randomness brings many benefits, it also introduces new challenges for formal verification. In particular, a probabilistic program may terminate with probability one (almost-sure termination, AST) while still admitting infinite executions. Moreover, even when termination is guaranteed with probability one, the expected runtime may be infinite, motivating the stronger notion of strong almost-sure termination (SAST) where for every initial state, the expected runtime is bounded by a finite constant, uniformly for all resolutions of the non-determinism.

In this thesis, we present a framework for the automatic analysis of AST and SAST for probabilistic term rewrite systems (PTRSs), a rule-based functional programming language that combines non-determinism with probabilistic choice. Since programs in languages such as Java, C, Haskell, and Prolog can be automatically transformed into term rewrite systems, approaches based on term rewriting have broad applications.

We present several contributions towards this goal. Most notable is the annotated dependency pair framework, a modular, divide-and-conquer method for proving AST and SAST. We extend this framework to derive upper bounds on the expected runtime and complement it with automated techniques for disproving AST and SAST via reductions to stochastic processes such as random walks. Moreover, we investigate several other important properties for termination analysis. First, we consider relations between different evaluation strategies: when does AST for an innermost (or call-by-value) evaluation strategy imply AST for an arbitrary evaluation strategy? This is an important question for automatic termination analysis as it is easier to prove termination for innermost evaluation than for an arbitrary evaluation strategy. Second, we show how the combination of two disjoint systems behaves if both original systems are AST or SAST. Finally, we present restrictions on PTRSs under which AST becomes decidable, yielding the first decision procedure for AST of a non-trivial subclass of PTRSs.

All techniques to analyze PTRSs have been implemented in the tool AProVE and demonstrate strong performance in the annual termination competition.