Verification and Certification of Probabilistic Recursive Programs via Pushdown Automata

Von der Fakultät für Informatik der RWTH Aachen University zur Erlangung des akademischen Grades eines Doktors der Naturwissenschaften genehmigte Dissertation

vorgelegt von

Tobias Winkler, M. Sc.

Berichter:
Prof. Dr. Ir. Dr. h.c. Joost-Pieter Katoen
Prof. Dr. Dr. h.c. Javier Esparza

Tag der mündlichen Prüfung: 7. August 2026

Diese Dissertation ist auf den Internetseiten der Universitätsbibliothek verfügbar.

Abstract

Probabilistic programs are conventional computer programs that incorporate randomness through coin flips, random sampling from distributions, and similar mechanisms. These programs have many applications across diverse disciplines such as data structures and algorithms, machine learning, and security. In this thesis, we focus on probabilistic programs with *recursion*—the ability of a sub-program to invoke itself, either directly or indirectly. Besides being a fundamental programming construct, recursion is particularly useful for programs operating on tree-like and other context-free data structures, which are ubiquitous in computer science.

Our approach to the quantitative analysis and verification of probabilistic recursive programs is fundamentally *automata-theoretic*. Specifically, as is common in the literature, we model such programs as *probabilistic pushdown automata* (pPDA). A key advantage of this approach is that it can be fully automated for programs operating on finite-domain variables, such as Booleans and bounded integers. However, despite the rich literature on pPDA, many fundamental challenges remain open.

First, while numeric methods for efficiently approximating various pPDA quantities of interest—termination probabilities, expected termination times, etc.—already exist, these methods do not output any form of proof that the computed values are sufficiently precise. This makes it difficult to trust the results of such methods, especially in safety-critical applications. To address this gap, we introduce and study *certificates* for pPDA: lightweight, independently verifiable proofs for specific verification problems. Most prominently, we develop certificates for *positive almost sure termination* (i.e., termination within finite expected time) and prove their *completeness*: every pPDA satisfying positive almost sure termination admits a corresponding certificate. We also present a novel *Optimistic Value Iteration* algorithm to compute our certificates numerically and evaluate its practical effectiveness on a suite of benchmarks.

Second, the context-free structure of the runs of a probabilistic recursive program calls for a dedicated property specification formalism. However, the literature has so far considered almost exclusively *regular* properties, such as those encoded by *linear temporal logic* (LTL). *CaRet* is a popular extension of LTL that can capture many *visibly pushdown languages* (VPLs), a class strictly between the regular and the context-free languages. In particular, CaRet can express standard Hoare-like total correctness properties of recursive programs (e.g., that a procedure returns a value satisfying a given postcondition whenever it is invoked in a state satisfying a given precondition), which are not expressible in LTL. We prove that model checking pPDA against CaRet formulas and, more generally, against VPLs given in terms of automata is decidable and establish corresponding complexity bounds.

Zusammenfassung

Probabilistische Programme sind gewöhnliche Computerprogramme, die Zufall durch Münzwürfe, Stichproben aus Wahrscheinlichkeitsverteilungen und ähnliche Mechanismen nutzen. Sie finden vielfältige Anwendungen, etwa bei Datenstrukturen und Algorithmen, im maschinellen Lernen und in der IT-Sicherheit. Wir untersuchen probabilistische Programme mit *Rekursion*, also der Fähigkeit eines Unterprogramms, sich direkt oder indirekt selbst aufzurufen. Rekursion ist ein grundlegendes Sprachkonstrukt und besonders nützlich für die Verarbeitung baumartiger oder allgemein kontextfreier Datenstrukturen, die in der Informatik allgegenwärtig sind.

Unser Ansatz zur quantitativen Analyse und Verifikation probabilistischer rekursiver Programme ist *automatentheoretisch*. Wie in der Literatur üblich, modellieren wir solche Programme als *probabilistische Kellerautomaten* (probabilistic pushdown automata, pPDA). Dieser Ansatz lässt sich für Programme mit Variablen über endlichen Wertebereichen, etwa booleschen Variablen oder beschränkten Ganzzahlen, vollständig automatisieren. Trotz der umfangreichen Forschung zu pPDA sind jedoch zahlreiche grundlegende Fragen offen.

Erstens gibt es bereits numerische Verfahren zur effizienten Approximation verschiedener für pPDA relevanter Größen, etwa der Terminierungswahrscheinlichkeit oder der erwarteten Terminierungszeit. Diese Verfahren liefern jedoch keinen Nachweis dafür, dass die berechneten Werte hinreichend genau sind. Gerade in sicherheitskritischen Anwendungen ist es daher schwierig, sich auf diese Ergebnisse zu verlassen. Um diese Lücke zu schließen, entwickeln und untersuchen wir *Zertifikate* für pPDA: einfache, unabhängig überprüfbare mathematische Nachweise für konkrete Verifikationsprobleme. Insbesondere entwickeln wir Zertifikate für *positive fast sichere Terminierung* (d. h. Terminierung in endlicher erwarteter Zeit) und beweisen ihre *Vollständigkeit*: Jeder pPDA, der positiv fast sicher terminiert, besitzt ein entsprechendes Zertifikat. Zudem stellen wir mit der *Optimistic Value Iteration* einen neuartigen Algorithmus zur numerischen Berechnung dieser Zertifikate vor und bewerten seine praktische Leistungsfähigkeit anhand von Benchmarks.

Zweitens erfordert die kontextfreie Struktur der Ausführungen probabilistischer rekursiver Programme eine darauf abgestimmte Spezifikationssprache. Die bisherige Literatur betrachtet jedoch nahezu ausschließlich *reguläre* Eigenschaften, etwa solche, die sich in *linearer temporaler Logik* (LTL) ausdrücken lassen. *CaRet* ist eine verbreitete Erweiterung von LTL, die viele *visibly pushdown languages* (VPLs) beschreiben kann. Diese Sprachklasse liegt strikt zwischen den regulären und den kontextfreien Sprachen. Insbesondere kann CaRet klassische Hoare-artige Eigenschaften der totalen Korrektheit rekursiver Programme ausdrücken, die sich in LTL nicht formulieren lassen. Dazu gehört etwa, dass ein Unterprogramm bei jedem Aufruf in einem Zustand, der eine gegebene Vorbedingung erfüllt, mit einem Rückgabewert terminiert, der eine vorgegebene Nachbedingung erfüllt. Wir zeigen, dass das *Model-Checking*-Problem für pPDA bezüglich CaRet-Formeln und allgemeiner bezüglich durch Automaten gegebener VPLs entscheidbar ist, und bestimmen entsprechende Komplexitätsschranken.

Acknowledgements

Joost-Pieter, I am deeply grateful for the years I spent at your chair. You always believed in me—so much so that you offered me a position twice—and supported me in every possible way. I am especially thankful for your guidance, encouragement, and trust, as well as for all the trips and conferences—from Argentina to Japan—, the papers and projects, the cycling tours, the coffee breaks, and so much more. I really hope that we can stay connected and continue our collaboration in the future.

Birgit and Elke, thank you so much for your constant support and for helping me with all the administrative and organizational matters—from contracts and travel arrangements to student assistants and UnRAVeL-related affairs. Thanks to you, everything always went smoothly, and I know that this is not easy to achieve—thank you!

Kevin and Lutz, I was extremely lucky to meet you on the very first day of our bachelor's studies—an incredible 13 years ago! Everything would have been different without you, and I am very grateful for your friendship over all these years. Thank you for studying, going on holiday, sharing an apartment, doing research, skiing, writing papers, attending conferences, renting a Toyota Hilux in Argentina, and so much more together with me. My student and PhD life would have been much less fun and productive without you—I am deeply thankful for that.

Sebastian, Benni, and Tim, you have always been my MOVES role models. Thank you for believing in me from the beginning, for teaching me so much, for all your support and advice, for all the collaborations, and for being great friends!

Maxi and Chana, I got to know you both during my time at TUM, which would have been far less fun and successful without you. You have been two important fixed points in my PhD life. I am very grateful for your friendship, your advice, our collaborations, and all the fun we had whenever we met.

My MOVES colleagues, current and former, made these past few years unforgettable. Matthias and Christoph, thank you both for all your advice and support, and for the good times we shared at conferences and workshops. Alex, thank you for the time we spent in Tokyo and for teaching me so much about POMDPs. Lena, thank you for your friendship, the cycling tips, and for the week in Saarbrücken. Hannah, thank you for teaching me all about EVTs and for making me run faster than ever before. Christina, thank you for trusting me to supervise both (!) your bachelor's and master's theses and for helping me with my quantum exam. Ira, thank you for your reliability and all the technical and teaching-related help and support. Roy, thank you for the endless hours at the ping-pong table and for all the fun (trick!). Daniel, thank you for our successful collaboration. Thomas, thank you for your continued support, for the time we shared in Argentina, and for being a great teacher. Philipp, thank you for our memorable trip through Argentina and Chile and for your inspiring work on Caesar. Emma, thank you for the via ferrata adventure and for our stimulating project. Jip, thank you for being a great office mate and ... bedankt dat je me wat Nederlands hebt geleerd! Darion, thank you for being yet another great office mate, for

our collaboration, for all your help with measure theory, Caesar, and so much more, and for all the fun.

I had the pleasure of supervising many brilliant students over last few years. Thank you, Christina, Johannes, Adrian, Tom, Dominik, Diane, Arman, and all the others—I really enjoyed working with you and am proud of your achievements. Keep up the excellent work, and I hope to work with you again soon.

I would also like to thank my colleagues at THS and PLV for creating such a nice research environment and work atmosphere. Lina and Erika, thank you for our collaboration. I greatly enjoyed this work and learned a lot from both of you. Erika, thank you also for chairing my doctoral committee and for giving me valuable opportunities early in my career. Nils, thank you for our joint work, for teaching me about recurrences, and for all the fun we had in Vienna. Jan-Christoph, thank you for all our discussions, for sharing your insights into AST and many other topics, and for our other joint adventures.

I thank the "UnRAVeLers" for all these years of learning together, and for all the talks, lectures, workshops, trips, and countless enjoyable moments. In particular, Birgit (once again) and Helen, I am grateful for all your coordination work and for making UnRAVeL such a wonderful research training group. Komal, thank you for your friendship and for all the good memories.

I met countless inspiring personalities during my PhD journey. Francesca, thank you for visiting us in Aachen. I learned so much during our project, and I hope we can continue our collaboration in the future. I enjoyed all the fun we had at those workshops and conferences. And ... grazie a te ho iniziato a imparare l'italiano! Stefano Maria Nicoletti and Felix Walter, thanks a lot for these unforgettable moments in Argentina! Marcel Moosbrugger, thank you for your inspiring work and for your hospitality in Vienna.

During my PhD, I had the opportunity to visit several institutes around the world. I would like to thank Krishnendu Chatterjee, Maximilian Weininger, Ichiro Hasuo, Benjamin Kaminski, Pedro D'Argenio, Pablo Castro, Pierre Ganty, Chana Weil-Kennedy, and Federico Olmedo for their hospitality, and for the inspiring discussions and collaborations.

Special thanks go to Javier Esparza for serving as the external reviewer of my thesis and for providing valuable feedback. I greatly appreciate the time and expertise you devoted to reviewing my work. I also want to thank Dominique Unruh for being part of my doctoral committee.

Last but not least, I want to thank my family and friends for their unconditional support and encouragement throughout my PhD. Raquel, I am thankful for your support and patience over the last year... y, aunque esto sea un poco cheesy, también por tu amor.

♥ ♥ ♥

Contents

Abstract iii

Zusammenfassung v

Acknowledgements vii

Contents ix

1 Introduction 1
- 1.1 Modeling Probability and Recursion . . . 1
  - 1.1.1 Probabilistic Recursive Programs . . . 1
  - 1.1.2 Program Analysis via Pushdown Automata . . . 2
  - 1.1.3 Certificates . . . 7
- 1.2 Synopsis and Reading Guide . . . 7
- 1.3 Related Work . . . 9
- 1.4 Origins of this Thesis: Prior Publications . . . 10

2 The Weighted Automata Framework 13
- 2.1 Algebraic Foundations . . . 16
  - 2.1.1 Monoids . . . 16
  - 2.1.2 Semirings . . . 19
  - 2.1.3 Formal Power Series . . . 20
  - 2.1.4 Matrices and Vectors . . . 25
- 2.2 Weighted Automata . . . 27
  - 2.2.1 Definition and Basic Examples . . . 27
  - 2.2.2 Semantics of Weighted Automata . . . 29
  - 2.2.3 Behavioral Moments . . . 32
- 2.3 Markov Chains as Weighted Automata . . . 35
  - 2.3.1 Termination Probabilities . . . 36
  - 2.3.2 Expected Runtimes . . . 38
  - 2.3.3 Conditional Expected Runtimes . . . 42
  - 2.3.4 A Taxonomy of Termination . . . 45
- 2.4 Additional Bibliographic Remarks . . . 48

3 Pushdown Automata 49
- 3.1 Weighted Pushdown Automata . . . 51
  - 3.1.1 Formal Definition and Basic Examples . . . 51
  - 3.1.2 Semantics of Weighted Pushdown Automata . . . 54
  - 3.1.3 Behavioral Moments and Generating Functions . . . 63
- 3.2 Probabilistic Pushdown Automata . . . 71
  - 3.2.1 Termination Probabilities of pPDA . . . 71
  - 3.2.2 Expected Runtimes of pPDA . . . 76

3.2.3 Conditional Expected Runtimes of pPDA . . . 80
3.3 Open Questions . . . 83
3.4 Additional Bibliographic Remarks . . . 83

4 Certificates for Probabilistic Pushdown Automata 85
4.1 Positive Polynomial Systems: Background . . . 87
4.1.1 Polynomials . . . 87
4.1.2 The Basics of Positive Polynomial Systems . . . 88
4.1.3 Feasible and Clean PPSs . . . 92
4.1.4 Linear PPSs . . . 95
4.2 Certificates for Positive Polynomial Systems . . . 95
4.2.1 Certificates for Upper Bounds on the LFP . . . 96
4.2.2 Certificates for Lower Bounds on the LFP . . . 110
4.3 Certificates for Probabilistic Pushdown Automata . . . 114
4.3.1 Certificates for Bounds on the Return Probabilities . . . 114
4.3.2 Certificates for PAST . . . 117
4.4 Open Questions . . . 121
4.5 Additional Bibliographic Remarks . . . 122

5 Optimistic Value Iteration for Positive Polynomial Systems 123
5.1 Reduction to Strongly Connected PPSs . . . 125
5.1.1 Irreducible Matrices . . . 125
5.1.2 Strongly Connected PPSs . . . 126
5.1.3 Decomposed PPS Analysis . . . 128
5.2 Perron-Frobenius Cones . . . 130
5.3 The Optimistic Value Iteration Algorithm . . . 133
5.3.1 Subroutines for Lower Bounds and Eigenvectors . . . 135
5.3.2 Correctness and Termination of OVI . . . 137
5.4 OVI in Practice . . . 139
5.4.1 Implementation . . . 140
5.4.2 Experimental Evaluation . . . 143
5.5 Open Questions . . . 146
5.6 Additional Bibliographic Remarks . . . 147

6 Model Checking Temporal Pushdown Specifications 149
6.1 Preliminary Definitions and Results . . . 153
6.1.1 Visibly Pushdown Languages and Automata . . . 153
6.1.2 A Deterministic Automaton Model for ω-VPLs . . . 157
6.1.3 CaRet, a Temporal Logic of Calls and Returns . . . 161
6.2 Probabilistic Visibly Pushdown Automata . . . 163
6.3 Checking pVPA vs Stair Parity DVPA . . . 167
6.3.1 Products of Visibly Pushdown Automata . . . 167
6.3.2 Stair Parity Acceptance Probabilities . . . 168
6.3.3 Reduction to the Existential Theory of the Reals . . . 177
6.4 Checking pVPA vs Büchi VPA and CaRet . . . 182
6.5 Open Questions . . . 184

6.6 Additional Bibliographic Remarks . . . 184

7 From Programs to Automata **187**
7.1 Basic Probabilistic Programs . . . 188
7.1.1 Syntax of Basic Programs . . . 189
7.1.2 Markov Chains as an Operational Model . . . 190
7.1.3 Semantics of Basic Programs . . . 192
7.1.4 Program Verification via Operational DTMCs . . . 195
7.1.5 Examples: Infinite and Exploding State Spaces . . . 197
7.2 Procedural Probabilistic Programs . . . 199
7.2.1 Syntax of Procedural Programs . . . 200
7.2.2 Pushdown Automata as an Operational Model . . . 202
7.2.3 Semantics of Procedural Programs . . . 205
7.2.4 Additional Examples . . . 210
7.3 Open Questions . . . 212
7.4 Additional Bibliographic Remarks . . . 213

8 Conclusion and Outlook **215**

Appendices **217**

A Details on the Experiments **219**
A.1 Probabilistic Program Benchmarks . . . 219
A.2 Comparison of z3 and cvc5 . . . 221

B Omitted Proofs **223**
B.1 Proof of Lemma 3.37 . . . 223
B.2 Proof of Lemma 6.22 . . . 224
B.3 Proof of Lemma 6.28 . . . 226

C List of Co-Authored Publications **233**

Bibliography **237**

List of Abbreviations **249**

Introduction 1

Ever since the early days of computing, scholars have been concerned with the correctness of computer programs. Turing, who in his 1949 article "Checking a Large Routine" [123] laid the groundwork for systematic *program verification*, was among the most prominent voices in this area. Today, as software pervades every aspect of our lives and increasingly complex programs are generated by artificial intelligence (AI), the need for program verification methods is more pressing than ever.

1.1 Modeling Probability and Recursion 1
1.1.1 Probabilistic Recursive Programs 1
1.1.2 Pushdown Automata 2
1.1.3 Certificates 7
1.2 Synopsis and Reading Guide . 7
1.3 Related Work . . 9
1.4 Origins of this Thesis 10

1.1 Modeling Probability and Recursion

We start with a lightweight, example-driven introduction to the main concepts and themes of this thesis.

1.1.1 Probabilistic Recursive Programs

This thesis is concerned with a special class of programs: those involving both *probabilistic behavior* and *recursion*. Such programs have myriad applications; they encode, for instance, randomized algorithms [69] and data structures [111], random generators for combinatorial structures [40], and sophisticated stochastic processes and models in a wide range of disciplines—including, for example, *theory of mind* models in cognitive science [121].

[69]: Hoare (1962), "Quicksort".

[111]: Pugh (1990), "Skip Lists: A Probabilistic Alternative to Balanced Trees".

[40]: Duchon et al. (2004), "Boltzmann Samplers for the Random Generation of Combinatorial Structures".

[121]: Stuhlmüller et al. (2014), "Reasoning about reasoning by nested conditioning: Modeling theory of mind with probabilistic programs".

Illustrative Example: Expression Sampling Let us illustrate probabilistic recursive programs with a simple example.[1] Suppose that we would like to generate random test data[2] for a program which parses and evaluates Boolean expressions. An example expression is as follows:

```
((true or false) and (true or not(false and true))),
```

where "`and`", "`or`", and "`not`" stand for logical conjunction, disjunction, and negation, respectively. Let us—for the sake of simplicity—focus on expressions involving only "`and`", `true`, and `false` in the following.

1: Our example is loosely inspired by a similar one due to Brázdil et al. [22].

2: Testing a program on random inputs is known as *fuzzing* [103].

```
string expr() {
  prob {
    1/4: return "true";
    1/4: return "false";
    1/2: return "(" + expr() + " and " + expr() + ")";
  }
}
```

Figure 1.1: Program generating random Boolean expressions in a Java-like syntax. The `prob` block executes each branch with the given probability. + denotes string concatenation.

A natural approach for generating such expressions is to write a recursive program, as shown in Figure 1.1. Notice that the program effectively samples strings from the *context-free grammar*

$$E \quad \to \quad \texttt{true} \mid \texttt{false} \mid (E \texttt{ and } E)\,, \tag{1.1}$$

where E is the only non-terminal symbol, and the three production rules are taken with probabilities $1/4$, $1/4$, and $1/2$, respectively.

Analysis and Correctness Questions What it means for a program like our sampler from Figure 1.1 to be *correct* depends on the concrete application and the intended use of the program. Typical analysis and verification tasks include the following:

1. *Termination:* Does the program terminate with probability 1?
2. *Expected length:* What is the expected length of the generated expression? Is it finite? If so, is it less than 100?[3]
3. *Probabilities of outcomes:* Does the program generate expressions evaluating to `true` and `false` with equal probability?
4. *Temporal properties:* Does the program generate expressions whose syntax trees have height at most, say, 3 with high probability? (This question depends on the order in which the possible program branches are taken; we thus classify it as a *temporal* one.)

3: This question is essentially equivalent to asking about the *expected runtime* of the program.

In this thesis, we address these questions following an *automata-theoretic* approach, which we outline next.

1.1.2 Program Analysis via Pushdown Automata

Automata are abstract, mathematical *machines* that can be used to model a wide variety of computational processes. Of particular interest in this work are automata equipped with a *stack*.

Stacks A program interpreter or operating system running a recursive program—such as our expression generator from Figure 1.1—must remember *where* a nested procedure call occurred to correctly resume execution in the calling procedure after the nested call returns. This kind of bookkeeping requires a specific data structure: a stack. A stack is a restricted list accessed only from the top end. Specifically, stacks support three operations: *reading* the topmost entry, *popping* (removing) it, and *pushing* (adding) a new entry, which becomes the new top. Pushing and popping correspond to calling and returning from, respectively, nested procedures.

Pushdown Automata *Pushdown automata* (PDA) augment standard automata with a stack, and allow transitions to perform the basic stack operations, such as pushing and popping. The stack is considered of *infinite capacity*, which renders PDA strictly more powerful than finite automata.

The most widespread applications of automata (including PDA) are in the context of recognizing formal languages. In this thesis, however, we also view automata as models of *generative* processes, such as the random behavior of a probabilistic (recursive) program.[4,5] To this end, it is useful to consider automata with additional *weights* assigned to their transitions: *weighted automata*. The special case of a weighted PDA whose weights are probabilities is called a *probabilistic pushdown automaton* (pPDA) [44]—the central model of this thesis.

In the following paragraphs, we illustrate how to tackle Questions 1–4 on the preceding page using automata-theoretic methods.

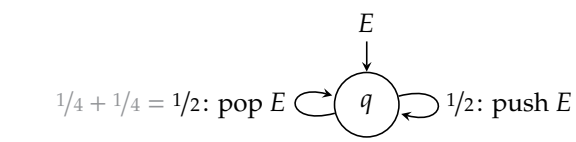
4: The introduction of Chapter 2 compares the two perspectives further.

5: Some authors refer to generative PDA as *pushdown systems* (e. g., [50]).

[44]: Esparza et al. (2004), "Model Checking Probabilistic Pushdown Automata".

Termination and Expected Runtime Regarding Questions 1 (termination) and 2 (expected length/runtime), we can model the program from Figure 1.1 abstractly as the following one-state pPDA, which starts with a stack containing the symbol E once:[6]

6: The initial stack content is indicated by the E next to the initial state marker.



The pPDA mimics sampling from the context-free grammar in Equation (1.1): The two transitions are taken with equal probability of $1/2$ and either pop E from the stack (corresponding to generating `true` or `false`), or push one additional E on the stack (corresponding to rewriting E to "`(` E `and` E `)`"). Thus, the number of E's on the

stack at any point in time corresponds to the number of nonterminals that still need to be expanded.

Using our pPDA model, it is easy to show that the empty stack is eventually reached with probability 1; that is, the program's termination probability is 1. Indeed, it is well-known[7] that the probability of termination is the least nonnegative solution x of the *nonlinear equation*

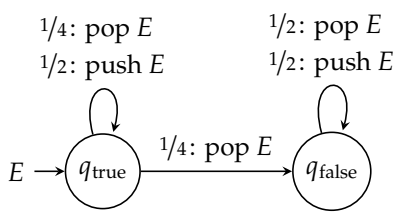
$$x \;=\; \frac{1}{2} + \frac{1}{2}x^2\,, \tag{1.2}$$

which is $x = 1$.[8] Similarly, it can be shown that the expected number of steps until termination is the least nonnegative solution y of the *linear* equation $y = 1 + y$, that is, $y = \infty$. Thus, perhaps somewhat counterintuitively, the program generates expressions of *infinite expected length*, even though it terminates with probability 1. Lowering the probability of generating an "`and`" expression (i.e., of pushing) to any number $p < 1/2$ changes the situation; the pPDA's expected number of steps prior to termination then equals $y = \frac{1}{1-2p} < \infty$, the solution of the linear equation $y = 1 + 2py$.

7: See Chapter 3 (especially Theorem 3.11 on page 58).

8: In this particular example, $x = 1$ is the *unique* solution.

Probabilities of Outcomes Toward answering Question 3 (probability of generating an expression evaluating to `true`), we can refine our pPDA model as follows:



The idea of this pPDA is to keep track of the value of the expression generated thus far. In particular, since we have only Boolean "`and`", the whole expression evaluates to false if and only if `false` occurs at least once. Using the refined model, we can show that the probability of generating an expression evaluating to `true` is

$$1 - \frac{1}{\sqrt{2}} \quad = \quad 0.2928932188\ldots\,,$$

revealing that our program is strongly biased toward expressions evaluating to `false`.

This example also demonstrates that the analysis of probabilistic recursive programs and their pPDA models may involve *irrational* numbers, even when the probabilities in the program are rational. This phenomenon, which is closely related to the nonlinear equations occurring in connection with pPDA,[9] poses a major challenge in the formal analysis of such programs

9: See, e. g., Equation (1.2).

Temporal Properties For Question 4 (temporal properties/height of syntax tree), we first clarify the relevant notions. For example, the expression

```
((true and (false and true)) and true)
```

has a syntax tree of height 3, which is the length of a longest path from the root to a leaf (Figure 1.2 in the margin). In contrast, the expression

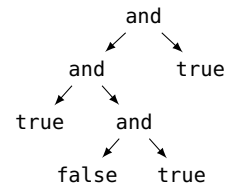


Figure 1.2: Example syntax tree.

```
((true and false) and (false and true))
```

yields a syntax tree with height only 2 (Figure 1.3 in the margin), despite containing the same number of `and` operators as the previous expression.

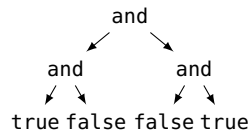
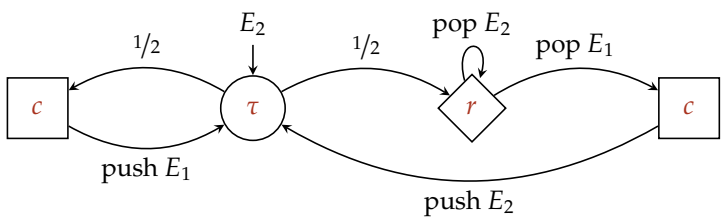


Figure 1.3: Another example syntax tree.

In general, the height of the syntax tree of any expression generated by our sampler from Figure 1.1 (restated for convenience in Figure 1.4) equals the maximum height of the call stack during program execution. Notably, the call stack is not directly related to the stacks of our previous two pPDA models—these automata use the stack for a different purpose, namely to keep track of the number of pending calls. Therefore, we propose the following alternative, more fine-grained pPDA encoding of our example program:[10]

10: For reasons explained further below, we graphically distinguish between three types of states: *push states* (rectangles), and *pop states* (diamonds), and states ignoring the stack (circles).



This pPDA uses *two* stack symbols, E_1 and E_2, which intuitively represent the following:

– E_1 stands for the *first* invocation of `expr()` in the recursive branch of the program (the one with probability `1/2`). This invocation

```
string expr() {
 prob {
  1/4: return "true";
  1/4: return "false";
  1/2: return "(" +
     expr() + " and "
     + expr() + ")";
 }
}
```

Figure 1.4: Figure 1.1 restated.

is special because after it returns, a second call to `expr()` is guaranteed to occur.
– Similarly, E_2 symbolizes the *second* invocation of `expr()` in the recursive branch. (E_2 also represents the initial invocation of `expr()` used to launch the program.)

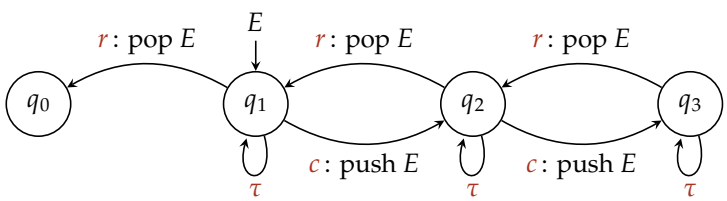
Constructing such models by hand for more complex programs is tedious and error-prone. Therefore, in Chapter 7, we present a general, effective method for translating probabilistic recursive programs to pPDA.[11]

11: In Example 7.22 on page 210, we *automatically* construct a pPDA similar to the one discussed here.

With the revised automaton model at hand, how can we characterize the probability of the stack height not exceeding 3? Our approach is to first decorate our pPDA with additional *state labels* (colored symbols inside the states). These labels are then fed as inputs to *another automaton* that monitors the current stack height. In this thesis, we employ *visibly pushdown automata* (VPA) [3]—a restricted class of traditional PDA—as monitors.[12] The following VPA accepts exactly the sequences of state labels corresponding to program executions in which the call stack height is at most 3 (acceptance occurs upon emptying the stack):

[3]: Alur et al. (2004), "Visibly pushdown languages".

12: Non-pushdown automata can also be used (e.g., [86]), but VPA are more expressive.



The *visibility restriction* requires each input symbol to trigger a fixed stack operation: in our example, c triggers a push, r triggers a pop, and τ leaves the stack unchanged.

Following standard automata-based verification techniques, we then compute a *product* of the pPDA and the monitoring VPA (omitted here for brevity). For the product to be well-defined, the two automata must synchronize their stack operations. This is only possible if, in the pPDA, all states labeled with c (for example) have only outgoing push transitions, and similarly for states labeled with r and τ. pPDA satisfying this (minor) restriction are called *probabilistic visibly pushdown automata* (pVPA) in this thesis.

Algorithmic model checking of pVPA against pushdown specifications—specifically, against the class of *ω-visibly pushdown languages* [3]—is among the main contributions of this thesis.[13]

13: We have published these findings previously in FoSSaCS and LMCS [131, 132]. See Section 1.4 for details on prior publications.

1.1.3 Certificates

This thesis places special emphasis on *certificates* for various pPDA properties. A certificate is a lightweight, easy-to-check *proof* of a claimed property. Ideally, certificates are produced as a by-product of a *certifying algorithm*—an algorithm that both decides whether a property holds and outputs a certificate substantiating its decision [96]. Certificates enhance the trustworthiness of formal verification tools, which are typically not themselves formally verified and may therefore contain errors such as implementation bugs and numerical instabilities due to floating-point arithmetic [82].

[96]: McConnell et al. (2011), "Certifying algorithms".

[82]: Kohlen et al. (2025), "A Formally Verified IEEE 754 Floating-Point Implementation of Interval Iteration for MDPs".

As a main contribution, we develop a certification mechanism for *positive almost sure termination* (PAST; termination with finite expected runtime) of pPDA.[14] These certificates are *complete* in the sense that every pPDA with finite expected runtime admits such a certificate. Our certificates comprise suitable *rational* bounds on the solutions of the nonlinear and linear equation systems arising from the pPDA.[15] Chapter 4 and Chapter 5 cover the theoretical and practical (i.e., algorithmic and empirical) aspects of certificates, respectively.[16]

14: We published this result in a prior paper at LICS [135].

15: See Section 1.1.2 for simple examples of such equations.

16: We presented the practical results of Chapter 5 in a paper at TACAS [134].

1.2 Synopsis and Reading Guide

In the following, we provide a brief overview of the main chapters of this thesis. Each of them contains additional, chapter-specific introductory material, questions left open for future work, and discussion of related literature.

The chapters are generally designed to be as self-contained as possible. Figure 1.5 reveals some unavoidable dependencies. Nonetheless, readers with appropriate background are invited to read the chapters in any order. To facilitate this, references to the relevant material from other chapters are placed as *side notes*[17] in the margin.

17: This is a side note.

Chapter 2: The Weighted Automata Framework This chapter introduces *weighted automata* over semirings as a foundational model for quantitative computational processes. We focus particularly on the semiring of nonnegative real numbers, which enables modeling of probabilistic processes and programs.

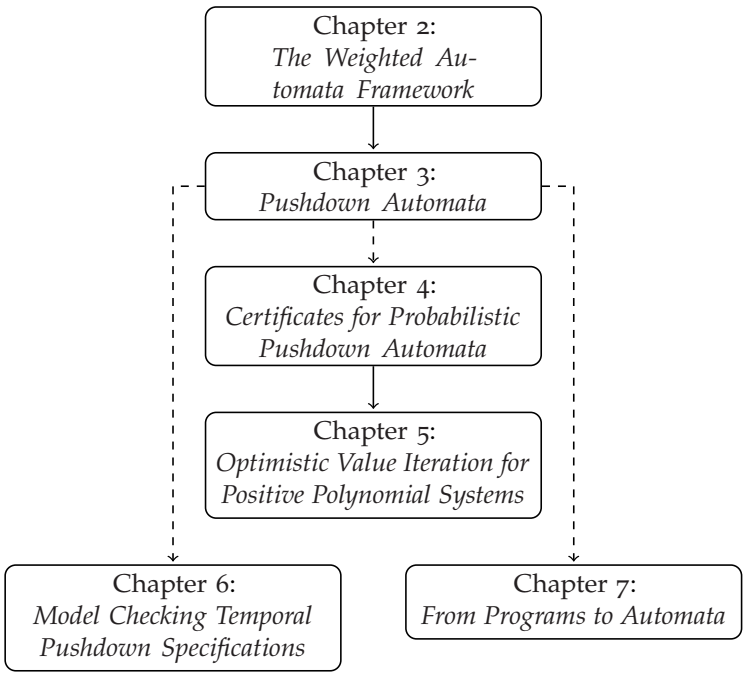


Figure 1.5: Main chapters and their dependencies. *Legend:* Solid arrows: strong dependencies; dashed arrows: weak dependencies.

Chapter 3: Pushdown Automata This chapter extends the weighted automata from Chapter 2 with a stack, yielding *weighted pushdown automata* (WPDA). We present several equation systems that characterize a range of quantitative properties of WPDA and, as a special case, of pPDA.

Chapter 4: Certificates for Probabilistic Pushdown Automata This chapter develops the theoretical foundations of our pPDA certificates. Its central result characterizes the existence of *self-certifying* (also called *inductive*) bounds on the relevant solutions of the nonlinear equations derived from a pPDA.

Chapter 5: Optimistic Value Iteration for Positive Polynomial Systems This chapter presents a certifying algorithm for computing the certificates introduced in Chapter 4, along with experimental results.

Chapter 6: Model Checking Temporal Pushdown Specifications This chapter investigates the algorithmic verification of (a mildly restricted variant of) pPDA against *pushdown specifications*. These specifications are encoded either as *CaRet formulas* [2] or as VPA

[2]: Alur et al. (2004), "A Temporal Logic of Nested Calls and Returns".

(visibly pushdown automata). The main contributions are decidability and complexity results for the corresponding model checking problems.

Chapter 7: From Programs to Automata This chapter formalizes a concrete translation from probabilistic recursive programs to pPDA, similar to the examples in Section 1.1.2. The translation has been implemented in our tool Pray.

1.3 Related Work

Here, we provide only a high-level overview of existing work on probabilistic (recursive) programs and pPDA. Each individual chapter contains further, more specific related work discussions.

Kozen was among the first to study formal semantics [85] and verification [83] of probabilistic programs. An early account of *automata-based* verification of probabilistic programs is due to Vardi [125]. Many impactful papers have followed since these pioneering works—too many to list them all here. Good starting points for further reading include McIver and Morgan's monograph [97], Chapter 10 of *Principles of Model Checking* [8], Baier et al.'s chapter in the *Handbook of Model Checking* [9, Chapter 28], Kaminski's PhD thesis [74], and the more recent volume *Foundations of Probabilistic Programming* [11].

Since the early to mid 2010's, there has been considerable interest in (fully or semi-) *automatic* probabilistic program analysis—we mention the work by Katoen et al. [78] and Chakarov and Sankaranarayanan [25] as early representatives, and that of Schröer et al. [118] and Batz [12] as recent developments. Additionally, the modeling languages used in *probabilistic model checkers* such as PRISM [88] and Storm [67], which support the fully automatic verification of certain probabilistic finite-state models, closely resemble probabilistic programming languages.

There also exists a large body of literature and several well-maintained tools on *sampling* probabilistic programs efficiently (e.g., [24, 58, 124]), usually with the goal of performing *Bayesian inference*. However, such approaches typically yield—if any—statistical guarantees only, which is in contrast to the hard, mathematical guarantees that we focus on.

[85]: Kozen (1979), "Semantics of Probabilistic Programs".

[83]: Kozen (1983), "A Probabilistic PDL".

[125]: Vardi (1985), "Automatic Verification of Probabilistic Concurrent Finite-State Programs".

[78]: Katoen et al. (2010), "Linear-Invariant Generation for Probabilistic Programs: - Automated Support for Proof-Based Methods".

[25]: Chakarov et al. (2013), "Probabilistic Program Analysis with Martingales".

[118]: Schröer et al. (2023), "A Deductive Verification Infrastructure for Probabilistic Programs".

[12]: Batz (2024), "Automated deductive verification of probabilistic programs".

[88]: Kwiatkowska et al. (2011), "PRISM 4.0: Verification of Probabilistic Real-Time Systems".

[67]: Hensel et al. (2022), "The probabilistic model checker Storm".

[24]: Carpenter et al. (2017), "Stan: A probabilistic programming language".

[58]: Goodman et al. (2014), *The Design and Implementation of Probabilistic Programming Languages*.

[124]: van de Meent et al. (2018), "An Introduction to Probabilistic Programming".

In the remainder, we discuss more specific related work on the formal, that is, mathematically rigorous, verification of probabilistic *recursive* programs.

Olmedo et al. [105] provide, among other contributions, a formal program semantics in terms pPDA. The tool APROVE can automatically analyze recursive probabilistic programs modeled as *probabilistic term rewriting systems* [77]. Chiang, McDonald, and Shan [30] study *exact inference* in recursive probabilistic programming.

pPDA have long been recognized as a natural model for probabilistic recursive programs. Many aspects of the formal verification and analysis of pPDA and related models—such as *recursive Markov chains* [49]—have been studied extensively since the mid 2000's [20–22, 44, 45, 48–50, 86];[18] also see Wojtczak's and Stewart's PhD theses [120, 141]. The relation to these works is discussed in more detail in the respective chapters.

Finally, recursive models combining *nondeterministic and probabilistic* branching [51], as well as models for probabilistic higher-order recursion [81], have also been investigated, but are beyond the scope of this thesis.

[105]: Olmedo et al. (2016), "Reasoning about Recursive Probabilistic Programs".

[77]: Kassing et al. (2025), "From Innermost to Full Probabilistic Term Rewriting: Almost-Sure Termination, Complexity, and Modularity".

[30]: Chiang et al. (2023), "Exact Recursive Probabilistic Programming".

[20]: Brázdil et al. (2005).

[21]: Brázdil et al. (2013).

[22]: Brázdil et al. (2015).

[44]: Esparza et al. (2004).

[45]: Esparza et al. (2005).

[48]: Etessami et al. (2012).

[49]: Etessami et al. (2005).

[50]: Etessami et al. (2009).

[86]: Kucera et al. (2006).

18: The list is non-exhaustive.

[51]: Etessami et al. (2015), "Recursive Markov Decision Processes and Recursive Stochastic Games".

[81]: Kobayashi et al. (2020), "On the Termination Problem for Probabilistic Higher-Order Recursive Programs".

1.4 Origins of this Thesis: Prior Publications

The following peer-reviewed prior publications are covered in detail in this thesis (listed in chronological order):

1. Tobias Winkler, Christina Gehnen, and Joost-Pieter Katoen. "Model Checking Temporal Properties of Recursive Probabilistic Programs". In: *FoSSaCS*. Vol. 13242. Lecture Notes in Computer Science. Springer, 2022

2. Tobias Winkler and Joost-Pieter Katoen. "Certificates for Probabilistic Pushdown Automata via Optimistic Value Iteration". In: *TACAS (2)*. Vol. 13994. Lecture Notes in Computer Science. Springer, 2023

3. Tobias Winkler and Joost-Pieter Katoen. "On Certificates, Expected Runtimes, and Termination in Probabilistic Pushdown Automata". In: *LICS*. IEEE, 2023

4. Tobias Winkler, Christina Gehnen, and Joost-Pieter Katoen. "Model Checking Temporal Properties of Recursive Probabilistic Programs". In: *Log. Methods Comput. Sci.* 19.4 (2023)

The content of Publication 1 and its extended journal version, Publication 4, are covered in Chapter 6. Publication 2 is covered in Chapter 5, and most of Publication 3 is covered in Chapters 3 and 4.

Publication 1 was inspired by Christina Gehnen's B. Sc. thesis [55], for which I served as a consultant. I would like to thank Christina for her outstanding work, and Christof Löding for providing her with crucial pointers to the literature.

[55]: Gehnen (2021), "Automata-based model checking of recursive systems".

During my time as a scientific employee and doctoral student at RWTH Aachen University, I have co-authored several additional peer-reviewed publications.[19] While these papers are not discussed in detail in this thesis, they have naturally influenced the ideas presented here. References are placed throughout the text where appropriate.

19: See Appendix C on page 233 for a complete list.

The Weighted Automata Framework

2

This chapter introduces a fairly general variant of *weighted automata* [87] as a common foundation for the subsequent chapters. We start with a lightweight introduction; the formal development begins in Section 2.1 on page 16.

2.1 Foundations . . . 16
2.1.1 Monoids 16
2.1.2 Semirings 19
2.1.3 Formal Power Series 20
2.1.4 Matrices and Vectors 25
2.2 Weighted Automata 27
2.2.1 Definition 27
2.2.2 Semantics 29
2.2.3 Behavioral Moments 32
2.3 Markov Chains . 35
2.3.1 Termination Probabilities . . . 36
2.3.2 Expected Runtimes 38
2.3.3 Conditional Expected Runtimes 42
2.3.4 A Taxonomy of Termination . . . 45
2.4 Bibliographic Remarks 48

Automata as Reactive Machines Traditionally, automata are abstract machines used to distinguish desirable input words from undesirable ones. They do so by processing the input symbol-by-symbol. Each consumed symbol prescribes a set of *transitions* according to which the automaton updates its *state*. To put it short:

Automaton = Device *reacting* to input symbols

The reactive metaphor is commonly applied to both deterministic and nondeterministic automata. For the latter, *reacting* may include *guessing*.

In this thesis, however, we rarely adopt the traditional, reactive view—indeed, *most automata we consider do not have inputs*; the only exceptions are in Chapter 6.

The Generative Perspective Instead, our perception of automata is similar to their interpretation as *generative* processes (e. g., [71]).[1] In this view, an automaton may take any of its transitions available in the current state in a purely nondeterministic fashion. An external stimulus or input, as in the reactive paradigm, is not required. We call the sequences of states traversed by possible transitions *computation paths*. The generative interpretation then amounts to:

Automaton = Device for *generating* computation paths

The reactive and generative interpretations may seem vastly different at first glance. In fact, other authors have emphasized that "problems studied [...] for reactive probabilistic automata do not make much sense for Segala [generative] automata" [52]. Nonetheless, we observe that generative automata subsume reactive ones in the following sense. Consider, for instance, a standard deterministic finite automaton (DFA) over an input alphabet with at least two symbols. Notably, from the generative point of view, such a DFA is not at all deterministic: At every state, there are multiple (at least

[71]: Icard (2020), "Calibrating generative models: The probabilistic Chomsky–Schützenberger hierarchy".

1: We use the term *generative* broadly, not just for probabilistic models.

[52]: Fijalkow (2017), "Undecidability results for probabilistic automata".

two) available transitions, each of which may be taken nondeterministically. However, it is well-known that a DFA accepts a word w if and only if there exists a computation path from the initial to a final state inscribed with w. It is therefore justified to say that a DFA *generates* precisely the set of words it accepts.

Weighted Automata Our notion of weighted automata takes the generative paradigm to its extreme. Such automata can generate many kinds of mathematical objects—they are not limited to generating formal languages, as in the case of the DFA from the previous paragraph. This flexibility is achieved by allowing more general objects—not just symbols from an alphabet—as *weights* on the transitions. In a nutshell, weighted automata resemble *weighted directed graphs*, possibly equipped with additional *initial* and *final state weights*. This general notion of weighted automata, surveyed by Kuich [87], is even somewhat more abstract than much of the existing weighted automata theory, which often focuses on *weighted formal languages* (e.g., [42]).

[87]: Kuich (1997), "Semirings and Formal Power Series: Their Relevance to Formal Languages and Automata".

[42]: Ésik et al. (2009), "Finite Automata".

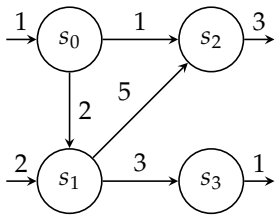
A Weighted Automata Appetizer Before diving into the formal details, let us provide a taste of the mechanics of weighted automata. Figure 2.1 on the following page depicts a weighted automaton with four states. Weights occur in three different places: on *transitions* (arrows between states), as *initial weights* (dangling incoming arrows), and as *final weights* (dangling outgoing arrows).[2] In our example, the weights are nonnegative integers. In general, as mentioned, a wide range of objects can be used as weights, including probabilities, formal power series, and matrices.

2: From now on, we often omit unit weights in the graphical representations of weighted automata.

To define the mathematical meaning—also called semantics or *behavior*—of a general weighted automaton, it is necessary to adopt the generative perspective and inspect the automaton's *computation paths*. A computation path is any finite sequence of states from a state with nonzero initial weight to a state with nonzero final weight traversing nonzero-weight transitions only. The table in Figure 2.1 lists these paths together with their associated weights for our example automaton. We remark that, in the presence of cycles, even a *finite* weighted automaton has *infinitely* many computation paths—this makes the theory somewhat more involved.

Much of the power and flexibility of weighted automata comes from aggregating the computation paths and their weights in a specific manner. Two aggregator functions must be defined:

– A *product* operator aggregates the weights along individual paths.



Computation path	Weights
(s_0, s_2)	$1, 1, 3$
(s_0, s_1, s_2)	$1, 2, 5, 3$
(s_0, s_1, s_3)	$1, 2, 3, 1$
(s_1, s_2)	$2, 5, 3$
(s_1, s_3)	$2, 3, 1$

Figure 2.1: A weighted automaton with nonnegative integer weights and its computation paths.

– A *sum* operator aggregates the collective weight of all paths.

The two operators have to satisfy some axioms so that they form a *semiring*[3] on the set of weights. Choosing product and sum as standard integer multiplication and addition, respectively, the behavior (semantics) of our example automaton is the integer 75, because

$$1 \cdot 1 \cdot 3 \;+\; 1 \cdot 2 \cdot 5 \cdot 3 \;+\; 1 \cdot 2 \cdot 3 \cdot 1 \;+\; 2 \cdot 5 \cdot 3 \;+\; 2 \cdot 3 \cdot 1 \\ = 75\,.$$

3: See Section 2.1.2 on page 19 for the formal definitions.

This quantity is the sum of the products of the weights of all computation paths. Alternatively, we could define the *product as standard addition*, and the *sum as the binary minimum operator* $\sqcap$. This results in a behavior of 5, since[4]

$$1{+}1{+}3 \;\sqcap\; 1{+}2{+}5{+}3 \;\sqcap\; 1{+}2{+}3{+}1 \;\sqcap\; 2{+}5{+}3 \;\sqcap\; 2{+}3{+}1 \\ = 5\,.$$

4: We assume that $\sqcap$ binds stronger than $+$.

Notably, 5 is the weight of a *shortest path* through the automaton. This style of encoding relevant quantities as behaviors of weighted automata is a recurring theme in our work.

Chapter Outline, Contributions, and Origins The remainder of this chapter is structured as follows:

Section 2.1 (Algebraic Foundations) We introduce semirings, formal power series, and matrices. The presented material is mostly standard [36, 87].

[36]: Droste et al. (2009).
[87]: Kuich (1997).

Section 2.2 (Weighted Automata) We formally introduce weighted automata and their semantics, loosely following Kuich [87]. We also define a—to our knowledge novel—variation of an automaton's behavior that we call *behavioral moments* (Definition 2.32).

Section 2.3 (Markov Chains as Weighted Automata) We express quantitative properties of (discrete-time) *Markov chains* as behaviors of carefully constructed weighted automata. This section relies in part on one of our previous publications [135].

[135]: Winkler et al. (2023), "On Certificates, Expected Runtimes, and Termination in Probabilistic Pushdown Automata".

Additional bibliographic remarks are deferred to Section 2.4.

2.1 Algebraic Foundations

As mentioned in the introductory paragraphs of this chapter, weighted automata are defined relative to a *semiring*, an algebraic structure prescribing the available weights and their aggregation operations. The main goal of this section is to introduce semirings formally (Sections 2.1.1 and 2.1.2), and provide constructions for obtaining new semirings from existing ones (Sections 2.1.3 and 2.1.4).

2.1.1 Monoids

We will see in Section 2.1.2 that a semiring is a combination of two *monoids*, a simpler kind of structure which is defined as follows:

Definition 2.1 ***(Monoid)*** A *monoid* is a triple $(W, +, 0)$ where

- W is a nonempty *carrier set*,
- $+ \colon W \times W \to W$ is an associative binary operation, and
- $0 \in W$ is *neutral* for $+$, i.e., $\forall w \in W \colon w + 0 = w = 0 + w$.[5] △

5: The definition implies that 0 is in fact the *unique* neutral element of the monoid.

► **Example 2.2** (A selection of basic monoids) It can be verified that all of the following are monoids:

- $\mathbb{N}, \mathbb{Z}, \mathbb{Q}$, and $\mathbb{R}$ with addition and neutral element zero, as well as with multiplication and neutral element one.
- $(\overline{\mathbb{R}}_{\geq 0}, +, 0)$, where $\overline{\mathbb{R}}_{\geq 0} = \mathbb{R}_{\geq 0} \cup \{\infty\}$ are the *extended* nonnegative reals, and where $r + \infty = \infty + r = \infty$ for all $r \in \overline{\mathbb{R}}_{\geq 0}$.[6]
- $(\overline{\mathbb{R}}_{\geq 0}, \cdot, 1)$ with the convention that $r \cdot \infty = \infty \cdot r = \infty$ for all $r > 0$, and $0 \cdot \infty = \infty \cdot 0 = 0$.
- $(\Sigma^*, \cdot, \varepsilon)$, where Σ is a finite alphabet, $\cdot$ denotes concatenation of finite words from Σ^*, and ε is the empty word.
- $(2^A, \cup, \emptyset)$, the subsets of an arbitrary set A with union.
- $(\mathbb{B}, \vee, \mathsf{false})$ and $(\mathbb{B}, \wedge, \mathsf{true})$, the set of truth values with disjunction and conjunction, respectively.

6: The symbol ∞ is a formal element of $\overline{\mathbb{R}}_{\geq 0}$. It is *not* just a placeholder for an arbitrarily large number.

The two monoids $(\overline{\mathbb{R}}_{\geq 0}, +, 0)$ and $(\overline{\mathbb{R}}_{\geq 0}, \cdot, 1)$ are of particular interest in this thesis; together, they form the important *semiring of extended nonnegative real numbers* (to be formally introduced in Section 2.1.2). ◄

As usual, an *isomorphism* between two monoids is a bijection between the two carrier sets that preserves $+$ and 0. For example, $(\Sigma^*, \cdot, \varepsilon)$ with $|\Sigma| = 1$ is isomorphic to $(\mathbb{N}, +, 0)$, and $(\mathbb{B}, \vee, \mathsf{false})$ is isomorphic to $(\mathbb{B}, \wedge, \mathsf{true})$ by virtue of negation and De Morgan's laws.

The following additional monoid properties will be of interest:

Definition 2.3 ***(Types of monoids)*** Let $(W, +, 0)$ be a monoid.

A. $(W, +, 0)$ is *commutative* if $+$ is commutative.
B. $(W, +, 0)$ is *naturally ordered* if it is commutative and the binary relation $\sqsubseteq$ defined by

$$w_1 \sqsubseteq w_2 \quad \Longleftrightarrow \quad \exists w \in W : w_1 + w = w_2$$

is a partial order[7] on W.
C. $(W, +, 0)$ is *ω-continuous* if it is naturally ordered and all its *ω-chains*, that is, nondecreasing infinite sequences

$$w_0 \sqsubseteq w_1 \sqsubseteq w_2 \sqsubseteq \dots ,$$

where $w_0, w_1, \dots \in W$, have a *supremum* (least upper bound) $\sup_{i\in\mathbb{N}} w_i$ in W.[8] Furthermore, the monoid operation is required to preserve these suprema: For all $w \in W$, it must hold that

$$w + \sup_{i\in\mathbb{N}} w_i = \sup_{i\in\mathbb{N}} (w + w_i) . \qquad \triangle$$

7: A *partial order* is a reflexive, transitive, and antisymmetric binary relation.

8: This is equivalent to requiring that $(W, \sqsubseteq)$ is an *(ω-)complete partial order* [94, 140].

In a naturally ordered monoid $(W, +, 0)$, it follows from the definitions that $+$ preserves the order $\sqsubseteq$, that is,

$$\forall w_1, w_2, w \in W : \quad w_1 \sqsubseteq w_2 \implies w_1 + w \sqsubseteq w_2 + w .$$

Moreover, $0 \in W$ is the unique least element with regards to $\sqsubseteq$.

It is instructive to notice that *nonzero elements do not have inverses* in naturally ordered monoids. Indeed, assume that w has an inverse $-w$, that is, $w + (-w) = 0$. But then $w \sqsubseteq 0$ by definition of the order, and therefore $w = 0$ since 0 is the least element. The term *semiring*[9] points to this observation—intuitively, a semiring resembles a ring in which the *negative half* of the elements may not exist.

9: To be defined in Section 2.1.2 on page 19.

► **Example 2.4** (Commutative, naturally ordered, and ω-continuous) Reconsider the monoids from Example 2.2.

- All of these monoids except $(\Sigma^*, \cdot, \varepsilon)$ with $|\Sigma| \geq 2$ are commutative.
- Some of the commutative monoids, such as $(\mathbb{Z}, +, 0)$ and $(\overline{\mathbb{R}}_{\geq 0}, \cdot, 1)$, have nonzero elements with inverses; therefore, they are *not* naturally ordered.
- The naturally ordered monoids include $(\mathbb{N}, +, 0)$, $(\overline{\mathbb{R}}_{\geq 0}, +, 0)$, $(2^A, \cup, \emptyset)$, and the two Boolean monoids $(\mathbb{B}, \vee, \mathsf{false})$ and $(\mathbb{B}, \wedge, \mathsf{true})$. The natural orders of these five examples coincide with $\leq, \leq, \subseteq, \Rightarrow$, and $\Leftarrow$, respectively.
- The monoids $(\overline{\mathbb{R}}_{\geq 0}, +, 0)$, $(2^A, \cup, \emptyset)$, $(\mathbb{B}, \vee, \mathsf{false})$, and $(\mathbb{B}, \wedge, \mathsf{true})$ are moreover ω-continuous. $(\mathbb{N}, +, 0)$, on the other hand, is *not* ω-continuous is because some of its ω-chains do not have a supremum in $\mathbb{N}$. More

precisely, an ω-chain in $(\mathbb{N}, +, 0)$ has a supremum if and only if the chain stabilizes eventually. For example, the infinitely ascending chain

$$0 \leq 1 \leq 2 \leq 3 \leq \ldots$$

does *not* have a supremum in $\mathbb{N}$. In $(\overline{\mathbb{R}}_{\geq 0}, +, 0)$, on the other hand, the supremum of the above chain is the ∞-element. ◀

Next, we define infinite sums in monoids, which will immediately carry over to semirings. Infinite sums are key to sensibly define the behavior of weighted automata with infinitely many computation paths (see Section 2.2.2).

Definition 2.5 ***(Infinite sums in monoids)*** Let $(W, +, 0)$ be an ω-continuous monoid. For every infinite sequence $w_0, w_1, \ldots \in W$ we define the *infinite sum*

$$\sum_{i \in \mathbb{N}} w_i \;=\; \sup_{n \in \mathbb{N}} \sum_{i=0}^{n} w_i \,. \qquad \triangle$$

Notice that the supremum on the right hand side of the equation in Definition 2.5 exists because the *partial sums*

$$w_0 \quad \sqsubseteq \quad w_0 + w_1 \quad \sqsubseteq \quad w_0 + w_1 + w_2 \quad \sqsubseteq \quad \ldots$$

constitute an ω-chain.

Lemma 2.6 *Infinite sums in monoids are commutative.*

Proof. Consider an arbitrary reordering $w'_0, w'_1, \ldots$ of the original sequence.[10] We will show the following statement:

$$\forall k \in \mathbb{N}\colon \quad \sum_{i=0}^{k} w'_i \;\sqsubseteq\; \sup_{n} \sum_{i=0}^{n} w_i \,. \tag{2.1}$$

10: Formally, this means that we consider an arbitrary bijection $\tau\colon \mathbb{N} \to \mathbb{N}$ and define $w'_i = w_{\tau(i)}$ for all $i \in \mathbb{N}$.

Equation (2.1) implies $\sum_{i \in \mathbb{N}} w'_i \sqsubseteq \sum_{i \in \mathbb{N}} w_i$ by definition of the supremum, and thus the claim since the reverse inequality holds by symmetry.

To show Equation (2.1), let $k \in \mathbb{N}$. We can pick n sufficiently large such that the elements $w'_0, \ldots, w'_k$ are fully contained in $w_0, \ldots, w_n$ (in the sense of multiset inclusion, i.e., with duplicates). By definition of the natural order, it follows that $\sum_{i=0}^{k} w'_i \sqsubseteq \sum_{i=0}^{n} w_i \sqsubseteq \sup_n \sum_{i=0}^{n} w_i$. □

Lemma 2.6 justifies notations such as $\sum_{w \in V} w$, where V is a countable set (or multiset) of elements from some ω-continuous monoid.

2.1.2 Semirings

We are now ready to define semirings, the structures in which the weights of a weighted automaton reside.

Definition 2.7 ***(Semiring)*** A *semiring* is a 5-tuple $\mathcal{S} = (W, +, \cdot, 0, 1)$, where:

- $(W, +, 0)$ is a commutative monoid[11] whose operation $+$ is called *addition*.
- $(W, \cdot, 1)$ is a monoid whose operation $\cdot$ is called *multiplication* and is often denoted by juxtaposition.[12]
- Multiplication *distributes* over addition: For all $w, v_1, v_2 \in W$,

$$w(v_1 + v_2) \;=\; wv_1 + wv_2 \quad \text{and} \quad (v_1 + v_2)w \;=\; v_1 w + v_2 w\,.$$

- Multiplication by 0 *annihilates*, that is, for all $w \in W$, we have $0w = w0 = 0$. △

11: See Definition 2.1 on page 16.

12: Unlike addition, multiplication is possibly non-commutative.

► **Example 2.8** (Selected semirings) It can be verified that all of the following are semirings:

- $\overline{\mathcal{R}}_{\geq 0} = (\overline{\mathbb{R}}_{\geq 0}, +, \cdot, 0, 1)$, the *semiring of extended nonnegative real numbers* with addition and multiplication as defined in Example 2.2 on page 16.
- $\mathcal{L}_\Sigma = (2^{\Sigma^*}, \cup, \cdot, \emptyset, \{\varepsilon\})$, the *semiring of formal languages* over a finite alphabet Σ. Here, $\cdot$ denotes standard word-wise language concatenation; for example,

$$\{a, ab\} \cdot \{a, ba\} \;=\; \{aa, aba, abba\}\,.$$

- $\mathcal{B} = (\mathbb{B}, \vee, \wedge, \mathsf{false}, \mathsf{true})$, the *Boolean semiring*.

We often consider weighted automata over $\overline{\mathcal{R}}_{\geq 0}$ in this thesis. .....◄

The following parallels Definition 2.3 on page 17:

Definition 2.9 ***(Types of Semirings)*** Let $\mathcal{S} = (W, +, \cdot, 0, 1)$ be a semiring.

- A. $\mathcal{S}$ is *commutative* if $(W, \cdot, 1)$ is commutative.
- B. $\mathcal{S}$ is *naturally ordered* if $(W, +, 0)$ is naturally ordered.[13]
- C. $\mathcal{S}$ is *ω-continuous* if $(W, +, 0)$ is ω-continuous[14] and multiplication preserves suprema of ω-chains; that is, for all ω-chains $w_0 \sqsubseteq w_1 \sqsubseteq \ldots \in W$ and $w \in W$, we have

$$w \cdot \sup_{i \in \mathbb{N}} w_i \;=\; \sup_{i \in \mathbb{N}} (w \cdot w_i)$$
$$\text{and} \quad \Big(\sup_{i \in \mathbb{N}} w_i\Big) \cdot w \;=\; \sup_{i \in \mathbb{N}} (w_i \cdot w)\,. \qquad \triangle$$

13: See Definition 2.3B.

14: See Definition 2.3C. Our definition of ω-continuous semirings slightly differs from Kuich's [87]. See Section 2.4 for details.

It follows from the definitions that multiplication—just like addition—preserves the natural order relation $\sqsubseteq$ in any semiring.[15]

15: This holds even if the semiring is not naturally ordered (i. e., if the relation $\sqsubseteq$ is not actually a partial order).

For the following definition, recall that infinite sums in an ω-continuous semiring are defined by virtue of Definition 2.5 on page 18. Note that, by Definition 2.9c, the distributivity of multiplication over addition extends to such infinite sums as well.

Definition 2.10 ***(Star)*** Let $(W, +, \cdot, 0, 1)$ be an ω-continuous semiring. For every $w \in W$, we define the *star* of w as

$$w^* \;=\; w^0 + w^1 + w^2 + \ldots \;=\; \sum_{n \in \mathbb{N}} w^n \;,$$

where, as usual, $w^0 = 1$ and $w^{n+1} = w^n \cdot w$ for all $n \in \mathbb{N}$. △

16: More abstractly, semirings with an additional unary star operator—not necessarily identical to the one from Definition 2.10—are called *star semirings* in the literature (e. g., [36]).

It follows that the so-defined star operator[16] satisfies

$$w^* \;=\; 1 + w \cdot w^* \;=\; 1 + w^* \cdot w \;.$$

In the semiring of formal languages, the star operator is the classic *Kleene star* (or iteration) of a formal language. In $\overline{\mathcal{R}}_{\geq 0}$, the extended nonnegative real semiring, the star is the limit of the corresponding *geometric series* if the series converges, and ∞ if the series diverges. More precisely, we have $r^* = \frac{1}{1-r}$ for $r \in [0, 1)$, and $r^* = \infty$ for $r \in [1, \infty]$.

2.1.3 Formal Power Series

Loosely following Kuich [87], we now introduce constructions that allow obtaining new semirings from existing ones. We start with *formal power series* (FPS).

Traditionally, an FPS is an infinite *formal sum*

$$\sum_{n=0}^{\infty} r_n z^n \;,$$

where z is a formal variable, also called *indeterminate*, and the *coefficients* r_n are numbers. For example,

$$\frac{1}{2} \;+\; \frac{1}{4} z \;+\; \frac{1}{8} z^2 \;+\; \frac{1}{16} z^3 \;+\; \ldots$$

is an FPS.[17] The attribute *formal* indicates that we are usually not concerned with questions about convergence of the series.

17: This particular example has the special property that the coefficients are nonnegative and sum up to 1—such FPS are called *probability generating functions* (PGF). They play a role later on in Section 3.1.3.

We present a more abstract view on traditional FPS, which will also lead to their generalization. The first and most important observation is that an FPS is really just a mapping from the set of *monomials* $\mathsf{Mon}[z] = \{z^n \mid n \in \mathbb{N}\}$ to the domain of the coefficients; that is, each monomial is assigned a unique coefficient (possibly zero). Second, we observe that the coefficients, which are numbers, can be both added *and* multiplied. In contrast, monomials can only be multiplied, but not added—adding two monomials yields a *polynomial*. These observations lead to the following generalization, which is standard in automata theory [87, 117]:

[87]: Kuich (1997).
[117]: Sakarovitch (2009).

$$\text{FPS} \;=\; \text{Map of type } \textit{monoid} \to \textit{semiring}$$

Notice that $(\mathsf{Mon}[z], \cdot, 1)$, where $\cdot$ denotes standard multiplication and $1 = z^0$, is indeed a monoid, which is moreover isomorphic to $(\mathbb{N}, +, 0)$. Therefore, traditional FPS are isomorphic to *infinite sequences*—families of numbers indexed by $\mathbb{N}$.

2.1.3.1 The Semiring of Formal Power Series

We now state the formal definition of generalized FPS and their operations. We say that an element $v \in V$ of a monoid $(V, \odot, e)$ is *finitely decomposable*[18] if there exist at most finitely many pairs $(v_1, v_2) \in V \times V$ such that $v = v_1 \odot v_2$. For example, in $(\mathbb{N}, +, 0)$ and $(\Sigma^*, \cdot, \varepsilon)$, all elements are finitely decomposable. In $(\mathbb{Z}, +, 0)$, on the other hand, no element is finitely decomposable: The equation $v = v_1 + v_2$ has infinitely many solutions for every fixed $v \in \mathbb{Z}$.

18: This terminology is adopted from [117].

Definition 2.11 **(FPS: *Formal power series*)** Let $(W, +, \cdot, 0, 1)$ be a semiring and let $(V, \odot, e)$ be a monoid in which every element is finitely decomposable.

A. An FPS over V with coefficients in W is a function $f : V \to W$, which is often denoted by a *formal sum*

$$f \;=\; \sum_{v \in V} f(v)\, v \;.$$

The set of all such FPS is denoted by $W\langle\langle V \rangle\rangle$. In the special case $V = \mathsf{Mon}[z]$,[19] we use the standard notation $W[[z]]$ instead.

B. The *support* of an FPS $f : V \to W$ is defined as $\mathit{supp}(f) = \{w \in V \mid f(w) \neq 0\}$. A *polynomial* is an FPS whose support is finite.

C. The *sum* of two FPS $f, g \in W\langle\langle V \rangle\rangle$ is defined coefficientwise:

$$f + g \;=\; \sum_{v \in V} \big(f(v) + g(v)\big)\, v \;.$$

19: Recall that $\mathsf{Mon}[z]$ is the monoid of monomials in indeterminate z introduced on the current page.

D. The *product* of two FPS $f, g \in W\langle\langle V \rangle\rangle$ is defined as

$$f \cdot g \;=\; \sum_{v \in V} \left(\sum_{v_1 \odot v_2 = v} f(v_1) \cdot g(v_2) \right) v \,.$$

E. We write 0 and 1 (or e) for the FPS $\sum_{v \in V} 0v$ and $1e + \sum_{v \in V \setminus \{e\}} 0v$, the neutral elements relative to FPS addition and multiplication, respectively. △

Note that the inner sum in Definition 2.11D is an actual sum in the semiring, not a formal one. The sum is finite thanks to the assumption that all monoid elements are finitely decomposable.

Lemma 2.12 (The FPS semiring [87]) *Let $\mathcal{S} = (W, +, \cdot, 0, 1)$ be a semiring and let $(V, \odot, e)$ be a monoid in which every element is finitely decomposable. Then $\mathcal{S}\langle\langle V \rangle\rangle = (W\langle\langle V \rangle\rangle, +, \cdot, 0, 1)$ is again a semiring.*[20] *If $\mathcal{S}$ is ω-continuous,*[21] *then so is $\mathcal{S}\langle\langle V \rangle\rangle$.*

20: With $+$, $\cdot$, 0, and 1 as in Definition 2.11.

It is worth mentioning that every FPS semiring contains the set of polynomials as a sub-semiring; however, in general, ω-continuity of the coefficient semiring does not carry over to the semiring of polynomials. For example, in $\overline{\mathcal{R}}_{\geq 0}[[z]]$, the supremum of the polynomials $1 \sqsubseteq z \sqsubseteq z^2 \sqsubseteq \ldots$ is the infinite-support FPS $\sum_{n \in \mathbb{N}} z^n$.

21: See Definition 2.9 on page 19.

► **Example 2.13** (FPS)

– The infinite sum

$$\frac{1}{2} + \frac{1}{4}z + \frac{1}{8}z^2 + \frac{1}{16}z^3 + \ldots \;=\; \sum_{n \in \mathbb{N}} \frac{1}{2^{n+1}} z^n$$

can be viewed as an FPS in $\overline{\mathcal{R}}_{\geq 0}[[z]]$ which, as explained previously, is isomorphic to $\overline{\mathcal{R}}_{\geq 0}\langle\langle \mathbb{N} \rangle\rangle$. The above FPS is thus formally indistinguishable from the function $f : \mathbb{N} \to \overline{\mathbb{R}}_{\geq 0}$, $n \mapsto \frac{1}{2^{n+1}}$ which, in turn, can be viewed as the infinite sequence of negative powers of two. Consequently, the semiring operations $+$ and $\cdot$ of $\overline{\mathcal{R}}_{\geq 0}[[z]]$ correspond to elementwise addition and *discrete convolution* of sequences.

– The semiring $(\mathbb{B}\langle\langle \Sigma^* \rangle\rangle, \vee, \wedge, \mathsf{false}, \mathsf{true})$, where Σ is a finite alphabet, is isomorphic to the semiring $\mathcal{L}_\Sigma = (2^{\Sigma^*}, \cup, \cdot, \emptyset, \{\varepsilon\})$ of formal languages.[22]

22: See Example 2.8 on page 19.

See Examples 2.15, 2.16 and 2.19 for additional instances. ..... ◄

2.1.3.2 *Formal Derivatives and Substitution*

Next, we define some notions exclusive to FPS from $\mathcal{S}[[z]]$, where $\mathcal{S}$ is a semiring and z is a formal indeterminate. They play a key

role in Section 3.1.3 where they are applied in the context of our automata-theoretic take on the *method of generating functions* [128].

[128]: Wilf (2005), *generatingfunctionology*.

We start with *formal derivatives*. Let $\mathcal{S} = (W, +, \cdot, 0, 1)$ be a semiring. For $k \in \mathbb{N}$ and $w \in W$, define

$$kw \;=\; \underbrace{w + \ldots + w}_{k \text{ times}}$$

with the convention that $0w = 0$. This operation is associative and distributive.[23]

23: More precisely, for $k, \ell \in \mathbb{N}$ and $w, v \in W$:

- $(k\ell)w = k(\ell w)$
- $k(w \cdot v) = (kw)v$
- $(k + \ell)w = kw + \ell w$
- $k(w + v) = kw + kv$

Our formal derivative operator generalizes traditional derivatives of polynomials:

Definition 2.14 ***(Derivative)*** Let $\mathcal{S}$ be a semiring and let

$$f \;=\; \sum_{n \in \mathbb{N}} f_n \, z^n \;\in\; \mathcal{S}[[z]]$$

be an FPS. The *(formal) derivative* of f is defined as

$$\partial f \;=\; \sum_{n \in \mathbb{N}} (n+1) f_{n+1} \, z^n \;\in\; \mathcal{S}[[z]] \,. \qquad \triangle$$

In this thesis, we are mostly concerned with derivatives in the semiring $\overline{\mathcal{R}}_{\geq 0}[[z]]$.

► **Example 2.15** (Derivative of an FPS) The derivative of the FPS

$$f \;=\; \frac{1}{2} + \frac{1}{4}z + \frac{1}{8}z^2 + \frac{1}{16}z^3 + \ldots \;=\; \sum_{n \in \mathbb{N}} \frac{1}{2^{n+1}} z^n$$

from Example 2.13 is

$$\partial f \;=\; \frac{1}{4} + \frac{2}{8}z + \frac{3}{16}z^2 + \frac{4}{32}z^3 + \ldots \;=\; \sum_{n \in \mathbb{N}} \frac{n+1}{2^{n+2}} z^n \,.$$

Informally, to obtain the derivative, one considers each term of the FPS separately, *pulls down* the exponent in front of the coefficient, and decreases the exponent by one. ◄

► **Example 2.16** (Derivative of a more exotic FPS) While Example 2.15 will be familiar to many readers, we stress that we can also take derivatives of FPS with non-numeric coefficients. For instance, in $\overline{\mathcal{R}}_{\geq 0}[[z]]$, we have

$$\begin{aligned} &\partial\big(\infty + \infty z + \infty z^2 + \infty z^3 + \ldots\big) \\ &= 1\infty + 2\infty z + 3\infty z^2 + \ldots \\ &= \infty + \infty z + \infty z^2 + \ldots \,; \end{aligned}$$

the derivative operator thus has no effect on this particular FPS. ..... ◂

The following result shows that derivatives interact with the semiring operations in the familiar way:

Proposition 2.17 (Sum and product rule) *Let $\mathcal{S}$ be a semiring and let $f, g \in \mathcal{S}[[z]]$ be FPS. The derivative satisfies the following sum and product rules:*

A. $\partial(f + g) = \partial f + \partial g$
B. $\partial(f \cdot g) = \partial f \cdot g + f \cdot \partial g$

Proof sketch. Item A is an immediate consequence of the definitions. For Item B, we can determine expressions for the coefficients of the FPS on either side of the equation, and show that they coincide via basic arithmetic manipulations. □

The last operation we introduce is *substitution*.

Definition 2.18 ***(Substitution in FPS)*** Let $\mathcal{S}$ be an ω-continuous[24] semiring with carrier set W. Furthermore, let $f = \sum_{n\in\mathbb{N}} f_n z^n \in \mathcal{S}[[z]]$ and $w \in W$. The *substitution* of w for z in f is defined as

$$f\big|_{z=w} = \sum_{n\in\mathbb{N}} f_n w^n \in \mathcal{S} . \qquad \triangle$$

24: See Definition 2.9 on page 19.

Crucially, the sum on the right hand side of the equation in Definition 2.18 is an actual *infinite sum*[25] in the semiring $\mathcal{S}$.

25: See Definition 2.5 on page 18.

► **Example 2.19** (Substitution) Let $\mathcal{S} = (W, +, \cdot, 0, 1)$ be ω-continuous.

– Substituting 1 for z in an FPS $\sum_n f_n z^n \in \mathcal{S}[[z]]$ yields the infinite semiring-sum of its coefficients:

$$f\big|_{z=1} = \sum_n f_n .$$

– Similarly, taking the derivative[26] and then substituting 1 yields the following weighted sum of the coefficients:

$$\partial f\big|_{z=1} = \sum_n n f_n .$$

26: See Definition 2.14 on the preceding page.

– Consider the FPS $f = 1 + z + z^2 + \ldots = \sum_n z^n$. For all $w \in W$, we have

$$f\big|_{z=w} = w^* ,$$

where w^* is the star operator from Definition 2.10.

Substitution in FPS plays a crucial role in Section 3.1.3. ..... ◂

2.1.4 Matrices and Vectors

Traditionally, the two basic matrix operations, addition and multiplication, are defined entirely in terms of sums and products of the coefficients—neither subtraction nor division is needed. Many standard matrix computations can thus be performed on semirings. Moreover, it turns out that the set of *square* matrices (of a fixed dimension) with coefficients in a semiring forms again a semiring.[27]

27: See Lemma 2.21 on the next page.

Matrices In this thesis, we slightly deviate from traditional matrices such as, say, $A \in \mathbb{R}^{n \times m}$ in two ways: First, we allow more general index sets other than just $[1..n]$ and $[1..m]$. Second, we explicitly allow these index sets to be (countably) *infinite*. We stress that, in infinite matrices, the standard definition of matrix multiplication necessitates taking infinite sums. We formalize these conventions as follows:

Definition 2.20 ***(Matrices)*** Let $(W, +, \cdot, 0, 1)$ be a semiring.

A. Let I and J be countable index sets. The set of $I \times J$*-matrices* with *coefficients* in W is the set $W^{I \times J}$ of functions of type $I \times J \to W$. An $I \times J$-matrix is called *infinite* if at least one of I and J are infinite, and *square* if $I = J$.
B. *Addition* of matrices is defined coefficientwise: For two $I \times J$-matrices A and B, we set $(A + B)(i, j) = A(i, j) + B(i, j)$ for all $i \in I$ and $j \in J$.
C. *Multiplication* of matrices is defined as usual, that is, for a $I \times J$-matrix A and a $J \times K$-matrix B, we define the $I \times K$-matrix $A \cdot B$ as

$$(A \cdot B)(i, k) \;=\; \sum_{j \in J} A(i, j) \cdot B(j, k)$$

for all $i \in I$ and $k \in K$. If J is infinite, we require the semiring to be ω-continuous so that the resulting infinite sum is defined.[28]
D. The zero and identity matrices, whose dimensions will always follow from the context, are denoted by 0 and Id, respectively. △

28: See Definition 2.5 on page 18.

As usual, we often denote matrix multiplication by juxtaposition. Other common notions such as the *transpose* $A^\top$ are defined in the standard way and obey the familiar rules.[29] Notice that to display a matrix $A \in W^{I \times J}$ in the usual way as a rectangular array of coefficients, we need to fix an arbitrary linear order on the index sets. In most examples, an appropriate order will be clear from the context.[30]

29: In particular, the transpose satisfies $(A + B)^\top = A^\top + B^\top$ and $(AB)^\top = B^\top A^\top$.

30: See, for instance, Example 2.22 on the next page.

Lemma 2.21 (The matrix semiring [87]) *Let I be a countable index set and let $\mathcal{S} = (W, +, \cdot, 0, 1)$ be a semiring (required to be ω-continuous if I is infinite). Then $(W^{I\times I}, +, \cdot, 0, Id)$ with $+, \cdot, 0$, and Id as in Definition 2.20 is again a semiring, called the* semiring of matrices *over $\mathcal{S}$. If $\mathcal{S}$ is ω-continuous, then so is the semiring of matrices.*

► **Example 2.22** (Matrix arithmetic in the formal languages semiring) Let $2 = \{1, 2\}$ be an index set and consider the following 2×2-matrix A over the semiring $\mathcal{L}_\Sigma = (\Sigma^*, \cup, \cdot, \emptyset, \{\varepsilon\})$ of formal languages with alphabet $\Sigma = \{a, b\}$:

$$A = \begin{pmatrix} \{a,b\} & \{\varepsilon\} \\ \emptyset & \{a\}^* \end{pmatrix}$$

This representation means that the coefficients of A are $A(1,1) = \{a, b\}$, $A(1,2) = \{\varepsilon\}$, etc. Recall that $A(2,2) = \{a\}^* = \{\varepsilon, a, aa, aaa, \ldots\}$ is the (Kleene) star[31] of the language $\{a\}$.

31: See Definition 2.10 on page 20.

To illustrate matrix arithmetic, we showcase the computation of the square of A:

$$A^2 = \begin{pmatrix} \{a,b\} & \{\varepsilon\} \\ \emptyset & \{a\}^* \end{pmatrix} \cdot \begin{pmatrix} \{a,b\} & \{\varepsilon\} \\ \emptyset & \{a\}^* \end{pmatrix} = \begin{pmatrix} \{aa, ba, ab, bb\} & \{b\} \cup \{a\}^* \\ \emptyset & \{a\}^* \end{pmatrix}$$

Notice that, for example, $A^2(1,2)$ is determined as

$$\{a,b\} \cdot \{\varepsilon\} \cup \{\varepsilon\} \cdot \{a\}^* = \{a,b\} \cup \{a\}^* = \{b\} \cup \{a\}^* .$$

Importantly, by Lemma 2.21, the set of all such 2×2-matrices with coefficients in $\mathcal{L}_\Sigma$ forms again a semiring.[32] The neutral elements of this semiring are the zero matrix and the identity matrix:

32: Even an ω-continuous one since $\mathcal{L}_\Sigma$ is ω-continuous

$$0 = \begin{pmatrix} \emptyset & \emptyset \\ \emptyset & \emptyset \end{pmatrix} \qquad Id = \begin{pmatrix} \{\varepsilon\} & \emptyset \\ \emptyset & \{\varepsilon\} \end{pmatrix}$$

Furthermore, we stress that the usual rules of matrix manipulation, in particular distributivity and associativity, apply. ◄

Vectors Finally, a *vector* is either a $1 \times I$-matrix or a $I \times 1$-matrix, where 1 is a singleton set. More specifically, $1 \times I$-matrices are called *row* vectors, and $I \times 1$-matrices are called *column* vectors. The following convention will rest in place for the remainder of the whole text:

Convention 2.23 (Row and column vectors) We identify the set of *column* vectors $W^{I\times1}$ with W^I. Column vectors are usually denoted by boldface lowercase letters (e.g., $\mathbf{v}, \mathbf{u}$). *Row* vectors are usually written as explicitly transposed column vectors (e.g., $\mathbf{v}^\top, \mathbf{u}^\top$).

Notice that a column vector $\mathbf{v} \in W^I$ can be multiplied with a matrix $A \in W^{J \times I}$ *from the right*[33] to form the new column vector

$$A\mathbf{v} \in W^J .$$

Similarly, a row vector can be multiplied with a matrix of appropriate dimensions *from the left*, which yields again a row vector.

33: Matrix-vector multiplication is a special case of matrix-matrix multiplication.

Finally, we define *indicator vectors* as follows: Let 0 and 1 be the neutral elements of a semiring, and let I be an index set.[34] The indicator vector $\mathbf{e}_J$ of $J \subseteq I$ is a column vector defined as

$$\mathbf{e}_J(i) = \begin{cases} 1 & \text{if } i \in J \\ 0 & \text{otherwise} \end{cases}$$

for all $i \in I$. If $J = \{j\}$ is a singleton, we may write $\mathbf{e}_j$ instead of $\mathbf{e}_{\{j\}}$.

34: $0, 1$, and I will always be clear from the context whenever we use indicator vectors.

2.2 Weighted Automata

In this thesis, we identify automata with (finite or countably infinite) *directed weighted graphs*, where the weights are from some semiring.[35] The vertices of these graphs—the *states* of the automata—may additionally carry *initial* and *final weights*.

35: See Definition 2.7 on page 19.

Following the generative philosophy,[36] an automaton's semantics, also called *behavior*, is a certain aggregation of its *computation paths*, the finite paths in the underlying directed graph. Concretely, the behavior is the semiring-sum of the semiring-products of the weights on the transitions along each computation path. The paths are additionally multiplied with the initial weight of the first and the final weight of the last state.[37] It will be convenient to define additional *open* variants of the behavior requiring only initial *or* final weights, but not both.

36: See the introductory discussion of this chapter (page 13).

37: See Definition 2.28 for formal details.

2.2.1 Definition and Basic Examples

As there may exist *infinitely* many computation paths between any pair of states, we require the semiring of weights to be ω-continuous so that the resulting infinite sums over paths are well-defined. Overall, our notion of weighted automata is quite similar to that of Kuich [87]. The formal definition is as follows:

[87]: Kuich (1997), "Semirings and Formal Power Series: Their Relevance to Formal Languages and Automata".

Definition 2.24 ***(Weighted automaton)*** Let $\mathcal{S}$ be an ω-continuous semiring[38] with carrier set W.

A. An $\mathcal{S}$*-weighted automaton* $\mathfrak{A}$ is a tuple $\mathfrak{A} = (S, M)$, where
   - S is a countable set called *state space*, and
   - $M \in W^{S \times S}$ is a *transition matrix*.[39]

   $\mathfrak{A}$ is called *finite* if its state space S is a finite set.

B. An *initial* or *final weight vector* is any vector from the set W^S. △

38: See Definition 2.9 on page 19.

39: See Definition 2.20 on page 25.

Slightly deviating from the literature [87], our weighted automata are defined without fixed initial and final weight vectors. This is technically convenient because we will often consider the *same* automaton with *different* initial or final weights.

Weighted automata are sufficiently expressive to model a variety of standard graph-like structures, as the following examples illustrate:

► **Example 2.25** (Directed graphs) A *directed graph* (V, E) consists of a countable set V of vertices and a set $E \subseteq V \times V$ of edges. We can view a directed graph (V, E) as an $\mathcal{S}$-weighted automaton (V, M), where $\mathcal{S} = (W, +, \cdot, 0, 1)$ is an arbitrary ω-continuous semiring, and $M \in \{0, 1\}^{V \times V}$ is the adjacency matrix of the graph, that is, $M(v, u) = 1$ if and only if $(u, v) \in E$ for all $u, v \in V$. We will see in Example 2.29 on page 31 that, with an appropriate choice of $\mathcal{S}$ and of the initial and final weights, various properties of the graph can be expressed as a behavior of the resulting automaton. ◄

► **Example 2.26** (Standard NFA) A traditional *nondeterministic finite automaton* (NFA) is a 5-tuple $(S, \Sigma, \delta, s_0, F)$ where S is a finite set of states, Σ is a finite input alphabet, $\delta \subseteq S \times \Sigma \times S$ is a transition relation, $s_0 \in S$ is an initial state, and $F \subseteq S$ is a set of accepting states. An NFA can be viewed as a weighted automaton (S, M) over the formal power series semiring[40] $\mathcal{B}\langle\langle \Sigma^* \rangle\rangle$ where M is defined as

$$\forall s, t \in S\colon \quad M(s, t) \;=\; \sum_{(s,a,t) \in \delta} a \,.$$

Alternatively, since $\mathcal{B}\langle\langle \Sigma^* \rangle\rangle$ is isomorphic to $\mathcal{L}_\Sigma$, the semiring of formal languages,[41] we may also view an NFA as an $\mathcal{L}_\Sigma$-weighted automaton where

$$\forall s, t \in S\colon \quad M(s, t) \;=\; \bigcup_{(s,a,t) \in \delta} \{a\} \,.$$

Finally, to model the NFA's initial and accepting states, the characteristic vectors[42] of the sets $\{s_0\}$ and F can be employed as initial and final weight vectors, respectively. ◄

40: See Definition 2.11 on page 21.

41: See Example 2.8 on page 19.

42: See Section 2.1.4 on page 25.

► **Example 2.27** (Markov chains) Lastly, we briefly mention that a *(discrete-time) Markov chain* (DTMC) is a directed graph whose edges are weighted with probabilities such that the sum of the weights on the outgoing edges of every vertex equals one. We can thus view a DTMC as an $\overline{\mathcal{R}}_{\geq 0}$-weighted automaton (S, M) with the additional restriction that

$$\forall s \in S\colon \quad \sum_{t \in S} M(s,t) \;=\; 1\,.$$

Moreover, we can encode the DTMC's initial distribution—if given—as the automaton's initial weight vector. DTMCs are of central importance in this thesis; we shall study them in greater detail in Section 2.3. .....◄

2.2.2 *Semantics of Weighted Automata*

We now define the semantics (subsequently called *behavior*) of weighted automata. Let $\mathfrak{A} = (S, M)$ be a weighted automaton with initial and final weight vectors $\mathbf{g}$ and $\mathbf{f}$ over some semiring $\mathcal{S} = (W, +, \cdot, 0, 1)$. A *computation path* of length $k \in \mathbb{N}$ is a finite sequence $(s_0, \ldots, s_k)$ of states from S. Note that this includes paths of length zero consisting of a single state; this state is considered both the start and the end point of the path.

Recall from the introduction of this chapter that determining the behavior of $\mathfrak{A}$ amounts to summing up the weights of all computations paths. To this end, we may consider the paths by increasing length k. First, let $k = 0$. For every path (s_0) of length zero, we determine its weight as $\mathbf{g}(s_0) \cdot \mathbf{f}(s_0)$, i.e., the initial weight is multiplied with the final weight. The sum of the weights of *all* zero-length paths can thus be written as the dot product

$$\sum_{s_0 \in S} \mathbf{g}(s_0) \cdot \mathbf{f}(s_0) \;=\; \mathbf{g}^\top \cdot \mathbf{f}\,.$$

Next, let $k = 1$, i.e., we consider paths of the form (s_0, s_1) with exactly one transition. The weight of such a path is:

$$\underbrace{\mathbf{g}(s_0)}_{\text{initial weight of } s_0} \cdot \underbrace{M(s_0, s_1)}_{\text{weight of transition } s_0 \to s_1} \cdot \underbrace{\mathbf{f}(s_1)}_{\text{final weight of } s_1}$$

Summing the weights of all paths of length one yields

$$\sum_{s_0, s_1 \in S} \mathbf{g}(s_0) \cdot M(s_0, s_1) \cdot \mathbf{f}(s_1) \;=\; \mathbf{g}^\top \cdot M \cdot \mathbf{f}\,.$$

Finally, let us consider paths of length $k = 2$. The weight of a path (s_0, s_1, s_2) is $\mathbf{g}(s_0) \cdot M(s_0, s_1) \cdot M(s_1, s_2) \cdot \mathbf{f}(s_2)$, and the sum of the

weights of all such paths can be written as

$$\sum_{s_0,s_1,s_2 \in S} \mathbf{g}(s_0) \cdot M(s_0,s_1) \cdot M(s_1,s_2) \cdot \mathbf{f}(s_2) \;=\; \mathbf{g}^\top \cdot M^2 \cdot \mathbf{f}\,.$$

Following this pattern, it is natural to define the behavior of $\mathfrak{A}$ as the infinite sum

$$\mathbf{g}^\top \cdot \mathbf{f} \;+\; \mathbf{g}^\top \cdot M \cdot \mathbf{f} \;+\; \mathbf{g}^\top \cdot M^2 \cdot \mathbf{f} \;+\; \ldots \;=\; \sum_{k \geq 0} \mathbf{g}^\top \cdot M^k \cdot \mathbf{f}\,.$$

Using distributivity of matrix-vector multiplication, the behavior can be written more concisely in terms of the Kleene star[43] of M as in the following definition:

43: See Definition 2.10 on page 20.

Definition 2.28 ***(Behavior)*** Let $\mathfrak{A} = (S, M)$ be a weighted automaton over $(W, +, \cdot, 0, 1)$.

A. For given vectors $\mathbf{g}, \mathbf{f} \in W^S$ of initial and final weights, respectively, the semiring element

$$[\![\mathbf{g}\,, \mathfrak{A}\,, \mathbf{f}]\!] \;=\; \mathbf{g}^\top \cdot M^* \cdot \mathbf{f}$$

is called the *behavior* of $\mathfrak{A}$ with regards to $\mathbf{g}$ and $\mathbf{f}$ [87]. [87]: Kuich (1997).

B. For a given final weight vector $\mathbf{f} \in W^S$, the vector

$$\langle\!\langle \mathfrak{A}\,, \mathbf{f}]\!] \;=\; M^* \cdot \mathbf{f}$$

is called the *backward behavior* of $\mathfrak{A}$ with respect to $\mathbf{f}$.

C. For a given initial weight vector $\mathbf{g} \in W^S$, we call the vector

$$[\![\mathbf{g}\,, \mathfrak{A}\rangle\!\rangle \;=\; \left(\mathbf{g}^\top \cdot M^*\right)^\top \;=\; (M^*)^\top \cdot \mathbf{g}$$

the *forward behavior* of $\mathfrak{A}$ relative to $\mathbf{g}$.

Whenever the initial and final weight vectors are clear from the context, we may simply speak of *the* behavior of $\mathfrak{A}$. △

To our knowledge, the above notions of forward and backward behavior have not appeared explicitly in previous literature. Of course, the difference to the standard behavior is not substantial. Indeed, the definitions immediately imply the following straightforward relationship (let $\mathbf{g}$, $\mathfrak{A}$, $\mathbf{f}$ be as in Definition 2.28):

$$[\![\mathbf{g}\,, \mathfrak{A}\rangle\!\rangle^\top \cdot \mathbf{f} \;=\; [\![\mathbf{g}\,, \mathfrak{A}\,, \mathbf{f}]\!] \;=\; \mathbf{g}^\top \cdot \langle\!\langle \mathfrak{A}\,, \mathbf{f}]\!] \,. \tag{2.2}$$

We also remark that forward and backward behaviors are (somewhat) interchangeable in the following sense: The forward behavior $[\![\mathbf{g}\,, \mathfrak{A}\rangle\!\rangle$ coincides with the backward behavior $\langle\!\langle \mathfrak{A}^\top\,, \mathbf{g}]\!]$, where $\mathfrak{A}^\top$

arises from $\mathfrak{A}$ by transposing the transition matrix. Similarly, it holds that $\langle\!\langle \mathfrak{A}, \mathbf{f}]\!] = [\![\mathbf{f}, \mathfrak{A}^\top \rangle\!\rangle$.

Despite these close relationships, considering forward and backward behaviors as separate notions is useful:

- They can both be characterized as a solution of a linear equation system[44]. In fact, a common approach to determine the (standard) behavior is to solve said equation system and apply Equation (2.2).
- In situations where only either initial or final weights are available, forward and backward behaviors often yield useful results. For instance, when viewing Markov chains[45] as weighted automata, initial weights are usually available in the form of the initial distribution, but it is not immediately clear what the final weights should be.[46]

44: See Proposition 2.31 on the next page.

45: See Example 2.27 on page 29.

46: Useful choices for final weights in Markov chains are explored in Section 2.3.

In the following examples, we interpret the forward, backward, and (standard) behavior of various kinds of weighted automata.

► **Example 2.29** (Properties of directed graphs as behaviors) Let $\mathfrak{A} = (S, M)$ be the $\mathcal{S}$-weighted automaton encoding a directed graph (S, E) as in Example 2.25. Moreover, let $G, F \subseteq S$ be two distinguished sets of vertices with characteristic vectors $\mathbf{g}$ and $\mathbf{f}$, respectively.

- Let $\mathcal{S} = \mathcal{B}$ be the Boolean semiring. Then $[\![\mathbf{g}, \mathfrak{A}, \mathbf{f}]\!] = \mathsf{true}$ iff some vertex in F is reachable from some vertex in G. Moreover, $\langle\!\langle \mathfrak{A}, \mathbf{f}]\!]$ is the characteristic vector of the set of vertices that can reach F, whereas $[\![\mathbf{g}, \mathfrak{A} \rangle\!\rangle$ is the characteristic vector of the vertices reachable from some vertex in G.
- Let $\mathcal{S} = \overline{\mathcal{R}}_{\geq 0}$ be the semiring of extended nonnegative real numbers. In this case, $[\![\mathbf{g}, \mathfrak{A}, \mathbf{f}]\!]$ is the *number of paths* between G and F, which is possibly ∞.
- Let $\mathcal{S} = (\overline{\mathbb{N}}, \min, +, \infty, 0)$ be the *tropical semiring*.[47] Then $[\![\mathbf{g}, \mathfrak{A}, \mathbf{f}]\!]$ is the length of a *shortest path* from G to F, or ∞ if no such path exists. .... ◄

47: The tropical semiring is only mentioned here for ilustration; it does not play a role in the further developments of this thesis.

► **Example 2.30** (Behavior of an NFA) Let $\mathfrak{A}$ be an NFA over alphabet Σ, i.e., a $\mathcal{B}\langle\!\langle \Sigma^* \rangle\!\rangle$-weighted automaton as in Example 2.26, with vectors $\mathbf{g}$ and $\mathbf{f}$ encoding the initial and accepting states, respectively. Then $[\![\mathbf{g}, \mathfrak{A}, \mathbf{f}]\!]$ is the set of words[48] accepted by $\mathfrak{A}$. Similarly, $\langle\!\langle \mathfrak{A}, \mathbf{f}]\!]$ associates each state with the language accepted by $\mathfrak{A}$ if the automaton starts in that state; the interpretation of $[\![\mathbf{g}, \mathfrak{A} \rangle\!\rangle$ is dual.

48: Recall that $\mathcal{B}\langle\!\langle \Sigma^* \rangle\!\rangle$ is isomorphic to $\mathcal{L}_\Sigma$, the semiring of formal languages.

We can also switch the base semiring from $\mathcal{B}$ to, say, $\overline{\mathcal{R}}_{\geq 0}$ and view $\mathfrak{A}$ as a $\overline{\mathcal{R}}_{\geq 0}\langle\!\langle \Sigma^* \rangle\!\rangle$-weighted automaton. The behavior $[\![\mathbf{g}, \mathfrak{A}, \mathbf{f}]\!]$ is then a formal power series associating each word $w \in \Sigma^*$ with the number of accepting runs of $\mathfrak{A}$ on w. ..... ◄

The following result characterizes forward and backward behaviors as least solutions of linear equation systems.[49] The first item is Theorem 4.1 in [87]; the second item is an easy consequence of the

49: Possibly over infinitely many variables

[87]: Kuich (1997).

first.

Proposition 2.31 (Linear equation systems for behaviors [87]) *Let $\mathfrak{A} = (S, M)$ be a weighted automaton over an ω-continuous semiring $(W, +, \cdot, 0, 1)$.*

A. *Let $\mathbf{f} \in W^S$ be a final weight vector. The backward behavior $\langle\!\langle \mathfrak{A}, \mathbf{f}]\!]$ is the componentwise least solution[50] $\boldsymbol{x} \in W^S$ of the linear equation system*

$$\boldsymbol{x} \;=\; M\boldsymbol{x} + \mathbf{f}\,.$$

B. *Similarly, letting $\mathbf{g} \in W^S$ be an initial weight vector, the forward behavior $[\![\mathbf{g}, \mathfrak{A} \rangle\!\rangle$ equals the componentwise least solution $\boldsymbol{x} \in W^S$ of*

$$\boldsymbol{x}^\top \;=\; \mathbf{g}^\top + \boldsymbol{x}^\top M\,.$$

50: In particular, the set of solutions has a unique (componentwise) minimum.

Explicit applications of Proposition 2.31 can be found in Section 2.3 (e. g., Example 2.42 on page 37).

2.2.3 Behavioral Moments

We now introduce a generalization of the behavior[51] of a weighted automaton. The idea is to increase the weight of the *longer* paths. Specifically, for some $n \in \mathbb{N}$, our proposed modification of the behavior—we call it nth *behavioral moment*—counts the paths of length k with multiplicity k^n.[52]

51: See Definition 2.28 on page 30.

52: This modification only has an effect in semirings that are *not* idempotent, i. e., where $w + w$ may differ from w. For example, $\mathcal{B}$ is idempotent, but $\overline{\mathcal{R}}_{\geq 0}$ is not.

The following definition assumes that $0^0 = 1$.

Definition 2.32 ***(Behavioral moments)*** Let $\mathfrak{A} = (S, M)$ be a weighted automaton over $(W, +, \cdot, 0, 1)$. Furthermore, let $n \in \mathbb{N}$.

53: Notice that $M^{\langle 0 \rangle} = M^*$.

A. We define the following generalization[53] of the Kleene star:

$$M^{\langle n \rangle} \;=\; \sum_{k \geq 0} k^n M^k\,.$$

54: This is completely analogous to Definition 2.28; we merely replaced M^* by $M^{\langle n \rangle}$.

B. The nth *(forward, backward) behavioral moment* relative to initial and final weight vectors $\mathbf{g}, \mathbf{f} \in W^S$ is defined as:[54]

$$\begin{aligned} [\![\mathbf{g}, \mathfrak{A}, \mathbf{f}]\!]^{\langle n \rangle} &= \mathbf{g}^\top \cdot M^{\langle n \rangle} \cdot \mathbf{f} \\ [\![\mathbf{g}, \mathfrak{A} \rangle\!\rangle^{\langle n \rangle} &= \mathbf{g}^\top \cdot M^{\langle n \rangle} \\ \langle\!\langle \mathfrak{A}, \mathbf{f}]\!]^{\langle n \rangle} &= M^{\langle n \rangle} \cdot \mathbf{f} \end{aligned}$$

△

The definition of the behavioral moments is reminiscent of the moments of a random variable with values in $\mathbb{N}$.[55] In fact, we will see in Section 2.3.3 that certain expectations can be expressed as behavioral moments. However, behavioral moments are not limited to probabilistic applications.

55: That is, the expected values $E[X^n]$ where $n \in \mathbb{N}$ and X is the random variable in question.

► **Example 2.33** (Behavioral moment of a directed graph) Let $\mathfrak{A} = (S, M)$ be the $\overline{\mathcal{R}}_{\geq 0}$-weighted automaton encoding a directed graph (S, E) equipped with two distinguished sets of vertices $G, F \subseteq S$ with characteristic vectors $\mathbf{g}$ and $\mathbf{f}$, respectively.[56] The first behavioral moment $[\![\mathbf{g}, \mathfrak{A}, \mathbf{f}]\!]^{\langle 1 \rangle}$ is the *sum of the lengths of all paths from G to F*. If this number is neither 0 nor ∞, then the same holds for $[\![\mathbf{g}, \mathfrak{A}, \mathbf{f}]\!]$, the number of paths from G to F, and we can determine the *average path length* as $[\![\mathbf{g}, \mathfrak{A}, \mathbf{f}]\!]^{\langle 1 \rangle} / [\![\mathbf{g}, \mathfrak{A}, \mathbf{f}]\!]$. ◄

56: See Example 2.29 on page 31.

We close this section with two lemmas that will be useful later on. The first is an immediate consequence of the definitions:

Lemma 2.34 *Let $\mathfrak{A} = (S, M)$ be a weighted automaton with initial and final weight vector $\mathbf{g}$ and $\mathbf{f}$, respectively. For all $n \in \mathbb{N}$,*

$$[\![\mathbf{g}, \mathfrak{A}, \mathbf{f}]\!] = 0 \quad \iff \quad [\![\mathbf{g}, \mathfrak{A}, \mathbf{f}]\!]^{\langle n \rangle} = 0 \,.$$

This holds analogously for forward and backward behaviors.

From now on, we focus on the *first* behavioral moments. The corresponding generalized Kleene star operator can be expressed in terms of the standard Kleene star as follows:

Lemma 2.35 *For every square matrix M over an ω-continuous semiring,*

$$M^{\langle 1 \rangle} \;=\; M^* \cdot M \cdot M^* \,.$$

Proof. The statement can be proved as follows:

$$\begin{aligned}
& M^{\langle 1 \rangle} \\
&= M + (M^2 + M^2) + (M^3 + M^3 + M^3) + \ldots \\
& \qquad\qquad \text{(Definition of } M^{\langle 1 \rangle}\text{)} \\
&= (M + M^2 + M^3 + \ldots) + (M^2 + M^3 + \ldots) + (M^3 + \ldots) + \ldots \\
& \qquad \text{(Commutativity of infinite sums, see Lemma 2.6)} \\
&= \sum_{k \geq 1} \sum_{\ell \geq k} M^\ell \qquad \text{(Previous line in sum notation)} \\
&= \sum_{k \geq 1} \left(M^k \cdot \sum_{\ell \geq 0} M^\ell \right) \qquad \text{(Distributivity)} \\
&= \sum_{k \geq 1} \left(M^k \cdot M^* \right) \qquad \text{(Definition of } M^*\text{)}
\end{aligned}$$

$$= \sum_{k\geq 0} \left(M^k \cdot M \cdot M^* \right) \qquad \text{(Index shift)}$$
$$= M^* \cdot M \cdot M^* \qquad \text{(Distributivity and definition of } M^*\text{)}$$

This completes the proof of Lemma 2.35. □

▷ **Remark 2.36** (From behavioral moments to behaviors) A consequence of Lemma 2.35 is that the first behavioral moment of a weighted automaton $\mathfrak{A} = (S, M)$ with initial and final vectors $\mathbf{g}$ and $\mathbf{f}$ can be expressed as the (standard) behavior of a modified automaton $\mathfrak{A}' = (S', M')$, where S' consists of two disjoint copies of S, and:

$$\mathbf{g'}^\top = \begin{pmatrix} \mathbf{g}^\top & \mathbf{0} \end{pmatrix} \qquad M' = \begin{pmatrix} M & M \\ 0 & M \end{pmatrix} \qquad \mathbf{f}' = \begin{pmatrix} \mathbf{0} \\ \mathbf{f} \end{pmatrix}$$

This particular shape of M' implies that there are transitions from the first to the second copy, but not the other way around. We have [87, Corollary 2.6]

[87]: Kuich (1997).

$$M^* = \begin{pmatrix} M^* & M^* \cdot M \cdot M^* \\ 0 & M^* \end{pmatrix}$$

and therefore

$$[\![\mathbf{g}', \mathfrak{A}', \mathbf{f}']\!] = [\![\mathbf{g}, \mathfrak{A}, \mathbf{f}]\!]^{\langle 1 \rangle}.$$

This construction can also be generalized to accommodate higher moments. While we do not rely explicitly on this technique in the remainder, we have opted for including it here as it appears to be of independent interest. We are aware that a similar observation has been used in the context of *parametric Markov chains* in [66], but beyond that, the fact does not seem well-known in the literature. ◁

[66]: Heck et al. (2022), "Gradient-Descent for Randomized Controllers Under Partial Observability".

▷ **Remark 2.37** (Moment semirings) An alternative approach for characterizing the behavioral moments is via *moment semirings* [127]. Let $\mathcal{S} = (W, +, \cdot, 0, 1)$ be a semiring. The *first order moment semiring* of $\mathcal{S}$ is defined as

[127]: Wang et al. (2021), "Central moment analysis for cost accumulators in probabilistic programs".

$$\mathcal{S}^{\langle 1 \rangle} = (W \times W, +, \cdot, (0,0), (1,0)),$$

where

- $(w_1, v_1) + (w_2, v_2) = (w_1 + w_2, v_1 + v_2)$, and
- $(w_1, v_1) \cdot (w_2, v_2) = (w_1 \cdot w_2, w_1 \cdot v_2 + v_1 \cdot w_2)$.

It can be shown that $\mathcal{S}^{\langle 1 \rangle}$ inherits ω-continuity from $\mathcal{S}$. We now explain how to express the first behavioral moment of an $\mathcal{S}$-weighted automaton $\mathfrak{A}$ as the standard behavior of a suitably defined $\mathcal{S}^{\langle 1 \rangle}$-weighted automaton. The crucial observation is that, for all $n \in \mathbb{N}$ and $w_1, \ldots, w_n \in W$,

$$\prod_{i=1}^{n} (w_i, w_i) = \left(\prod_{i=1}^{n} w_i \,,\; n \cdot \prod_{i=1}^{n} w_i \right),$$

which can be proved by induction on n. This observation motivates constructing $\mathfrak{A}'$ by transforming every weight w of $\mathfrak{A}$ into (w, w). Assume further that $\mathfrak{A}$ has initial and final weights $\mathbf{g}$ and $\mathbf{f}$, and define $\mathbf{g}'(s) = (\mathbf{g}(s), 0)$ and $\mathbf{f}'(s) = (\mathbf{f}(s), 0)$ for every $s \in S$. Then we can show that

$$[\![\mathbf{g}', \mathfrak{A}', \mathbf{f}']\!] = \left([\![\mathbf{g}, \mathfrak{A}, \mathbf{f}]\!], [\![\mathbf{g}, \mathfrak{A}, \mathbf{f}]\!]^{\langle 1 \rangle}\right) .$$

In words, the first behavioral moment of $\mathfrak{A}$ can be recovered as the second component of the behavior of $\mathfrak{A}'$.

Without going into details, we remark that it is possible to generalize the construction of $\mathcal{S}^{\langle 1 \rangle}$ to the nth order moment semiring $\mathcal{S}^{\langle n \rangle}$, where $n \in \mathbb{N}$ [127]. The construction of $\mathfrak{A}'$ from above can also be generalized accordingly. This allows expressing behavioral moments of arbitrary order $n \in \mathbb{N}$ as the standard behavior of an $\mathcal{S}^{\langle n \rangle}$-weighted automaton. ..... ◁

2.3 Markov Chains as Weighted Automata

In this section, we explain how DTMCs (discrete-time Markov chains) on countable states spaces can be viewed as weighted automata. Furthermore, we characterize various important quantities associated with a DTMC in terms of suitable behaviors or behavioral moments. We will work with the following definition:[57]

57: Cf. Example 2.27 on page 29.

Definition 2.38 ***(DTMC: Discrete-time Markov chain)*** A DTMC is an $\overline{\mathcal{R}}_{\geq 0}$-weighted automaton[58] $\mathfrak{M} = (S, M)$ whose transition matrix M is *substochastic*, that is,

58: See Definition 2.24 on page 28.

$$\forall s \in S: \quad \sum_{t \in S} M(s, t) \leq 1 .$$

$\mathfrak{M}$ is called *proper* if M is a proper stochastic matrix, that is, if $\sum_{t \in S} M(s, t) = 1$ for every $s \in S$; otherwise, $\mathfrak{M}$ is called *improper*. △

In the literature, the term *DTMC* usually refers to *proper* DTMCs with a fixed initial distribution (e. g., [8, Chapter 10]). In the weighted automata framework, it is natural to model such an initial distribution as an appropriate initial weight vector $\mathbf{g}$ with $\sum_{s \in S} \mathbf{g}(s) = 1$. Moreover, we mention that many authors define DTMCs over more general—possibly uncountable—state spaces; in this thesis, however, we only consider DTMCs with at most countable state space.

[8]: Baier et al. (2008), *Principles of Model Checking*.

The primary reason for allowing improper DTMCs with a substochastic transitions matrix is technical convenience. Moreover, improper DTMC will prove natural for our automata-theoretic approach: The *missing* probability mass at each state turns out to be

an interesting choice for a final weight vector; see Section 2.3.1 on page 36.

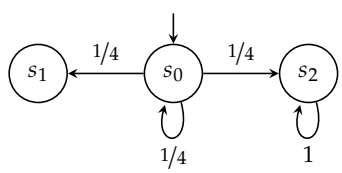
The vocabulary used throughout this section (e. g., *termination probabilities* in Section 2.3.1, *expected runtimes* in Section 2.3.2) is inspired by probabilistic program analysis. We remark that the corresponding concepts in the mathematical literature on Markov chains are usually called differently.

2.3.1 Termination Probabilities

Recall that we allow DTMCs $\mathfrak{M} = (S, M)$ with a *sub*stochastic transition matrix; that is, $M\mathbf{1} < \mathbf{1}$ is possible. Our interpretation of states $s \in S$ with $(M\mathbf{1})(s) < 1$ is that, upon entering s, the chain *terminates* with the missing probability mass $1 - (M\mathbf{1})(s)$. Intuitively, the *termination probability* is the probability that the DTMC terminates eventually (assuming some initial distribution). DTMCs with a proper stochastic transition matrix have a termination probability of zero. Termination probabilities are closely related to *reachability probabilities*.[59]

59: See Remark 2.43 on page 38.

► **Example 2.39** (Termination probability of an improper DTMC) Consider the following DTMC that starts in s_0 with probability 1:



The DTMC is improper: s_1 has no outgoing transitions and s_0 lacks an outgoing probability of 1/4. Intuitively, the DTMC can behave in three different ways:

- s_1 is reached eventually and the chain immediately terminates there.
- s_2 is reached eventually and the chain cycles there forever.
- The chain terminates in s_0 by picking the missing probability 1/4 *before* transitioning to either s_1 or s_2.

These three events are equally likely—each occurs with probability 1/3. The (overall) termination probability of the DTMC is thus 2/3. ◄

We now state the formal definition of termination probabilities and the associated notion of *almost sure termination* (AST), a term borrowed from probabilistic program analysis [61, 116].

[61]: Hart et al. (1982), "Termination of Probabilistic Concurrent Programs".

[116]: Saheb-Djahromi (1978), "Probabilistic LCF".

Definition 2.40 ***(Termination probabilities and AST)*** Let $\mathfrak{M} = (S, M)$ be a DTMC.

A. The vector of *immediate termination probabilities*[60] of $\mathfrak{M}$ is

$$\mathbf{t} \;=\; \mathbf{1} - M\mathbf{1} \;\in\; [0,1]^S \,.$$

B. The *termination probability* of $\mathfrak{M}$ with regards to an initial distribution $\mathbf{g}$ on S is defined as the behavior[61] $[\![\mathbf{g}\,, \mathfrak{M}\,, \mathbf{t}]\!]$.

C. $\mathfrak{M}$ is called *almost surely terminating* (AST) relative to initial distribution $\mathbf{g}$ (or $\mathbf{g}$-AST, for short) if

$$[\![\mathbf{g}\,, \mathfrak{M}\,, \mathbf{t}]\!] \;=\; 1\,,$$

that is, the termination probability of $\mathfrak{M}$ starting in $\mathbf{g}$ is one.

D. $\mathfrak{M}$ is called *universally AST* if $\mathfrak{M}$ is AST relative to *all* initial distributions on S. △

60: Intuitively, this is the missing probability mass on the outgoing transitions of every state. A proper DTMC satisfies $\mathbf{t} = \mathbf{0}$.

61: See Definition 2.28 on page 30.

Let us collect some straightforward consequences of Definition 2.40 in the following lemma:

Lemma 2.41 *Let $\mathfrak{M} = (S, M)$ be a DTMC with immediate termination probabilities $\mathbf{t} = \mathbf{1} - M\mathbf{1}$.*

A. *For all initial distributions $\mathbf{g}$ on S, we have $[\![\mathbf{g}\,, \mathfrak{M}\,, \mathbf{t}]\!] \leq 1$.*[62]

B. *It holds that $\langle\!\langle \mathfrak{M}\,, \mathbf{t}]\!] \leq \mathbf{1}$*[63] *and*

$$\mathfrak{M} \textit{ is universally AST} \quad\Longleftrightarrow\quad \langle\!\langle \mathfrak{M}\,, \mathbf{t}]\!] \;=\; \mathbf{1}\,.$$

62: This confirms that termination probabilities are in fact probabilities (numbers in $[0,1]$), which may not be immediately obvious since we defined termination probabilities ad hoc as behaviors of weighted automata.

63: Recall that $\langle\!\langle \mathfrak{M}\,, \mathbf{t}]\!]$ is a *backward behavior;* see Definition 2.28 on page 30.

Proof. For the first item, it suffices to notice that

$$M\mathbf{1} + \mathbf{t} \;=\; M\mathbf{1} + \mathbf{1} - M\mathbf{1} \;=\; \mathbf{1}\,,$$

which means that $\mathbf{1}$ upper-bounds the least solution of the equation system from Proposition 2.31 on page 32. The second item is immediate. □

► **Example 2.42** (Computing the termination probability) Reconsider the DTMC $\mathfrak{M} = (\{s_0, s_1, s_2\}, M)$ from Example 2.39 (restated in the margin), where

$$M \;=\; \begin{pmatrix} 1/4 & 1/4 & 1/4 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

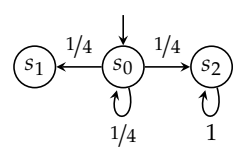


Figure 2.2: DTMC from Example 2.39 restated.

The vector of immediate termination probabilities[64] of $\mathfrak{M}$ is

$$\mathbf{t} \;=\; \mathbf{1} - M\mathbf{1} \;=\; \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} - \begin{pmatrix} 3/4 \\ 0 \\ 1 \end{pmatrix} \;=\; \begin{pmatrix} 1/4 \\ 1 \\ 0 \end{pmatrix}.$$

64: See Definition 2.40A.

Let us compute the backward behavior $\langle\!\langle \mathfrak{M}, \mathbf{t}]\!]$. By Proposition 2.31, $\langle\!\langle \mathfrak{M}, \mathbf{t}]\!]$ is the componentwise least nonnegative solution $\boldsymbol{x}$ of the linear equation system $\boldsymbol{x} = M\boldsymbol{x} + \mathbf{t}$. A standard computation yields

$$\boldsymbol{x} = \langle\!\langle \mathfrak{M}, \mathbf{t}]\!] = \begin{pmatrix} 2/3 \\ 1 \\ 0 \end{pmatrix}.$$

Thus, that the termination probability is $2/3$ if $\mathfrak{M}$ starts in s_0, 1 if $\mathfrak{M}$ starts in s_1, and 0 if $\mathfrak{M}$ starts in s_2. Moreover, following the terminology of Definition 2.40, $\mathfrak{M}$ is *not* universally AST, but $\mathbf{g}$-AST for the initial distribution $\mathbf{g}$ with $\mathbf{g}(s_1) = 1$. ◀

▷ **Remark 2.43** (Termination *vs* reachability probabilities) The termination probability of an improper DTMC can equivalently be represented as a *reachability probability*[65] in a *proper* DTMC. The latter is constructed by adding a new target state $s_\perp$ and redirecting the missing probability mass at each state to $s_\perp$. Conversely, the reachability probability towards a given set of target states in a proper DTMC matches the termination probability of the corresponding improper DTMC obtained by removing all outgoing transitions from the target states. Although these two notions are equivalent, we base our presentation on termination probabilities which seem more natural in the automata-theoretic setting. ◁

65: See, e. g., [8, Chapter 10].

▷ **Remark 2.44** (Weakest pre-expectations) *Weakest pre-expectations* are a well-established concept in probabilistic program verification [74, 84, 97]. In DTMCs, we can interpret weakest pre-expectations as follows [12]: Consider a—possibly improper—DTMC $\mathfrak{M} = (S, M)$ equipped with a *reward vector* $\mathbf{f} \colon S \to \overline{\mathbb{R}}_{\geq 0}$, also called *postexpectation*. The meaning of $\mathbf{f}$ is that upon termination in state $s \in S$, the chain collects the reward $\mathbf{f}(s)$. No reward is collected prior to termination. The weakest pre-expectation of $\mathfrak{M}$ with respect to $\mathbf{f}$ is then a function assigning to each $s \in S$ the *expected* reward given that the chain starts in s. It can be shown that this corresponds to the backward behavior

$$\langle\!\langle \mathfrak{M}, \mathbf{t} \cdot \mathbf{f}]\!],$$

where $\mathbf{t}$ denotes the immediate termination probabilities of $\mathfrak{M}$, and the multiplication is defined componentwise. Notice that the termination probabilities are the weakest pre-expectation with respect to $\mathbf{f} = \mathbf{1}$. ◁

[74]: Kaminski (2019), "Advanced weakest precondition calculi for probabilistic programs".

[84]: Kozen (1985), "A Probabilistic PDL".

[97]: McIver et al. (2005), *Abstraction, Refinement and Proof for Probabilistic Systems*.

[12]: Batz (2024), "Automated deductive verification of probabilistic programs".

2.3.2 *Expected Runtimes*

The next quantity we introduce is called *expected runtime*, yet another notion borrowed from probabilistic program analysis [76]. Intuitively, the expected runtime of a DTMC is the expected number of transitions taken until termination. Just like termination probabilities, expected runtimes depend on the initial distribution in general. If a DTMC is not AST[66] for a given initial distribution, then the

[76]: Kaminski et al. (2016), "Weakest Precondition Reasoning for Expected Run-Times of Probabilistic Programs".

66: See Definition 2.40 on the previous page.

expected runtime is ∞. Therefore, finite expected runtime, which has been dubbed *positive AST* (PAST) [19], is a *sufficient* condition for AST:

[19]: Bournez et al. (2005), "Proving Positive Almost-Sure Termination".

$$\text{PAST} \quad \Longrightarrow \quad \text{AST}$$

In general, PAST is stronger than AST: The reverse of the above implication does not hold. These facts are well-known; we reprove them formally in our framework in Theorem 2.54 on page 45.

In probability theoretic terms, a DTMC's runtime is a *random variable* X taking on values in $\mathbb{N} \cup \{\infty\}$. The expected value of X can be written as[67]

67: Intuitively, this infinite sum counts the probability $Pr[X = k]$ exactly k times, even if $k = \infty$.

$$E[X] \;=\; \sum_{k \in \mathbb{N}} Pr[X > k]\,,$$

where $Pr[X > k]$ denotes the probability that X is greater than $k \in \mathbb{N}$. For a given DTMC $\mathfrak{M} = (S, M)$ with initial distribution $\mathbf{g}$, this probability can be expressed as

$$Pr[X > k] \;=\; \mathbf{g}^\top M^k \mathbf{1}\,.$$

The following definition (especially Item B) is inspired by these observations:

Definition 2.45 ***(Expected runtimes and PAST)*** Let $\mathfrak{M} = (S, M)$ be a DTMC with immediate termination probabilities $\mathbf{t} = \mathbf{1} - M\mathbf{1}$.

A. The vector of *immediate continuation probabilities* of $\mathfrak{M}$ is

$$\mathbf{1} - \mathbf{t} \;=\; M\mathbf{1}\,.$$

B. The *expected runtime* of $\mathfrak{M}$ from initial distribution $\mathbf{g}$ is defined as the behavior

$$[\![\mathbf{g}\,, \mathfrak{M}\,, \mathbf{1} - \mathbf{t}]\!] \;=\; \sum_{k \geq 0} \mathbf{g}^\top M^k (\mathbf{1} - \mathbf{t}) \;=\; \sum_{k \geq 1} \mathbf{g}^\top M^k \mathbf{1}\,.$$

C. $\mathfrak{M}$ is called *positively almost surely terminating* (PAST) w.r.t. initial distribution $\mathbf{g}$ if

$$[\![\mathbf{g}\,, \mathfrak{M}\,, \mathbf{1} - \mathbf{t}]\!] \;<\; \infty\,,$$

that is, the expected runtime is finite.

D. $\mathfrak{M}$ is called *universally PAST* if $\mathfrak{M}$ is PAST relative to *all* initial distributions on S. △

Notice the striking formal similarity between termination probabili-

ties[68] and expected runtimes—the only difference lies in the final weights: $\mathbf{t}$ for termination probabilities and the *complementary* vector $\mathbf{1} - \mathbf{t}$ for expected runtimes.

68: See Definition 2.40 on page 37.

The following observation is analogous to Lemma 2.41B:

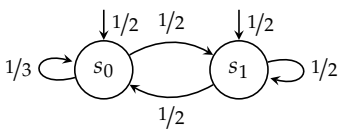
Lemma 2.46 *Let $\mathfrak{M} = (S, M)$ be a DTMC with immediate termination probabilities $\mathbf{t} = \mathbf{1} - M\mathbf{1}$. Then*

$$\mathfrak{M} \text{ is universally PAST} \iff \langle\!\langle \mathfrak{M}, \mathbf{1} - \mathbf{t} \rangle\!\rangle \prec \infty ,$$

where $\prec$ denotes componentwise strict *inequality.*

Proof. Immediate consequence of Definition 2.45. □

► **Example 2.47** (Expected runtimes) Consider the following DTMC with two states and a uniform initial distribution:



The DTMC is improper as s_0 lacks an outgoing probability of $1-1/2-1/3 = 1/6$; therefore, the vector of immediate termination probabilities is

$$\mathbf{t} = \mathbf{1} - M\mathbf{1} = \begin{pmatrix} 1/6 \\ 0 \end{pmatrix} .$$

The expected runtimes $\langle\!\langle \mathfrak{M}, \mathbf{1} - \mathbf{t} \rangle\!\rangle$ of the DTMC can be determined using the linear equation system characterization from Proposition 2.31 on page 32 as follows. We set up and solve the linear equation system $M\boldsymbol{x} + (\mathbf{1} - \mathbf{t}) = \boldsymbol{x}$ which, in this example, reads:

$$\begin{pmatrix} 1/3 & 1/2 \\ 1/2 & 1/2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} 5/6 \\ 1 \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

This linear equation system has the unique solution $x_1 = 11$ and $x_2 = 13$. Taking the initial distribution into account, the expected runtime is therefore $1/2 \cdot 11 + 1/2 \cdot 13 = 12$. This number can be approximately confirmed in experiments; see Figure 2.3. ◄

▷ **Remark 2.48** (Limitations of expected values) The plot in Figure 2.3 moreover gives an idea of the actual *distribution* of the runtime of the DTMC from Example 2.47. Even though the expected runtime is exactly 12, we stress that it would be wrong to conclude that runtimes in the vicinity of 12 are typical or particularly frequent outcomes. Indeed, the experiment shows that it is significantly more likely for the runtime to fall into the interval, say,

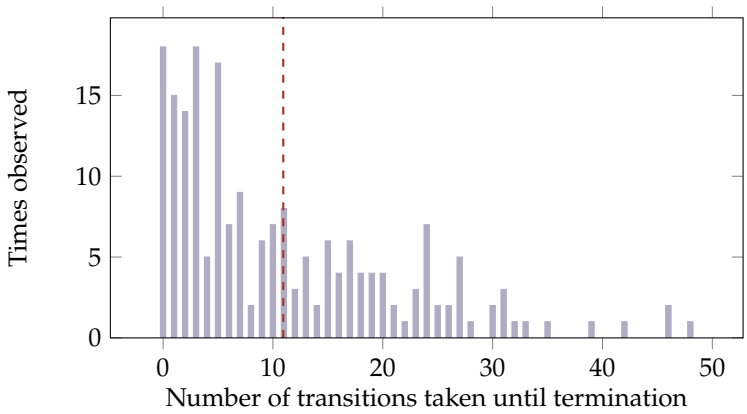


Figure 2.3: Visualization of the lengths of 200 sample runs of the DTMC from Example 2.47. The dashed line is the empirical average, which was 10.94 in this experiment.

$[0, 2]$ than in the interval $[11, 13]$. Of course, the experiment only confirms the basic—yet perhaps somewhat counterintuitive—statistical fact that *the average is not necessarily representative of the sample set*. ◁

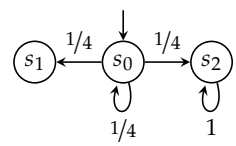


Figure 2.4: DTMC from Examples 2.39 and 2.42 restated.

► **Example 2.49** (Infinite expected runtimes) Reconsider the DTMC from Examples 2.39 and 2.42 (restated in the margin). Since the DTMC terminates with probability strictly smaller than 1 from s_0 and s_2, and terminates immediately in s_1, it is intuitively clear that the expected runtimes are

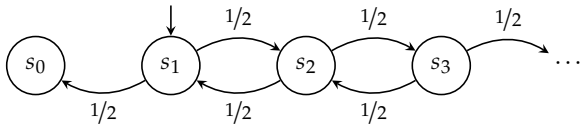
$$\langle\!\langle \mathfrak{M}, \mathbf{1} - \mathbf{t} \rangle\!\rangle = \begin{pmatrix} \infty \\ 0 \\ \infty \end{pmatrix}.$$

The fact that non-almost sure termination implies infinite expected runtime will be proved in general in Section 2.3.4. Alternatively, we may use Proposition 2.31 and verify that the above vector is the least nonnegative (extended) real solution of

$$\begin{pmatrix} 1/4 & 1/4 & 1/4 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} 3/4 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix}.$$

Indeed, the third equation reads $x_2 + 1 = x_2$, which only admits the solution $x_2 = \infty$ and implies that $x_0 = \infty$ as well. The second equation is $x_1 = 0$. ◄

► **Example 2.50** (AST but not PAST) The DTMC from the previous Example 2.49 is not PAST (its expected runtime is infinite); on the other hand, it is not AST either. However, as is well-known, there also exist DTMCs that are AST but not PAST. Such DTMCs are necessarily infinite. A standard example is the following *symmetric random walk* on the positive integers that terminates once it hits zero:



It is possible to establish, again using Proposition 2.31, that this DTMC is AST but not PAST.[69] In Chapter 3, we present an alternative proof exploiting the fact that the random walk can be encoded as a *probabilistic pushdown automaton*. ◀

69: This involves solving a linear equation system with infinitely many variables.

2.3.3 Conditional Expected Runtimes

We introduce a third and final quantity together with an associated notion of termination: The *conditional expected runtime*. This quantity can take on any value from 0 to ∞ (both inclusive) and has the following intuitive meaning: Suppose that we conduct an experiment in which we collect n *terminating* runs (computation paths) of the DTMC at hand.[70] The conditional expected runtime is then the average length of these runs we would observe in the limit $n \to \infty$.

70: This is only possible if the termination probability is nonzero.

In the parlance of probability theory, the conditional expected runtime is an instance of a *conditional expected value*—it is the usual expected runtime[71] *conditioned* on the event of termination. The termination probability must be positive for the conditional expected runtime to be well-defined. Moreover, if the DTMC is AST, the conditional expected runtime equals its unconditional counterpart.

71: See Section 2.3.2.

Conditional positive almost sure termination (cPAST) means that a DTMC's conditional expected runtime is finite (or undefined). We will see that cPAST does *not* imply AST (i.e., cPAST is not actually a notion of termination in the narrow sense of the term). Intuitively, a DTMC that is cPAST either terminates *fast* (in finite expected time),[72] or not at all.

72: This should be taken with a grain of salt: In some instances, the expected runtime is finite but *huge*; see Example 3.33.

Conditional expected runtimes and cPAST, while being of fundamental importance in this thesis, have an auxiliary character and are primarily introduced for technical reasons explained in detail in Chapter 4. Suffice it to say here that cPAST characterizes an important algebraic property of the nonlinear equations associated with probabilistic pushdown automata (to be introduced in Chapter 3).

Before summarizing the formalities in Definition 2.51 on the following page, we explain our approach for expressing conditional expected runtimes in the weighted automata framework. The key idea is to consider a variant of a DTMC's runtime where the infinite, nonterminating runs contribute a runtime of *zero*—we call this

auxiliary quantity *termination moment*. Informally, if X denotes the runtime, then the termination moment can be written as

$$\sum_{k \in \mathbb{N}} k \cdot Pr[X = k] \,. \tag{2.3}$$

Assuming a DTMC $\mathfrak{M} = (S, M)$ with initial distribution $\mathbf{g}$, the probability $Pr[X = k]$ in Equation (2.3) can be expressed as $\mathbf{g}^\top \cdot M^k \cdot \mathbf{t}$. It is therefore coherent to define the termination moment as the behavioral moment[73]

$$[\![\mathbf{g}, \mathfrak{M}, \mathbf{t}]\!]^{\langle 1 \rangle} \;=\; \sum_{k \in \mathbb{N}} \mathbf{g}^\top \cdot k \cdot M^k \cdot \mathbf{t} \,.$$

73: See Definition 2.32 on page 32.

We summarize and formalize these notions as follows:

Definition 2.51 ***(Conditional expected runtimes and cPAST)*** Let $\mathfrak{M} = (S, M)$ be a DTMC with immediate termination probabilities $\mathbf{t} = \mathbf{1} - M\mathbf{1}$.

A. The *(first) termination moment*[74] of $\mathfrak{M}$ with regard to initial distribution $\mathbf{g}$ is the first behavioral moment[75]

$$[\![\mathbf{g}, \mathfrak{M}, \mathbf{t}]\!]^{\langle 1 \rangle} \,.$$

B. The *conditional expected runtime* of $\mathfrak{M}$ from initial distribution $\mathbf{g}$ is defined as

$$\frac{[\![\mathbf{g}, \mathfrak{M}, \mathbf{t}]\!]^{\langle 1 \rangle}}{[\![\mathbf{g}, \mathfrak{M}, \mathbf{t}]\!]} \quad = \quad \frac{\text{termination moment}}{\text{termination probability}}$$

if $[\![\mathbf{g}, \mathfrak{M}, \mathbf{t}]\!] > 0$, and undefined otherwise.

C. $\mathfrak{M}$ is *conditionally positively almost surely terminating* (cPAST) from initial distribution $\mathbf{g}$ (or $\mathbf{g}$-cPAST) if

$$[\![\mathbf{g}, \mathfrak{M}, \mathbf{t}]\!] \;>\; 0 \quad \Longrightarrow \quad \frac{[\![\mathbf{g}, \mathfrak{M}, \mathbf{t}]\!]^{\langle 1 \rangle}}{[\![\mathbf{g}, \mathfrak{M}, \mathbf{t}]\!]} \;\leq\; \infty \,,$$

that is, the conditional expected runtime is undefined or finite.

D. $\mathfrak{M}$ is called *universally cPAST* if $\mathfrak{M}$ is cPAST from *all* initial distributions. △

74: Higher termination moments can be defined analogously but are not needed in this thesis.

75: See Definition 2.32 on page 32.

The following is analogous to Lemma 2.46 (recall that "$<$" denotes componentwise strict inequality):

Lemma 2.52 *Let $\mathfrak{M} = (S, M)$ be a DTMC with immediate termination*

probabilities $\mathbf{t} = \mathbf{1} - M\mathbf{1}$. *Then*

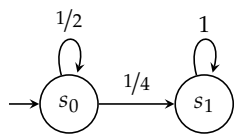
$$\mathfrak{M} \text{ is universally cPAST} \iff \langle\!\langle \mathfrak{M}, \mathbf{t} \rangle\!\rangle^{\langle 1 \rangle} < \infty .$$

Proof. We argue as follows:

$$\begin{aligned}
& \mathfrak{M} \text{ is universally cPAST} \\
\iff\ & \forall \text{ initial distributions } \mathbf{g}\colon \\
& \qquad [\![\mathbf{g}, \mathfrak{M}, \mathbf{t}]\!] > 0 \implies \frac{[\![\mathbf{g}, \mathfrak{M}, \mathbf{t}]\!]^{\langle 1 \rangle}}{[\![\mathbf{g}, \mathfrak{M}, \mathbf{t}]\!]} < \infty \\
& \qquad\qquad\qquad\qquad\qquad\qquad \text{(Definition 2.51)} \\
\iff\ & \forall \text{ initial distributions } \mathbf{g}\colon \\
& \qquad [\![\mathbf{g}, \mathfrak{M}, \mathbf{t}]\!] = 0 \ \vee\ [\![\mathbf{g}, \mathfrak{M}, \mathbf{t}]\!]^{\langle 1 \rangle} < \infty \\
& \qquad\qquad\qquad\qquad\qquad\qquad \text{(Rewriting)} \\
\iff\ & \forall \text{ initial distributions } \mathbf{g}\colon [\![\mathbf{g}, \mathfrak{M}, \mathbf{t}]\!]^{\langle 1 \rangle} < \infty \\
& \qquad\qquad\qquad\qquad\qquad\qquad \text{(Lemma 2.34 on page 33)} \\
\iff\ & \langle\!\langle \mathfrak{M}, \mathbf{t} \rangle\!\rangle^{\langle 1 \rangle} < \infty
\end{aligned}$$

This completes the proof of Lemma 2.52. □

► **Example 2.53** (Conditional expected runtimes) Consider the following DTMC $\mathfrak{M}$ with an initial distribution satisfying $\mathbf{g}(s_0) = 1$:



$\mathfrak{M}$ is improper (s_0 lacks an outgoing probability of $1/4$) and can be shown to terminate with probability $[\![\mathbf{g}, \mathfrak{M}, \mathbf{t}]\!] = 1/2$ from $\mathbf{g}$. Therefore, the (unconditional) expected runtime is infinite, but the *conditional* expected runtime is finite. To determine the latter, we first compute the termination moment,[76] which, in this case, can be written as:

$$[\![\mathbf{g}, \mathfrak{M}, \mathbf{t}]\!]^{\langle 1 \rangle} = \left(1 \cdot \frac{1}{2} + 2 \cdot \frac{1}{4} + 3 \cdot \frac{1}{8} + \ \ldots\ \right) \cdot \frac{1}{4}$$

The infinite sum in the above expression, a variation of the geometric series, converges to 2; the termination moment is thus $2 \cdot 1/4 = 1/2$. The conditional expected runtime can now be determined as

$$\frac{[\![\mathbf{g}, \mathfrak{M}, \mathbf{t}]\!]^{\langle 1 \rangle}}{[\![\mathbf{g}, \mathfrak{M}, \mathbf{t}]\!]} = \frac{1/2}{1/2} = 1 .$$

76: See Definition 2.51 on the previous page.

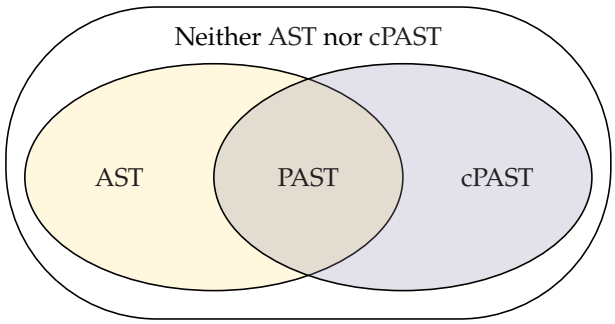


Figure 2.5: A taxonomy of qualitative termination behaviors of DTMCs.

This means that if we collect n terminating (i.e., finite) sample runs of the DTMC, then their average length approaches 1 as n grows large.[77] .... ◀

77: To conduct this experiment in practice, one can simply abort the simulations reaching state s_1.

2.3.4 A Taxonomy of Termination

We now examine the relationship between AST, PAST, and cPAST, the three notions of termination behaviors introduced in Sections 2.3.1 to 2.3.3. As mentioned earlier, the connection between AST and PAST is well-understood (e.g., [105, Theorem 5.3]). Our contribution is to complete the picture with cPAST, and provide self-contained proofs in the weighted automata framework. The resulting *taxonomy of termination* is summarized in Figure 2.5.

[105]: Olmedo et al. (2016), "Reasoning about Recursive Probabilistic Programs".

Theorem 2.54 (Taxonomy of termination) *Let $\mathfrak{M} = (S, M)$ be a DTMC.*

A. *For all initial distributions $\mathbf{g}$ on S, it holds that*

$$\mathfrak{M} \text{ is } \mathbf{g}\text{-PAST} \iff \mathfrak{M} \text{ is } \mathbf{g}\text{-AST and } \mathfrak{M} \text{ is } \mathbf{g}\text{-cPAST}.$$

B. *Similarly, $\mathfrak{M}$ is universally PAST if and only if $\mathfrak{M}$ is both universally AST and universally cPAST.*[78]

C. *For both the universal and the non-universal variant of the respective notion of termination, there exist DTMCs that are*
   - *AST but not cPAST (and hence not PAST);*
   - *cPAST but not AST (and hence not PAST);*
   - *both AST and cPAST (and hence PAST);*
   - *neither AST nor cPAST (and hence not PAST).*

   Thus, none of the four classes in the taxonomy of Figure 2.5 is empty.

78: This is an immediate consequence of Item A.

Proof. Throughout the proof, let $\mathbf{t} = \mathbf{1} - M\mathbf{1}$ be the immediate termination probabilities of $\mathfrak{M}$.

We first observe that, for all $n \in \mathbb{N}$,

$$M^n(\mathbf{1} - \mathbf{t}) \;+\; \sum_{k=0}^{n} M^k \mathbf{t} \;=\; \mathbf{1}\,. \tag{2.4}$$

Equation (2.4) can be shown as follows:

$$\begin{aligned}
& M^n(\mathbf{1} - \mathbf{t}) \;+\; \sum_{k=0}^{n} M^k \mathbf{t} \\
=\;& M^{n+1}\mathbf{1} \;+\; \sum_{k=0}^{n} M^k(\mathbf{1} - M\mathbf{1}) && \text{(Substitute } \mathbf{1} - M\mathbf{1} \text{ for } \mathbf{t}\text{)} \\
=\;& 1\,. && \text{(\textit{Telescope sum})}
\end{aligned}$$

We now prove Item A. Let $\mathbf{g}$ be an arbitrary initial distribution. To show that $\mathbf{g}$-PAST implies $\mathbf{g}$-AST, we argue as follows:

$$\begin{aligned}
& \mathfrak{M} \text{ is } \mathbf{g}\text{-PAST} \\
\iff\;& [\![\mathbf{g}, \mathfrak{M}, \mathbf{1} - \mathbf{t}]\!] \;<\; \infty && \text{(Definition of PAST)} \\
\iff\;& \mathbf{g}^\top M^*(\mathbf{1} - \mathbf{t}) \;<\; \infty && \text{(Definition of behavior)} \\
\iff\;& \sum_{n \geq 0} \mathbf{g}^\top M^n(\mathbf{1} - \mathbf{t}) \;<\; \infty && \text{(Definition of } M^*\text{)} \\
\implies\;& \lim_{n \to \infty} \mathbf{g}^\top M^n(\mathbf{1} - \mathbf{t}) \;=\; 0 \\
& \qquad \text{(Because the series in the preceding line converges)} \\
\iff\;& \lim_{n \to \infty} \sum_{k=0}^{n} \mathbf{g}^\top M^k \mathbf{t} \;=\; 1 \\
& \qquad \text{(Multiply (2.4) by } \mathbf{g}^\top \text{ and take limits on both sides)} \\
\iff\;& \mathbf{g}^\top M^* \mathbf{t} \;=\; 1 && \text{(Definition of } M^*\text{)} \\
\iff\;& \mathfrak{M} \text{ is } \mathbf{g}\text{-AST}\,. && \text{(Definition of AST)}
\end{aligned}$$

Next, we prove that $\mathbf{g}$-PAST implies $\mathbf{g}$-cPAST by showing a slightly stronger statement: The expected runtime $[\![\mathbf{g}, \mathfrak{M}, M\mathbf{1}]\!]$ equals the termination moment $[\![\mathbf{g}, \mathfrak{M}, \mathbf{t}]\!]^{\langle 1 \rangle}$ whenever $\mathfrak{M}$ is $\mathbf{g}$-AST (the latter is implied by $\mathbf{g}$-PAST as we have shown above):

$$\begin{aligned}
& [\![\mathbf{g}, \mathfrak{M}, M\mathbf{1}]\!] \\
=\;& \sum_{n \geq 0} \mathbf{g}^\top M^n M\mathbf{1} && \text{(Definitions)} \\
=\;& \sum_{n \geq 0} \left(1 - \sum_{k=0}^{n} \mathbf{g}^\top M^k \mathbf{t}\right) \\
& && \text{(Multiply (2.4) by } \mathbf{g}^\top \text{ and rearrange)}
\end{aligned}$$

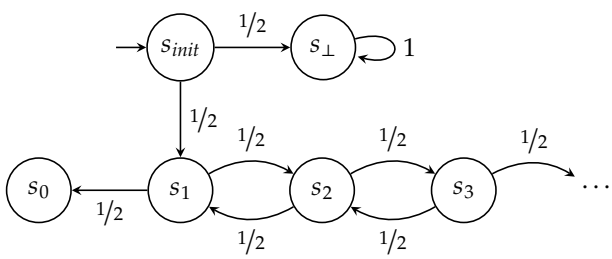
$$
\begin{aligned}
&= \sum_{n\geq 0}\left(\sum_{\ell\geq 0}\mathbf{g}^\top M^\ell \mathbf{t} - \sum_{k=0}^{n}\mathbf{g}M^k\mathbf{t}\right) && \text{(Since } \mathfrak{M} \text{ is } \mathbf{g}\text{-AST)}\\
&= \sum_{n\geq 0}\sum_{\ell> n}\mathbf{g}^\top M^\ell \mathbf{t} && \text{(Rearranging)}\\
&= \mathbf{g}^\top M^* M M^* \mathbf{t} && \text{(Argument as in proof of Lemma 2.35)}\\
&= [\![\mathbf{g}, \mathfrak{M}, \mathbf{t}]\!]^{\langle 1\rangle}\,. && \text{(Lemma 2.35 and definition)}
\end{aligned}
$$

To conclude the proof of Item A it remains to show that **g**-AST and **g**-cPAST together imply **g**-PAST. But we have just shown that **g**-AST entails equal expected runtime and termination moment relative to **g**. In particular, if the termination moment is finite, the expected runtime is finite as well, that is, $\mathfrak{M}$ is **g**-PAST.

Item B of Theorem 2.54 follows immediately from Item A.

It remains to specify example DTMCs as stated in Item C.

- The symmetric random walk DTMC from Example 2.50 on page 41 is AST (even universally AST) but not PAST; hence it is not cPAST either.
- See Example 2.53 on page 44 for a DTMC which is cPAST but not AST.
- The DTMC from Example 2.47 on page 40 is PAST, and therefore both AST and cPAST. The same example is, in fact, *universally* PAST.
- For the last case—neither AST nor cPAST—consider the following DTMC, a modification of the random walk from Example 2.50:



  Starting in s_{init}, the termination probability is only $1/2$. Moreover, the *conditional* expected runtime is infinite: Even if termination occurs (i.e., s_0 is reached), the expected number of steps is infinite because the chain has to traverse the symmetric random walk before reaching s_0.

This completes the proof of Theorem 2.54 □

2.4 Additional Bibliographic Remarks

Many authors (e.g., Droste and Kuske [38]) attribute weighted automata to Schützenberger's seminal 1961 paper "On the Definition of a Family of Automata" [119]. Today, there is a vast body of literature on weighted automata theory; the *Handbook of Weighted Automata* [37] serves as a good starting point for readers interested in the topic.

[38]: Droste et al. (2021), "Weighted automata".

[119]: Schützenberger (1961), "On the Definition of a Family of Automata".

[37]: Droste et al. (2009), *Handbook of Weighted Automata*.

As mentioned earlier, various notions introduced in this chapter are due to Kuich's chapter in the *Handbook of Formal Languages, Volume 1: Word, Language, Grammar* [87]. However, there are some notable technical differences.

[87]: Kuich (1997), "Semirings and Formal Power Series: Their Relevance to Formal Languages and Automata".

First, our definition of ω-continuous semirings[79] is slightly less general than that of Kuich, which requires sums over *arbitrary* index sets, not merely countable ones. To accommodate this, Kuich introduces the notion of *completeness*: a complete semiring is equipped with a general sum operator satisfying certain axioms (e.g., commutativity and distributivity). In contrast, we require ω-continuous semirings to form an ω-complete partial order and *define* countably infinite sums as suprema in this order.[80] As a result, an ω-continuous semiring in our sense is also ω-continuous in the sense of Kuich, except that uncountable sums remain undefined. We invoke only those results from [87] that require at most countable sums. Our notion of ω-continuous semirings aligns with usage elsewhere in the literature (e.g., [17, 41]).

79: See Definition 2.9 on page 19.

80: See Definition 2.5 on page 18.

[17]: Batz et al. (2022), "Weighted programming: a programming paradigm for specifying mathematical models".

[41]: Ésik (2008), "Iteration Semirings".

Second, beyond ω-continuity, our weighted automata definition also differs from Kuich's [87] in two respects. First, our definition permits weighted automata with unspecified initial and final weights. Second, and more substantially, Kuich's definition allows designating a *subset* of semiring elements as admissible weights on transitions. This is often natural: for instance, a traditional NFA over an alphabet Σ can be viewed as a weighted automaton over the formal languages semiring where only nonempty subsets of Σ occur on transitions. This setup naturally raises the question: *Which elements of the base semiring can be expressed as behaviors of finite automata?* For NFA, Kleene's celebrated theorem provides the answer: the behaviors of NFA are precisely the regular languages. A major achievement of weighted automata theory is the generalization of such results beyond formal languages (e.g., [87, Theorem 4.10]).

Pushdown Automata 3

This chapter generalizes the weighted automata framework from Chapter 2 to *weighted pushdown automata* (WPDA).

3.1 Weighted PDA . 51
3.1.1 Definition 51
3.1.2 Semantics 54
3.1.3 Behavioral Moments 63
3.2 Probabilistic PDA 71
3.2.1 Termination Probabilities . . . 71
3.2.2 Expected Runtimes 76
3.2.3 Conditional Expected Runtimes 80
3.3 Open Questions . 83
3.4 Bibliographic Remarks 83

Weighted Pushdown Automata The WPDA considered here are similar to the (generative) weighted automata from Chapter 2, but augmented with an additional *stack*. Alongside generating semiring elements, transitions in a WPDA can manipulate the stack via the standard interface: pushing, popping, and swapping the topmost element. The stack is assumed to have *infinite capacity*. This makes finite WPDA strictly more expressive than finite weighted automata: There exist WPDA behaviors that cannot be generated with finite weighted automata.[1] See Section 1.1 for additional introductory material to stacks, pushdown automata, and recursion.

1: For instance, finite WPDA can generate irrational behaviors even if all weights are rational numbers.

The Fundamental Equation System We saw in Chapter 2 that the behaviors of finite weighted automata can be characterized via *linear* equation systems.[2] For finite WPDA, these equation systems become infinitely large due to the stack, rendering them impractical. Nevertheless, as is well-known, certain WPDA behaviors can be characterized through a finite—but *nonlinear*—equation system [87], referred to as the *fundamental equation system* in the remainder.[3] We will see that the fundamental system is actually a *fixed point* equation system associated with a certain multivariable *polynomial* function, which is amenable to various algorithmic techniques. Nonetheless, nonlinearity makes the analysis of WPDA overall more challenging compared to the non-pushdown case.

2: See Proposition 2.31 on page 32.

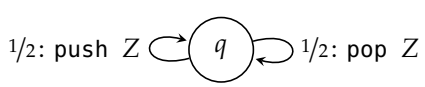
[87]: Kuich (1997), "Semirings and Formal Power Series: Their Relevance to Formal Languages and Automata".

3: See Theorem 3.11 on page 58.

Probabilistic Pushdown Automata Of particular interest in this thesis are *probabilistic pushdown automata* (pPDA) [44]. They relate to WPDA in the same way that DTMCs (discrete-time Markov chains)[4] relate to weighted automata: Their transition weights are probabilities summing to at most one across all transitions available in each configuration. An informal example of a pPDA is as follows:

[44]: Esparza et al. (2004), "Model Checking Probabilistic Pushdown Automata".

4: See Definition 2.38 on page 35.



This pPDA has just one state, q, and one possible stack entry, Z. In each step, the automaton either pushes or pops, both with probability $1/2$, thereby performing an unbiased *random walk* on the stack height. Once the stack is empty, no further transitions are possible and the pPDA terminates.

General pPDA—with an arbitrary number of states and stack symbols—are a natural model for recursive probabilistic programs with discrete random choices [44]. In Chapter 7, we describe an explicit translation from a recursive probabilistic programming language to pPDA.

Termination of pPDA A key question in the analysis of pPDA is to determine (or approximate) the *probability* of termination. In our simple example, the termination probability is the solution[5] to its fundamental equation

$$x = \frac{1}{2} + \frac{1}{2}x^2 .$$

Here, since the pPDA has just one state and one stack symbol, the equation system consists of a single equation only. In general, the fundamental system may involve up to *cubically* many variables in the size of the given WPDA.

5: The solution is unique in this particular example; in general, we are interested in the *least* solution.

Chapter Outline, Contributions, and Origins This chapter roughly parallels Chapter 2:

Section 3.1 (Weighted Pushdown Automata) We define WPDA and present their nonlinear fundamental equation systems [87]. Moreover, we present several—to our knowledge novel—additional equation systems: for *simple* final weight vectors [21], and for behavioral moments.[6]

Section 3.2 (Probabilistic Pushdown Automata) We instantiate the theory from Section 3.1 with pPDA, deriving explicit equation systems for termination probabilities and (conditional) expected runtimes. On the technical side, we prove a characterization of cPAST (conditional positive almost sure termination)[7], which will be fundamentally important in later chapters.

This chapter relies in part on our previous publication [135]. Additional bibliographic remarks are postponed to Section 3.4.

[21]: Brázdil et al. (2013), "Analyzing probabilistic pushdown automata".

6: See Definition 2.32 on page 32.

7: See Definition 2.51 on page 43.

[135]: Winkler et al. (2023), "On Certificates, Expected Runtimes, and Termination in Probabilistic Pushdown Automata".

3.1 Weighted Pushdown Automata

As mentioned, WPDA extend weighted automata by a stack. It is customary to model the contents of this stack as a word over a certain *stack alphabet* Γ. We use the notation

$$\Gamma^{\leq k} = \bigcup_{i=0}^{k} \Gamma^i$$

for the set of words (or stacks) over Γ of length *at most* $k \in \mathbb{N}$.[8] We moreover agree on the following convention, which is standard in the literature and will rest in place for the remainder of the thesis:

8: Notice that this includes $\Gamma^{\leq 0} = \{\varepsilon\}$ as a corner case.

Convention 3.1 Stacks are read *from top to bottom*.

For example, the stack

$$\begin{matrix} Z \\ Y \\ X \end{matrix} \quad \text{is identified with the word} \quad ZYX\,.$$

Consequently, pushing to and popping from the *top* of a stack amounts to manipulating the *leftmost* symbol of the stack's transcription as a word. For instance, popping from ZYX yields YX. If we then push X, we obtain XYX, and so on.

3.1.1 Formal Definition and Basic Examples

Our definition of WPDA is similar to the one given by Kuich [87, Section 6]:

Definition 3.2 ***(WPDA: Weighted pushdown automaton)*** Let $\mathcal{S}$ be an ω-continuous semiring[9] with carrier set W. An $\mathcal{S}$-WPDA is a triple $\Delta = (Q, \Gamma, P)$, where:

9: See Definition 2.9 on page 19.

- Q is a countable set called *state space*.
- Γ is a countable set called *stack alphabet*.
- $P\colon (Q \times \Gamma) \times (Q \times \Gamma^{\leq 2}) \to W$ is a matrix[10] with coefficients in W called *pushdown transition matrix*.

10: See Definition 2.20 on page 25.

Δ is called *finite* if both Q and Γ are finite sets. △

The state-stack tuples in $Q \times \Gamma^*$ are called *configurations* and are denoted without comma and parentheses. That is, we write

$$q\gamma \in Q \times \Gamma^*$$

rather than $(q, \gamma) \in Q \times \Gamma^*$. The set of *heads* is the subset $Q \times \Gamma$ of configurations.

Note that we consider only WPDA over *ω-continuous* semirings.[11] Moreover, we deliberately refrain from including initial and final weights in the definition. This is analogous to our definition of (non-pushdown) weighted automata.[12] Additionally, unlike in many definitions in the literature, our WPDA do not have a fixed initial stack symbol either.

11: See Definition 2.9 on page 19.

12: See Definition 2.24 on page 28.

While our definition allows WPDA with countably infinite state space and stack alphabet,[13] most results we present apply to *finite* WPDA only.

13: Such infinite WPDA play a (minor) role in Chapter 7.

We now examine Definition 3.2 in greater detail and provide some intuition. The pushdown transition matrix P is the heart of any WPDA. The intuitive meaning of a coefficient $P(qZ, r\gamma) = w$ is as follows:

q	If the automaton is in state q,
Z	... and the current topmost stack symbol is Z,
w	... then the automaton may generate weight w,
r	... transition to state r,
γ	... and replace the topmost stack symbol by γ.

Furthermore, since $\gamma \in \Gamma^{\leq 2}$ by definition, we either have

1. $\gamma = \varepsilon$, in which case the topmost symbol Z is effectively *popped*;
2. $\gamma = Y \in \Gamma$, meaning that the topmost symbol Z is *swapped* with Y, without in- or decreasing the height of the stack; or
3. $\gamma = YX \in \Gamma^2$, which means that the topmost symbol Z is swapped with X and addionally Y is *pushed* on the stack, becoming the new topmost symbol.[14]

14: Notice that this is consistent with Convention 3.1.

Since all types of transitions depend on the current topmost stack symbol, a WPDA cannot take any further transitions once its stack is empty.

[44]: Esparza et al. (2004), "Model Checking Probabilistic Pushdown Automata".

We shall often use the following more intuitive notation for transitions, which is standard in the literature (e.g., [44]):

$$qZ \xrightarrow{w}_{\Delta} r\gamma \quad \overset{\text{def}}{\iff} \quad P(qZ, r\gamma) = w \ .$$

Note that, by case distinction over the length of γ, the notation

concretizes to either

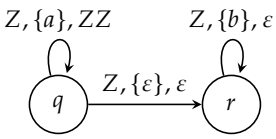
$$\underbrace{qZ \xrightarrow{w}_\Delta r}_{\text{pop}}\,, \qquad \text{or} \qquad \underbrace{qZ \xrightarrow{w}_\Delta rZ}_{\text{swap}}\,, \qquad \text{or} \qquad \underbrace{qZ \xrightarrow{w}_\Delta rZX}_{\text{push}}\,.$$

We may omit the subscript Δ if the automaton is clear from the context.

Let us consider some examples next.

► **Example 3.3** (WPDA generalize standard PDA) Consider the following finite WPDA over the formal languages semiring[15] $\mathcal{L}_\Sigma$ with $\Sigma = \{a, b\}$:

15: Recall that the elements of the formal languages semiring are *sets* of finite words; see Example 2.8.



The automaton has states $Q = \{q, r\}$ and stack alphabet $\Gamma = \{Z\}$. The comma-separated objects on the transitions denote (in this order):

1. The topmost stack symbol required for the transition to be enabled.
2. The generated weight.[16]
3. The string to be substituted for the topmost stack symbol.

16: In our example, all weights are singleton formal languages.

For example, the annotation "$Z, \{a\}, ZZ$" on the self-loop at state q means that the automaton has the transition

$$qZ \xrightarrow{\{a\}} qZZ\,.$$

In general, if a particular transition is *omitted* in the graphical representation, then the corresponding weight is assumed to be the semiring zero. For example, the above automaton satisfies

$$rZ \xrightarrow{\emptyset} q\gamma$$

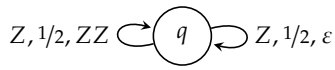
for all $\gamma \in \Gamma^{\leq 2}$. In the remainder, we often depict WPDA using this graphical style.

Interpreted as a standard PDA with initial state q, initial stack symbol Z, and acceptance upon reaching the empty stack, the above automaton recognizes the prototypical context-free (but non-regular) language $\{a^n b^n \mid n \in \mathbb{N}\}$. In Section 3.1.2 on the following page, we define the behavior (semantics) of general WPDA and provide tools to determine the behavior systematically. We will see that, using appropriate initial and final weights, the behavior of the above example automaton is indeed the language $\{a^n b^n \mid n \in \mathbb{N}\}$.[17] ◄

17: See Example 3.12 on page 58.

► **Example 3.4** (A probabilistic PDA) The following WPDA is defined over the extended nonnegative real number semiring $\overline{\mathcal{R}}_{\geq 0}$:[18]

18: This automaton is the formal version of the informal example from the introduction of this chapter.



This automaton is similar to a DTMC[19] because the weights on the transitions are probabilities summing up to 1. Intuitively, the automaton performs an unbiased *random walk* on the height of its stack. Indeed, as will be explained in Example 3.6, this example is closely related to the random walk DTMC from Example 2.50 on page 41.

19: See Definition 2.38 on page 35.

DTMC-like WPDA as illustrated in this example are called *probabilistic pushdown automata* (pPDA) [44]. They are the main object of interest in this thesis. Section 3.2 on page 71 is devoted to a thorough introduction to pPDA. ◀

[44]: Esparza et al. (2004).

3.1.2 Semantics of Weighted Pushdown Automata

Following [87], we view WPDA semantically as generators of (non-pushdown) weighted automata by encoding the stack content into the states. Crucially, this leads to weighted automata with *infinitely many states* in general.[20] The following definition formalizes this idea:

[87]: Kuich (1997).

20: See Example 3.6.

Definition 3.5 ***(Induced weighted automaton)*** Let $\Delta = (Q, \Gamma, P)$ be an $\mathcal{S}$-WPDA. The induced weighted automaton of Δ is the $\mathcal{S}$-weighted automaton[21]

$$\mathfrak{A}_\Delta \;=\; (\, Q \times \Gamma^*,\, M_\Delta \,)$$

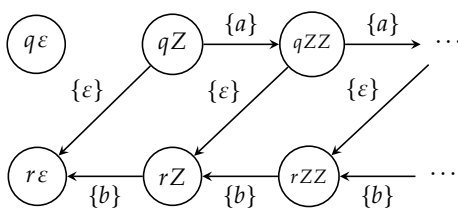
whose transition matrix M_Δ is defined as follows: For all $qZ \in Q \times \Gamma$, $r\gamma \in Q \times \Gamma^{\leq 2}$, and $\eta \in \Gamma^*$,

$$M_\Delta \left(qZ\eta,\, r\gamma\eta \right) \;=\; P\left(qZ,\, r\gamma \right)$$

and all other coefficients of M_Δ are 0, the zero of the semiring $\mathcal{S}$. △

21: See Definition 2.24 on page 28.

► **Example 3.6** (Induced weighted automata) The automaton from Example 3.3 on the preceding page (restated in the margin) induces the following *infinite* weighted automaton over the formal languages semiring:



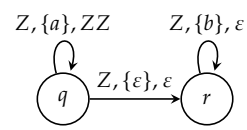
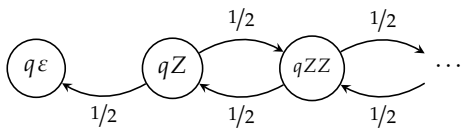


Figure 3.1: Example 3.3 restated.

Notice that the above automaton generates the set of words $\{a^n b^n \mid n \in \mathbb{N}\}$ from initial state qZ. The configuration $q\varepsilon$ is not reachable from any other configuration and could be removed for most purposes.

The pPDA from Example 3.4 on page 53 (also restated in the margin) induces the infinite random walk DTMC from Example 2.50 on page 41:



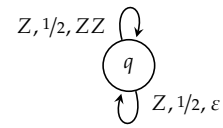


Figure 3.2: Example 3.4 restated.

This structure extends indefinitely toward the right. ◀

We stress that induced weighted automata are generally infinite by definition.[22] However, there also exist cases where, unlike in Example 3.6, each configuration can only reach *finitely* many other configurations.

22: The only exception are trivial WPDA with an empty state space or stack alphabet.

We now define the *behavior* of a WPDA, which is simply the behavior of its induced weighted automaton:

Definition 3.7 ***(Behavior of a WPDA)*** Let $\Delta = (Q, \Gamma, P)$ be a WPDA over a semiring with carrier set W. Further, let $\mathfrak{A}_\Delta = (Q \times \Gamma^*, M_\Delta)$ be the induced weighted automaton of Δ, and let $\mathbf{g}, \mathbf{f} \in W^{Q \times \Gamma^*}$ be vectors[23] of initial (final, resp.) weights on the configurations $Q \times \Gamma^*$.

23: Even though the index set of these vectors is a cross product, they should *not* be viewed as matrices.

We define the *behavior* of Δ with respect to $\mathbf{g}$ and $\mathbf{f}$ as

$$[\![\mathbf{g}, \Delta, \mathbf{f}]\!] \overset{\text{def}}{=} [\![\mathbf{g}, \mathfrak{A}_\Delta, \mathbf{f}]\!] = \mathbf{g}^\top \cdot M_\Delta^* \cdot \mathbf{f} ,$$

that is, the behavior of the induced weighted automaton. The *forward* and *backward* behaviors[24]

$$[\![\mathbf{g}, \Delta\rangle\!\rangle \quad \text{and} \quad \langle\!\langle \Delta, \mathbf{f}]\!] ,$$

24: See Definition 2.28 on page 30.

as well as the nth *behavioral moments*[25] ($n \in \mathbb{N}$)

$$[\![\mathbf{g}, \Delta, \mathbf{f}]\!]^{\langle n \rangle} , \quad [\![\mathbf{g}, \Delta\rangle\!\rangle^{\langle n \rangle} , \quad \text{and} \quad \langle\!\langle \Delta, \mathbf{f}]\!]^{\langle n \rangle}$$

25: See Definition 2.32 on page 32.

are defined similarly. △

3.1.2.1 Return Triples and the Fundamental Equation System

The following special WPDA behaviors turn out to be of central importance to pushdown automata in general, and to our work in particular:

Definition 3.8 ***(Return triples)*** Let $\Delta = (Q, \Gamma, P)$ be a WPDA. We define the return triple for a head $qZ \in Q \times \Gamma$ and a state $r \in Q$ as the behavior[26]

$$[qZ\,r] \quad = \quad [\![\mathbf{e}_{qZ}\,, \Delta\,, \mathbf{e}_{r\varepsilon}]\!]\,. \qquad \triangle$$

26: See Definition 3.7 on the previous page.

Recall from Section 2.1.4 that $\mathbf{e}_{qZ}$ and $\mathbf{e}_{r\varepsilon}$ denote the indicator vectors of the configurations given in the subscripts. Therefore, the triple $[qZ\,r]$ is the sum of the weights[27] of the—possibly infinitely many—paths of finite length from qZ to $r\,\varepsilon$ in the induced weighted automaton.

27: Recall that the weight of a path is the semiring product of the weights of the transitions along that path.

▷ **Remark 3.9** (On the terminology) The shorthand triple notation is very common in the literature (e.g., [44, 70]). However, there seems to be no widely accepted terminology for the triples $[qZ\,r]$ in the literature. Here, we propose the term *return triple*, which is inspired by program analysis. Roughly speaking, when modeling a procedural—possibly recursive—program as a PDA, the paths from qZ to $r\,\varepsilon$ correspond to the runs of the program from some point within a procedure (the precise point is encoded in qZ) until that procedure terminates, possibly *returning* a value (encoded in r). ◁

[44]: Esparza et al. (2004).
[70]: Hopcroft et al. (2001).

► **Example 3.10** Reconsider the traditional PDA from Example 3.3 on page 53 (restated in the margin). The return triple $[qZ\,r]$ is the set of accepted words if we assume the automaton starts in q with initial stack Z and accepts upon reaching the empty stack in state r. Here,

$$[qZ\,r] \;=\; \{a^n b^n \mid n \in \mathbb{N}\}\,.$$

We will prove this formally in Example 3.12 on page 58. ◄

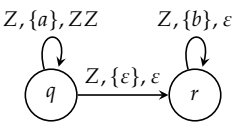


Figure 3.3: Example 3.3 restated.

It is known that the return triples of a finite WPDA are related to each other via certain *polynomial equations* [87, Corollary 6.5]. Many results and algorithmic techniques for pushdown automata rely on this fact. For instance, the textbook translation from traditional PDA to context-free grammars (e.g., [70]) can be seen as an instance of this construction.

[87]: Kuich (1997).
[70]: Hopcroft et al. (2001).

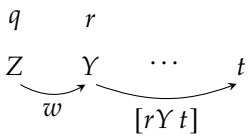
We provide some intuition before stating the polynomial equations formally in Theorem 3.11 on page 58. Let $\Delta = (Q, \Gamma, P)$ be a WPDA. The key idea to derive a polynomial equation

$$[qZ\,t] \;=\; \ldots$$

for a given return triple $[qZ\,t]$ is to characterize the paths from qZ to $t\,\varepsilon$. For this sake, one considers the transitions available at head qZ. Depending on the type (pop, swap, push), we obtain a different term for each of these transitions, which we then sum up in the underlying semiring.

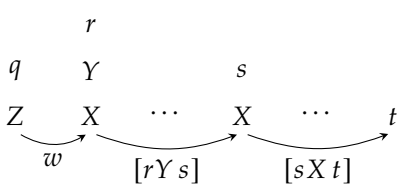
Pop transitions are the easiest case. A pop transition $qZ \xrightarrow{w} r$ contributes weight w to $[qZ\,t]$ if and only if $r = t$. Otherwise it contributes the semiring zero.

Next, consider a swap transition $qZ \xrightarrow{w} rY$. In this case, the situation can be visualized as follows:



The swap transition yields the immediate successor configuration rY and generates weight w. By definition, the sum of the weights of the paths from rY to the target configuration $t\varepsilon$ is $[rY\,t]$. The swap transition $qZ \xrightarrow{w} rY$ thus contributes a weight of $w \cdot [rY\,t]$ to $[qZ\,t]$.

Push transitions are more involved. The situation that results from taking a push transition $qZ \xrightarrow{w} rYX$ can be depicted as follows:



After pushing Y, the stack is YX—it now has height *two*. On any path that reaches the empty stack, the topmost symbol Y must be popped eventually, leading to a configuration sX, where $s \in Q$ is some state.[28] The key observation is that the sum of the weights of the paths from rYX to sX is equal to $[rY\,s]$. The intuitive reason is that these paths are in one-to-one correspondence with the paths from rY to $s\varepsilon$ as visualized in the following diagram (the stack may not drop below the dashed lines):

28: s could be a different state for different paths. In general, any state is possible.



In summary, the push transition $qZ \xrightarrow{w} rYX$ contributes the weight $w \cdot \sum_{s \in Q} [rY\,s] \cdot [sX\,t]$ to $[qZ\,t]$. Notably, unlike pop and swap transitions, push transitions yield *nonlinear* terms.

We now state the equation system arising from the above considerations. The system may have multiple solutions; it turns out that the actual return triples constitute the *least solution*:

Theorem 3.11 (Fundamental equation system [87, Corollary 6.5]) *Let $\Delta = (Q, \Gamma, P)$ be a finite WPDA*[29].

A. *For all $qZ \in Q \times \Gamma$ and $t \in Q$, the return triple*[30] *$[qZ\,t]$ satisfies the following equation:*

$$[qZ\,t] = \sum_{qZ \xrightarrow{w} t} w \qquad \text{(Pop}^{31}\text{)}$$
$$+ \sum_{qZ \xrightarrow{w} rY} w \cdot [rY\,t] \qquad \text{(Swap)}$$
$$+ \sum_{qZ \xrightarrow{w} rYX} w \cdot \sum_{s \in Q} [rY\,s] \cdot [sX\,t] \qquad \text{(Push)}$$

B. *Assume we treat the symbols $[qZ\,r]$, where $qZ \in Q \times \Gamma$ and $r \in Q$, in the equations from Item A as variables.*[32] *Then the return triples constitute the* componentwise least solution[33] *of the resulting equation system, called* fundamental equation system *of Δ.*

Additional notation: We occasionally denote the fundamental equation system of Δ in vector notation as follows:

$$\mathbf{r} = f_\Delta(\mathbf{r})$$

Here, $\mathbf{r}$ is a vector of triples[34] and f_Δ is a vector-valued polynomial function of the triples, i.e., f_Δ is of type $W^{Q \times \Gamma \times Q} \to W^{Q \times \Gamma \times Q}$, where W is the carrier set of the underlying semiring.

29: See Definition 3.2 on page 51.

30: See Definition 3.8 on page 56.

31: This sum is equal to the coefficient $P(qZ, t\varepsilon)$ of the transition matrix P. However, the sum notation is common in the literature [44] and highlights the similarity with the other two cases.

32: The number of variables is thus $|Q|^2 \cdot |\Gamma|$.

33: With respect to the natural order of the semiring

34: More formally, $\mathbf{r}$ is a column vector indexed by $Q \times \Gamma \times Q$.

[87]: Kuich (1997).

We refer the reader to [87] for a detailed proof of Theorem 3.11.

► **Example 3.12** (Fundamental equation system of a standard PDA) The WPDA over the formal languages semiring from Example 3.3 on page 53 is

restated in the margin. Its fundamental equation system is as follows:

$$\begin{aligned}
[qZ\,q] &= \{a\} \cdot \big([qZ\,q] \cdot [qZ\,q] \cup [qZ\,r] \cdot [rZ\,q]\big) \\
[qZ\,r] &= \{\varepsilon\} \cup \{a\} \cdot \big([qZ\,q] \cdot [qZ\,r] \cup [qZ\,r] \cdot [rZ\,r]\big) \\
[rZ\,q] &= \emptyset \\
[rZ\,r] &= \{b\}
\end{aligned}$$

The first two equations can be simplified to:

$$\begin{aligned}
[qZ\,q] &= \{a\} \cdot [qZ\,q] \cdot [qZ\,q] \\
[qZ\,r] &= \{\varepsilon\} \cup \{a\} \cdot \big([qZ\,q] \cdot [qZ\,r] \cup [qZ\,r] \cdot \{b\}\big)
\end{aligned}$$

The least solution with respect to set inclusion[35] satisfies $[qZ\,q] = \emptyset$. The equation for $[qZ\,r]$ can thus be simplified as follows:

$$[qZ\,r] = \{\varepsilon\} \cup \{a\} \cdot [qZ\,r] \cdot \{b\}$$

Its least solution is $[qZ\,r] = \{a^n b^n \mid n \in \mathbb{N}\}$, as expected.

Notice that constructing the fundamental equations of a traditional PDA as done in this example resembles the standard text-book translation of the PDA to a context-free grammar (e.g., [70]). ◀

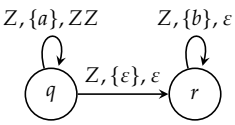


Figure 3.4: Automaton from Example 3.3 restated.

35: Set inclusion is the natural order of the formal languages semiring.

[70]: Hopcroft et al. (2001).

► **Example 3.13** (Fundamental equation system of a pPDA) The fundamental equation system of the pPDA from Example 3.4 on page 53 (restated in the margin) consists only of the following equation[36]:

$$[qZ\,q] = \frac{1}{2} + \frac{1}{2}[qZ\,q] \cdot [qZ\,q]$$

This quadratic equation has the two solutions 1 and ∞ in $\overline{\mathbb{R}}_{\geq 0}$. Theorem 3.11B asserts that the smallest solution is the one of interest, that is, $[qZ\,q] = 1$. Intuitively, this means that the automaton reaches the empty stack eventually with probability one if it is started with stack content Z. Alternatively, one could say that the induced DTMC is AST (almost surely terminating[37]) from the initial configuration qZ. See Section 3.2.1 for details on AST in pPDA. ◀

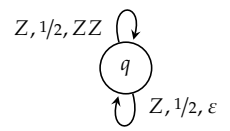


Figure 3.5: Example 3.4 restated.

36: Because the state space and the stack alphabet are both singletons.

37: See Definition 2.40 on page 37.

Additional instances of the fundamental equation system appear in subsequent examples and proofs (e.g., in the proof of Theorem 3.15 and in Example 3.24). An algorithm for solving the equations approximately over the extended real numbers semiring $\overline{\mathcal{R}}_{\geq 0}$ will be presented in Chapter 5.

3.1.2.2 Simple Final Weight Vectors

The fundamental equation system[38] characterizes only specific WPDA behaviors—those with initial and final weights of the form

38: See Theorem 3.11 on the preceding page.

$\mathbf{e}_{qZ}$ and $\mathbf{e}_{r\varepsilon}$, respectively. In this section, we study behaviors for a more general class of weight vectors that we call *simple*. Intuitively, a simple weight vector does not distinguish between configurations with the same head:

Definition 3.14 ***(Simple weight vector)*** Let $\Delta = (Q, \Gamma, P)$ be a WPDA with weights in W.

A. A weight vector $\mathbf{f} \in W^{Q \times \Gamma^*}$ for Δ is called a *simple* [21] if

$$\forall qZ \in Q \times \Gamma\colon\ \forall \gamma, \eta \in \Gamma^*\colon\ \ \mathbf{f}(qZ\gamma) \;=\; \mathbf{f}(qZ\eta)\,.$$

B. For a simple $\mathbf{f} \in W^{Q \times \Gamma^*}$, we define $\mathbf{f}^+ \in W^{Q \times \Gamma^*}$ as[39]

$$\mathbf{f}^+(q\gamma) \;=\; \begin{cases} 0 & \text{if } \gamma = \varepsilon\,, \\ \mathbf{f}(q\gamma) & \text{if } \gamma \neq \varepsilon\,. \end{cases} \qquad \triangle$$

[21]: Brázdil et al. (2013), "Analyzing probabilistic pushdown automata".

39: Notice that $\mathbf{f}^+$ is also simple.

A simple weight vector $\mathbf{f}$ for a finite WPDA can be represented in a finite manner by specifying $\mathbf{f}(qZ)$ and $\mathbf{f}(r\varepsilon)$ for all heads qZ and all states r. Basic examples of simple weight vectors include $\mathbf{e}_{qZ}$ and $\mathbf{e}_{r\varepsilon}$ (the indicator vectors used to define return triples[40]), as well as constant vectors. Notice that simple vectors are closed under componentwise operations such as addition and multiplication.

40: See Definition 3.8 on page 56.

In the remainder, we focus on WPDA with simple *final* weight vectors, which suffices for our purposes. The following theorem characterizes the resulting behaviors:

Theorem 3.15 (WPDA behavior for simple final weights) *Let $\Delta = (Q, \Gamma, P)$ be a WPDA and let $\mathbf{f}$ be a simple*[41] *final weight vector for Δ.*

41: See Definition 3.14.

A. *The behaviors*

$$\langle qZ \rangle \;\overset{\text{def}}{=}\; [\![\mathbf{e}_{qZ}, \Delta, \mathbf{f}^+]\!]\,, \quad qZ \in Q \times \Gamma\,,$$

constitute the least solution[42] *of the following linear equation system:*[43]

$$\begin{aligned} \langle qZ \rangle \;=\; & \mathbf{f}(qZ) \\ & + \sum_{qZ \xrightarrow{w} rY} w \cdot \langle rY \rangle \\ & + \sum_{qZ \xrightarrow{w} rYX} w \cdot \Big(\langle rY \rangle + \sum_{s \in Q} [rY\,s] \cdot \langle sX \rangle \Big) \end{aligned}$$

42: Assuming we treat the symbols $\langle qZ \rangle$ as variables similar to Theorem 3.11.

43: Note that the return triples $[qZ\,r]$ (Definition 3.8 on page 56) appear as coefficients in the equations.

B. *For every $qZ \in Q \times \Gamma$, we have*

$$[\![\mathbf{e}_{qZ}, \Delta, \mathbf{f}]\!] \quad = \quad \langle qZ \rangle \; + \; \sum_{r \in Q} [qZ\, r] \cdot \mathbf{f}(r\varepsilon) \,.$$

Additional notation: We may write the equation system from Item A in matrix-vector notation as follows:

$$\mathbf{h} \;=\; A_\Delta(\mathbf{r}) \cdot \mathbf{h} + \mathbf{f}\big|_{Q\times\Gamma} \,.$$

Here, $\mathbf{h}$ is a (column) vector containing the symbols $\langle qZ \rangle$, one for each head $qZ \in Q \times \Gamma$, $A_\Delta(\mathbf{r})$ is a square matrix indexed by $Q \times \Gamma$ whose coefficients depend on the return triples $\mathbf{r}$, and $\mathbf{f}\big|_{Q\times\Gamma}$ is the restriction of $\mathbf{f}$ to $Q \times \Gamma$.

Proof. The main idea for Item A is to construct a modified WPDA $\tilde{\Delta}$ with a fresh state $\bot \notin Q$ such that

$$[\![\mathbf{e}_{qZ}, \Delta, \mathbf{f}^+]\!] \;=\; [qZ\,\bot] \,,$$

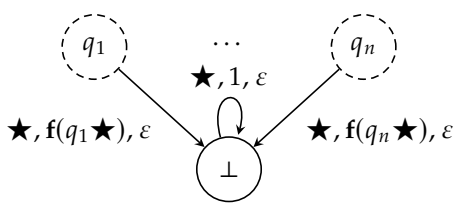
and then characterize the behavior by means of the fundamental equation system[44] of $\tilde{\Delta}$.

44: See Theorem 3.11 on page 58.

We construct $\tilde{\Delta}$ from Δ as follows. For every state $q \in Q$ and every stack symbol $Z \in \Gamma$, we add the pop transition

$$qZ \xrightarrow{\mathbf{f}(qZ)}_{\tilde{\Delta}} \bot\varepsilon \,.$$

The new state $\bot$ itself has a pop-transition $\bot Z \xrightarrow{1} \bot\varepsilon$ for every $Z \in \Gamma$; as a consequence, the stack can be emptied completely at $\bot$ without altering the weight of the path leading to $\bot$. Assuming that $Q = \{q_1, \ldots, q_n\}$, the overall construction can be visualized as follows ($\bigstar$ ranges over all stack symbols):



Next, consider the fundamental equation system of $\tilde{\Delta}$. It contains all equations from Δ as well as the following additional equation

for every $qZ \in Q \times \Gamma$:

$$\begin{aligned}
[qZ\,\bot] \;&=\; \mathbf{f}(qZ) && \text{(Pop)}\\
&+ \sum_{qZ \xrightarrow{w}_\Delta rY} w \cdot [rY\,\bot] && \text{(Swap)}\\
&+ \sum_{qZ \xrightarrow{w} rYX} w \cdot \Big(\sum_{s \in Q} [rY\,\bot] \cdot [\bot X\,\bot] + [rY\,s] \cdot [sX\,\bot] \Big)\,. && \text{(Push)}
\end{aligned}$$

Moreover, $[\bot Z\,\bot] = 1$ for every $Z \in \Gamma$. Simplifying and identifying the triples of the form $[qZ\,\bot]$ with $\langle qZ \rangle$ yields the linear equation system stated in Item A.

For Item B let

$$\mathbf{f}^{\varepsilon} \;=\; \sum_{r \in Q} \mathbf{e}_{r\varepsilon} \cdot \mathbf{f}(r\varepsilon)$$

be the weight vector that coincides with $\mathbf{f}$ on the terminal configurations $Q \times \{\varepsilon\}$ and is zero elsewhere. Notice that $\mathbf{f} = \mathbf{f}^{+} + \mathbf{f}^{\varepsilon}$, and therefore

$$\begin{aligned}
&[\![\mathbf{e}_{qZ}\,, \Delta\,, \mathbf{f}]\!]\\
=\;& [\![\mathbf{e}_{qZ}\,, \Delta\,, \mathbf{f}^{+} + \mathbf{f}^{\varepsilon}]\!]\\
=\;& [\![\mathbf{e}_{qZ}\,, \Delta\,, \mathbf{f}^{+}]\!] + [\![\mathbf{e}_{qZ}\,, \Delta\,, \mathbf{f}^{\varepsilon}]\!] && \text{(Linearity of behavior)}\\
=\;& \langle qZ \rangle + \sum_{r \in Q} [\![\mathbf{e}_{qZ}\,, \Delta\,, \mathbf{e}_{r\varepsilon}]\!] \cdot \mathbf{f}(r\varepsilon)\\
& && \text{(Definitions of } \langle qZ \rangle \text{ and } \mathbf{f}^{\varepsilon}\text{, linearity of behavior)}\\
=\;& \langle qZ \rangle + \sum_{r \in Q} [qZ\,r] \cdot \mathbf{f}(r\varepsilon)\,. && \text{(Definition of } [qZ\,r]\text{)}
\end{aligned}$$

This completes the proof of Theorem 3.15. □

► **Example 3.16** Reconsider the random walk pPDA from Example 3.4 on page 53 (restated in the margin). Consider the constant—and thus simple—final weight vector $\mathbf{f} = \mathbf{1}$. The equation system[45] from Theorem 3.15 is

$$\langle qZ \rangle \;=\; 1 + \frac{1}{2}\left(\langle qZ \rangle + [qZ\,q] \cdot \langle qZ \rangle \right)\,.$$

Using that $[qZ\,q] = 1$,[46] the equation can be simplified to

$$\langle qZ \rangle \;=\; 1 + \langle qZ \rangle\,.$$

This linear equation admits the unique solution $\langle qZ \rangle = \infty$ in $\overline{\mathbb{R}}_{\geq 0}$. It follows that $[\![\mathbf{e}_{qZ}\,, \Delta\,, \mathbf{1}^{+}]\!] = \infty$, which means that the *expected runtime* of the pPDA

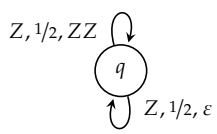


Figure 3.6: Example 3.4 restated.

45: The system consists of a single equation in this case.
46: As established in Example 3.13 on page 59.

is infinite from the initial configuration qZ. See Section 3.2.2 for more details on expected runtimes in pPDA. ◀

As a final result of this section, we present the following generalization of Theorem 3.15B:

Corollary 3.17 *Let* $\mathbf{f}$ *be a simple final weight vector*[47] *for a WPDA* $\Delta = (Q, \Gamma, P)$*. Then it holds for all* $qZ \in Q \times \Gamma$ *and* $\gamma \in \Gamma^*$ *that*

$$[\![\mathbf{e}_{qZ\gamma}, \Delta, \mathbf{f}]\!] = [\![\mathbf{e}_{qZ}, \Delta, \mathbf{f}^+]\!] + \sum_{r \in Q} [qZ\, r] \cdot [\![\mathbf{e}_{r\gamma}, \Delta, \mathbf{f}]\!] .$$

47: See Definition 3.14 on page 60.

The proof of Corollary 3.17 is similar to the one of Theorem 3.15B and is therefore omitted. Corollary 3.17 allows determining the behavior for an *arbitrary* initial configuration in a recursive fashion.

3.1.3 Behavioral Moments and Generating Functions

In Section 3.1.2, we presented equation systems for two different kinds of WPDA behaviors: return triples and behaviors for simple final weights. In this section, we study analogous equation systems for the corresponding (first) *behavioral moments*.[48] Recall that the latter are a variation of the standard behavior which additionally weighs each path with its *length*.

48: See Definition 3.7 on page 55. We do not consider higher moments here.

We will derive the desired equation systems with the help of FPS (formal power series).[49] Our approach is inspired by the *method of generating functions*, a prominent technique in combinatorics (e.g., [128]). For us, a generating function is a traditional FPS in a single indeterminate z, with coefficients in some semiring.

49: See Section 2.1.3 on page 20.

[128]: Wilf (2005), *generatingfunctionology*.

Moment Equations via Generating Functions Let us first illustrate our general approach for the simpler case of non-pushdown weighted automata. Consider an $\mathcal{S}$-weighted automaton $\mathfrak{A} = (S, M)$ with initial and final weight vectors $\mathbf{g}$ and $\mathbf{f}$, respectively.[50]

50: See Section 2.2 on page 27.

The crucial idea for characterizing the first behavioral moment $[\![\mathbf{g}, \mathfrak{A}, \mathbf{f}]\!]^{\langle 1 \rangle}$ is to multiply the weight of each transition of $\mathfrak{A}$ by a formal indeterminate z. This results in an automaton over the FPS semiring $\mathcal{S}[[z]]$, which we denote with $z\mathfrak{A}$. By definition, the behavior of $z\mathfrak{A}$ is

$$[\![\mathbf{g}, z\mathfrak{A}, \mathbf{f}]\!] = \sum_{k \geq 0} \mathbf{g}^\top z^k M^k \mathbf{f}$$

and its formal derivative[51] in z is therefore

$$\partial_z [\![\mathbf{g} , z\mathfrak{A} , \mathbf{f}]\!] \;=\; \sum_{k \geq 1} \mathbf{g}^\top k z^{k-1} M^k \mathbf{f} \,. \tag{3.1}$$

Substituting 1 for z in Equation (3.1) yields the first moment:

$$\partial_z [\![\mathbf{g} , z\mathfrak{A} , \mathbf{f}]\!] \Big|_{z=1} \;=\; [\![\mathbf{g} , \mathfrak{A} , \mathbf{f}]\!]^{\langle 1 \rangle} \tag{3.2}$$

We summarize and extend these observations in the following lemma:

Lemma 3.18 *Let $\mathfrak{A} = (S, M)$ be an $\mathcal{S}$-weighted automaton with initial and final weight vectors $\mathbf{g}$ and $\mathbf{f}$, respectively. Define the $\mathcal{S}[[z]]$-weighted automaton $z\mathfrak{A} = (S, z \cdot M)$. Then the following equations hold:*

$$\begin{aligned} [\![\mathbf{g} , \mathfrak{A} , \mathbf{f}]\!] &= [\![\mathbf{g} , z\mathfrak{A} , \mathbf{f}]\!] \Big|_{z=1} \\ [\![\mathbf{g} , \mathfrak{A} , \mathbf{f}]\!]^{\langle 1 \rangle} &= \partial_z [\![\mathbf{g} , z\mathfrak{A} , \mathbf{f}]\!] \Big|_{z=1} \end{aligned}$$

The equations hold analogously for forward and backward moments (assuming substitution and differentiation is applied componentwise).[52]

Proof. The first equation is immediate from the definitions. The second equation follows from Equations (3.1) and (3.2), which can be readily extended to forward and backward moments as well. □

How does Lemma 3.18 help toward our goal of obtaining an equation system for the behavioral moments? To answer this question, we focus on the backward moments and recall[53] that $\boldsymbol{x} = \langle\!\langle z\mathfrak{A} , \mathbf{f}]\!]$ is the least solution of the linear equation system

$$\boldsymbol{x} \;=\; zM\boldsymbol{x} + \mathbf{f} \,.$$

Now, the key idea is to carry out the steps from Equations (3.1) and (3.2) *on the level of this equation system*. First, taking the derivative yields

$$\partial_z \boldsymbol{x} \;=\; M\boldsymbol{x} + zM \cdot \partial_z \boldsymbol{x} \,,$$

where we have used the sum and product rules.[54] Second, by substituting 1 for z we obtain

$$\partial_z \boldsymbol{x} \Big|_{z=1} \;=\; M\, \boldsymbol{x} \Big|_{z=1} + M \cdot \partial_z \boldsymbol{x} \Big|_{z=1} \,.$$

Finally, writing $\boldsymbol{y}$ instead of $\partial_z \boldsymbol{x}|_{z=1}$ and using that $\boldsymbol{x}|_{z=1} = \langle\!\langle \mathfrak{A} , \mathbf{f}]\!]$

51: See Definition 2.14 on page 23.

52: The equations can moreover be generalized to higher order behavioral moments via repeated differentiation.

53: See Proposition 2.31 on page 32.

54: See Proposition 2.17 on page 24.

by Lemma 3.18, we obtain the linear equation system

$$\boldsymbol{y} \;=\; M \cdot \langle\!\langle \mathfrak{A}, \mathbf{f} \rangle\!\rangle + M\boldsymbol{y} \,.$$

Since derivative and substitution are monotone functions with respect to the natural FPS order, we may conclude that the least solution is indeed $\boldsymbol{y} = \langle\!\langle \mathfrak{A}, \mathbf{f} \rangle\!\rangle^{\langle 1 \rangle}$, as desired. Notice that the equation system for the *first* moment depends on the *zeroth* moment, that is, the standard behavior $\langle\!\langle \mathfrak{A}, \mathbf{f} \rangle\!\rangle$.

Equation System for Return Moment Triples We now apply the method outlined in the preceding paragraphs to pushdown automata. We start with the behavioral moments corresponding to the return triples[55] of a WPDA. We shall call these behavioral moments *return moment triples*.

55: See Definition 3.8 on page 56.

We state the resulting equation system in Theorem 3.19. Its generating function-based proof is given directly below. To understand to theorem, recall that the *Jacobi matrix* $\boldsymbol{f}'$ of a vector of multivariate polynomials[56] $\boldsymbol{f}$ (with coefficients in some semiring) has the partial derivatives of the polynomials in $\boldsymbol{f}$ as coefficients. More precisely, if the vector $\boldsymbol{f}$ is indexed by I and the set of variables is X, then $\boldsymbol{f}'$ is a $I \times X$-matrix whose coefficient $\boldsymbol{f}'_{i,x}$ is the partial derivative of $\boldsymbol{f}_i$ in x. For example, in $\overline{\mathcal{R}}_{\geq 0}$, the Jacobi matrix of

56: A *multivariate* polynomials is one that may contain more than one variable.

$$\boldsymbol{f} \;=\; \begin{pmatrix} x + y \\ y^2 + 1 \end{pmatrix} \qquad \text{is} \qquad \boldsymbol{f}' \;=\; \begin{pmatrix} 1 & 1 \\ 0 & 2y \end{pmatrix},$$

where the first column contains the derivatives in x and the second contains the derivatives in y. We revisit Jacobi matrices in Chapter 4.

Theorem 3.19 (Equation system for the return moment triples) *Let $\Delta = (Q, \Gamma, P)$ be a WPDA over a semiring $\mathcal{S}$.*

A. *The return moment triples*

$$[qZ\,r]^{\langle 1 \rangle} \quad \stackrel{\text{def}}{=} \quad [\![\mathbf{e}_{qZ}, \Delta, \mathbf{e}_{r\varepsilon}]\!]^{\langle 1 \rangle} \quad (qZ \in Q \times \Gamma, r \in Q)$$

constitute the least solution of the following linear equation system:[57]

$$[qZ\,t]^{\langle 1\rangle} \;=\; [qZ\,t] \;+\; \sum_{qZ \xrightarrow{w} rY} w \cdot [rY\,t]^{\langle 1\rangle} \;+\; \sum_{qZ \xrightarrow{w} rYX} w \cdot \sum_{s\in Q} \Big([rY\,s]^{\langle 1\rangle} \cdot [sX\,t] \;+\; [rY\,s] \cdot [sX\,t]^{\langle 1\rangle} \Big)$$

57: This equation system contains the standard return triples $[qZ\,r]$ as coefficients. Recall that these triples are themselves the least solution of the *nonlinear* fundamental system from Theorem 3.11 on page 58:

$$[qZ\,t] \;=\; \sum_{qZ \xrightarrow{w} t} w \;+\; \sum_{qZ \xrightarrow{w} rY} w[rY\,t] \;+\; \sum_{qZ \xrightarrow{w} rYX} w \sum_{s\in Q} [rY\,s][sX\,t]$$

B. *Let* $\mathbf{r} = f_\Delta(\mathbf{r})$ *be the fundamental system of* Δ *in vector notation.Then the equation system from Item A can be written as*

$$\mathbf{r}^{\langle 1\rangle} \;=\; \mathbf{r} + f'_\Delta(\mathbf{r}) \cdot \mathbf{r}^{\langle 1\rangle}\,,$$

where $f'_\Delta(\mathbf{r})$ *is the Jacobi matrix of* f_Δ *evaluated at* $\mathbf{r}$.

Proof. We first derive the equation system in matrix-vector notation as stated in Item B. Let $\mathbf{r} = f_\Delta(\mathbf{r})$ be the fundamental equation system of Δ.

We consider the automaton $z\Delta = (Q, \Gamma, z \cdot P)$ over the semiring $\mathcal{S}[[z]]$ of generating functions. Note that $z\Delta$ is like Δ with each transition weight multiplied by the formal variable z. We apply the reasoning from Equations (3.1) and (3.2) on page 64 to the fundamental equation system of $z\Delta$, which can be written as

$$\mathbf{r}_z \;=\; f_{z\Delta}(\mathbf{r}_z) \;=\; z \cdot f_\Delta(\mathbf{r}_z)\,.$$

Taking derivatives on both sides—analogous to Equation (3.1)—we find that

$$\partial_z \mathbf{r}_z \;=\; f_\Delta(\mathbf{r}_z) \;+\; z \cdot f'_\Delta(\mathbf{r}_z) \cdot \partial_z \mathbf{r}_z\,,$$

where we have applied the product and the chain rule (the latter introduces the Jacobi matrix). The next step is to substitute 1 for z as in Equation (3.2):

$$\partial_z \mathbf{r}_z \,\big|_{z=1} \;=\; f_\Delta\big(\mathbf{r}_z \,|_{z=1}\big) \;+\; f'_\Delta\big(\mathbf{r}_z \,|_{z=1}\big) \cdot \partial_z \mathbf{r}_z \,\big|_{z=1}\,.$$

By Lemma 3.18, $\partial_z \mathbf{r}_z \,|_{z=1}$ is the vector of return moment triples, that is, a vector containing the triples $[qZ\,r]^{\langle 1\rangle}$ for every head $qZ \in Q \times \Gamma$ and state $r \in Q$. Let us denote this vector with $\mathbf{r}^{\langle 1\rangle}$. By Lemma 3.18, we also have $\mathbf{r}_z \,|_{z=1} = \mathbf{r}$. Our equation system thus becomes:

$$\mathbf{r}^{\langle 1\rangle} \;=\; f_\Delta(\mathbf{r}) \;+\; f'_\Delta(\mathbf{r}) \cdot \mathbf{r}^{\langle 1\rangle}\,.$$

Finally, using that $f_\Delta(\mathbf{r}) = \mathbf{r}$ yields the desired equation.

The equations stated in Item A can be derived from the matrix-vector formulation Item B that we have just proved. Alternatively, we can also apply the *differentiate-and-substitute* trick[58] directly to the fundamental equations of $z\Delta$ in explicit form. See [135, Lemma 7] for details on the latter approach. □

58: See Equations (3.1) and (3.2) on page 64.

[135]: Winkler et al. (2023), "On Certificates, Expected Runtimes, and Termination in Probabilistic Pushdown Automata".

The fact that the Jacobi matrix $\boldsymbol{f}'_\Delta$ surfaces in the equation system for the return moment triples of Δ from Theorem 3.19B is remarkable. Indeed, we can view this system as the *linearization* (first-order Taylor approximation) of the fundamental equation system $\boldsymbol{f}_\Delta$ around the point $\mathbf{r}$. We will exploit this in Chapter 4.

Behavioral Moments for Simple Final Weights Theorem 3.20 below is the moment version of Theorem 3.15 on page 60. Its proof relies on the same generating function trick as the proof of Theorem 3.19.

Theorem 3.20 (Behavioral moments for simple final weights) *Let $\Delta = (Q, \Gamma, P)$ be a WPDA with simple[59] final weight vector $\mathbf{f}$. For every head $qZ \in Q \times \Gamma$ define the symbols[60]*

$$\langle qZ \rangle = [\![\mathbf{e}_{qZ}, \Delta, \mathbf{f}^+]\!] \quad \text{and} \quad \langle qZ \rangle^{\langle 1 \rangle} = [\![\mathbf{e}_{qZ}, \Delta, \mathbf{f}^+]\!]^{\langle 1 \rangle} .$$

59: See Definition 3.14 on page 60.

60: $\langle qZ \rangle$ is as in Theorem 3.15.

A. *The behavioral moments $\langle qZ \rangle^{\langle 1 \rangle}$ are the least solution of the following linear equation system:*[61]

$$\begin{aligned}
\langle qZ \rangle^{\langle 1 \rangle} &= \sum_{qZ \xrightarrow{w} rY} w \cdot \langle rY \rangle \\
&+ \sum_{qZ \xrightarrow{w} rYX} w \Big(\langle rY \rangle + \sum_{s \in Q} \big([rY\,s] + [rY\,s]^{\langle 1 \rangle} \big) \langle sX \rangle \Big) \\
&+ \sum_{qZ \xrightarrow{w} rY} w \cdot \langle rY \rangle^{\langle 1 \rangle} \\
&+ \sum_{qZ \xrightarrow{w} rYX} w \Big(\langle rY \rangle^{\langle 1 \rangle} + \sum_{s \in Q} [rY\,s] \cdot \langle sX \rangle^{\langle 1 \rangle} \Big)
\end{aligned}$$

61: The system contains the return triples as well as the return moment triples as coefficients; see Definition 3.8 and theorem 3.19 on page 56 and on page 65.

B. *For every $qZ \in Q \times \Gamma$, we have*

$$[\![\mathbf{e}_{qZ}, \Delta, \mathbf{f}]\!]^{\langle 1 \rangle} = \langle qZ \rangle^{\langle 1 \rangle} + \sum_{r \in Q} [qZ\,r]^{\langle 1 \rangle} \cdot \mathbf{f}(r\varepsilon) .$$

C. *Let $\mathbf{h} = A_\Delta(\mathbf{r}) \cdot \mathbf{h} + \mathbf{f}|_{Q \times \Gamma}$ be the linear equation system for the*

behaviors $\langle qZ \rangle$ in matrix-vector notation.[62] *Then the equation system from Item A can be written as:*

$$\mathbf{h}^{\langle 1 \rangle} \;=\; A_\Delta(\mathbf{r}) \cdot \mathbf{h}^{\langle 1 \rangle} + A_\Delta(\mathbf{r} + \mathbf{r}^{\langle 1 \rangle}) \cdot \mathbf{h} \;.$$

Here, $\mathbf{h}^{\langle 1 \rangle}$ is a vector containing the moments $\langle qZ \rangle^{\langle 1 \rangle}$ for all $qZ \in Q \times \Gamma$, and $\mathbf{r}^{\langle 1 \rangle}$ is the vector of return moment triples[63] *of Δ.*

62: See Theorem 3.15 on page 60.

63: Theorem 3.19 on page 65

Proof. We start with Item C. The proof parallels the one of Theorem 3.19B. Consider the equation system for the behaviors $\mathbf{h}_z$ of the automaton $z\Delta$, where z is a formal indeterminate:

$$\mathbf{h}_z \;=\; z \cdot A_\Delta(\mathbf{r}_z) \cdot \mathbf{h}_z + \mathbf{f}\,\big|_{Q \times \Gamma} \;.$$

In the following, we write $A(\cdot)$ instead of $A_\Delta(\cdot)$ to enhance readability. Taking componentwise derivatives on both sides yields:

$$\partial_z \mathbf{h}_z \;=\; A(\mathbf{r}_z) \cdot \mathbf{h}_z + z \cdot \partial_z(A(\mathbf{r}_z)) \cdot \mathbf{h}_z + z \cdot A(\mathbf{r}_z) \cdot \partial_z \mathbf{h}_z \;.$$

Here, $\partial_z(A(\mathbf{r}_z))$ is the coefficientwise derivative of the matrix $A(\mathbf{r}_z)$ in z. By the chain rule, we have $\partial_z(A(\mathbf{r}_z)) = \partial_z A(\mathbf{z}) \cdot \partial_z \mathbf{r}_z$. Notice that $\partial_z A(\mathbf{z})$ does not depend on z anymore because z only occurs linearly in the coefficients of $A(\mathbf{z})$. Substituting 1 for z, we thus obtain:

$$\begin{aligned} \partial_z \mathbf{h}_z\,\big|_{z=1} \;=\; & A(\mathbf{r}_z\,\big|_{z=1}) \cdot \mathbf{h}_z\,\big|_{z=1} \\ & + \partial_z A(\mathbf{z}) \cdot \partial_z \mathbf{r}_z\,\big|_{z=1} \cdot \mathbf{h}_z\,\big|_{z=1} \\ & + A(\mathbf{r}_z\,\big|_{z=1}) \cdot \partial_z \mathbf{h}_z\,\big|_{z=1} \;. \end{aligned}$$

Using that $\partial_z \mathbf{h}_z\,|_{z=1} = \mathbf{h}^{\langle 1 \rangle}$, $\partial_z \mathbf{r}_z\,|_{z=1} = \mathbf{r}^{\langle 1 \rangle}$, $\mathbf{h}_z\,|_{z=1} = \mathbf{h}$, as well as $\mathbf{r}_z\,|_{z=1} = \mathbf{r}$ by Lemma 3.18 on page 64, the previous equation becomes:

$$\begin{aligned} \mathbf{h}^{\langle 1 \rangle} \;&=\; A(\mathbf{r}) \cdot \mathbf{h} \;+\; \partial_z A(\mathbf{z}) \cdot \mathbf{r}^{\langle 1 \rangle} \cdot \mathbf{h} \;+\; A(\mathbf{r}) \cdot \mathbf{h}^{\langle 1 \rangle} \\ &=\; \Big(A(\mathbf{r}) + \partial_z A(\mathbf{z}) \cdot \mathbf{r}^{\langle 1 \rangle}\Big) \cdot \mathbf{h} \;+\; A(\mathbf{r}) \cdot \mathbf{h}^{\langle 1 \rangle} \;. \end{aligned}$$

The last step is to notice that $A(\mathbf{r}) + \partial_z A(\mathbf{z}) \cdot \mathbf{r}^{\langle 1 \rangle} = A(\mathbf{r} + \mathbf{r}^{\langle 1 \rangle})$. This equation follows by taking the first-order Taylor approximation of $A(\cdot)$ at $\mathbf{r}$ and using that this approximation is in fact exact because the dependency of $A(\cdot)$ on its argument is linear.

Item A follows from the matrix-vector formulation of Item C. This is best seen by comparing the latter with the explicit equations from Theorem 3.15A.

Item B is straightforward and analogous to Theorem 3.15B. □

▷ **Remark 3.21** (Alternative proofs with the moment semiring) An alternative approach for deriving the equation systems from Theorems 3.19 and 3.20 is via the moment semiring.[64] Indeed, notice that the terms

$$[rY\,s]^{\langle 1\rangle} \cdot [s X\,t] \;+\; [rY\,s] \cdot [s X\,t]^{\langle 1\rangle}$$

occurring in the return moment triples system from Theorem 3.19 correspond to multiplication in the moment semiring.[65]

The presentation based on generating functions has the advantage that it makes the connection between moments and differentiation explicit, namely via the *differentiate-and-substitute* technique.[66] It thus provides an intuitive explanation for the occurrence of the Jacobi matrix in the equation system $\mathbf{r}^{\langle 1\rangle} = \mathbf{r} + \boldsymbol{f}'_{\Delta}(\mathbf{r}) \cdot \mathbf{r}^{\langle 1\rangle}$ of return moment triples. . . . . ◁

Figure 3.7 on the next page provides an overview of the various equation systems presented in Sections 3.1.2 and 3.1.3.

64: The moment semiring was introduced in Remark 2.37 on page 34.

65: Recall that multiplication in the moment semiring is defined as

$$\begin{aligned}&(w_1, v_1) \cdot (w_2, v_2)\\ =\;&(w_1 w_2,\, w_1 v_2 + v_1 w_2)\,.\end{aligned}$$

66: See Equations (3.1) and (3.2) on page 64..

Description	**Type**	**Dimension**	**Vector notation**
1. *Return triples (fundamental system)* [Theorem 3.11]	nonlinear	$\lvert Q\rvert^2 \cdot \lvert\Gamma\rvert$	$\mathbf{r} = f_\Delta(\mathbf{r})$
$[qZ\,t] = \sum_{qZ \xrightarrow{w} t} w + \sum_{qZ \xrightarrow{w} rY} w \cdot [rY\,t] + \sum_{qZ \xrightarrow{w} rYX} w \cdot \sum_{s\in Q} [rY\,s] \cdot [sX\,t]$			
2. *Behaviors for simple final weights* $\mathbf{f}$ [Theorem 3.15]	linear	$\lvert Q\rvert \cdot \lvert\Gamma\rvert$	$\mathbf{h} = A_\Delta(\mathbf{r})\mathbf{h} + \mathbf{f}\vert_{Q\times\Gamma}$
$\langle qZ\rangle = \mathbf{f}(qZ) + \sum_{qZ \xrightarrow{w} rY} w \cdot \langle rY\rangle + \sum_{qZ \xrightarrow{w} rYX} w \cdot \Big(\langle rY\rangle + \sum_{s\in Q} [rY\,s] \cdot \langle sX\rangle\Big)$			
3. *Return moment triples* [Theorem 3.19]	linear	$\lvert Q\rvert^2 \cdot \lvert\Gamma\rvert$	$\mathbf{r}^{\langle 1\rangle} = \mathbf{r} + f'_\Delta(\mathbf{r}) \cdot \mathbf{r}^{\langle 1\rangle}$
$[qZ\,t]^{\langle 1\rangle} = [qZ\,t] + \sum_{qZ \xrightarrow{w} rY} w \cdot [rY\,t]^{\langle 1\rangle} + \sum_{qZ \xrightarrow{w} rYX} w \cdot \sum_{s\in Q} \Big([rY\,s]^{\langle 1\rangle} \cdot [sX\,t] + [rY\,s] \cdot [sX\,t]^{\langle 1\rangle}\Big)$			
4. *Moments for simple final weights* $\mathbf{f}$ [Theorem 3.19]	linear	$\lvert Q\rvert \cdot \lvert\Gamma\rvert$	$\mathbf{h}^{\langle 1\rangle} = A_\Delta(\mathbf{r})\mathbf{h}^{\langle 1\rangle} + A_\Delta(\mathbf{r} + \mathbf{r}^{\langle 1\rangle})\mathbf{h}$
$\langle qZ\rangle^{\langle 1\rangle} = \sum_{qZ \xrightarrow{w} rY} w \cdot \langle rY\rangle + \sum_{qZ \xrightarrow{w} rYX} w\Big(\langle rY\rangle + \sum_{s\in Q} \big([rY\,s] + [rY\,s]^{\langle 1\rangle}\big)\langle sX\rangle\Big) + \sum_{qZ \xrightarrow{w} rY} w \cdot \langle rY\rangle^{\langle 1\rangle} + \sum_{qZ \xrightarrow{w} rYX} w\Big(\langle rY\rangle^{\langle 1\rangle} + \sum_{s\in Q} [rY\,s] \cdot \langle sX\rangle^{\langle 1\rangle}\Big)$			

Figure 3.7: Overview of equation systems associated with a WPDA $\Delta = (Q, \Gamma, P)$. Notice that the linear systems 2–4 contain the solution of the nonlinear system 1 as coefficients.

3.2 Probabilistic Pushdown Automata

Probabilistic pushdown automtata are the *Markov chain version* of WPDA. We define them formally as follows:

Definition 3.22 (pPDA: *Probabilistic pushdown automaton)* A pPDA is an $\overline{\mathcal{R}}_{\geq 0}$-WPDA[67] $\Delta = (Q, \Gamma, P)$ with the following additional constraint:

$$\forall qZ \in Q \times \Gamma\colon \quad \sum_{qZ \xrightarrow{p} r\gamma} p \;\leq\; 1\,.$$

Δ is called *proper* if the above sum is *equal* to 1 for all qZ. △

67: See Definition 3.2 on page 51.

In words, a pPDA is a WPDA over the semiring $\overline{\mathcal{R}}_{\geq 0} = (\overline{\mathbb{R}}_{\geq 0}, +, \cdot, 0, 1)$ of extended nonnegative reals. Additionally, the sum of the weights of the available transitions at every head qZ is at most 1. This is equivalent to requiring the pushdown transition matrix P to be substochastic. We have already seen a pPDA in Example 3.4 on page 53, which is restated in the margin. Notice that this pPDA is proper.

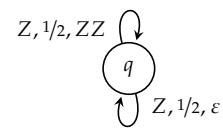


Figure 3.8: pPDA from Example 3.4 restated.

It is straightforward to verify that the induced automaton[68] of any pPDA is a DTMC.[69] Throughout this chapter, we treat pPDA primarily as compact representations of their infinite induced DTMCs. This means we focus on analyzing standard DTMC properties like termination probabilities and expected runtimes.[70] Later, in Chapter 6, we will study more sophisticated temporal properties tailored to the structured computations of pPDA.

68: See Definition 3.5 on page 54.

69: As illustrated in Example 3.6 on page 54.

70: These properties were defined in Section 2.3.

In the remainder of this section, we consider termination probabilities (Section 3.2.1), expected runtimes (Section 3.2.2), and conditional expected runtimes (Section 3.2.3) of pPDA.[71]

71: This parallels the structure of Section 2.3 in Chapter 2.

3.2.1 Termination Probabilities of pPDA

Below, we define the termination probability of a pPDA as that of its induced DTMC. Notably, induced DTMCs are never proper, that is, their transition matrix is strictly substochastic.[72] This occurs because a pPDA cannot perform any transitions from configurations with an empty stack. Such configurations have zero outgoing probability mass in the induced DTMC.

72: Not even if the pPDA is proper in the sense of Definition 3.22

Definition 3.23 ***(Termination probability of a pPDA)*** Let $\Delta = (Q, \Gamma, P)$ be a pPDA with an initial distribution $\mathbf{g}$ on its configurations $Q \times \Gamma^*$.

A. The *termination probability* of Δ from $\mathbf{g}$ is defined as the termination probability of the induced DTMC $\mathfrak{A}_\Delta$ from $\mathbf{g}$.[73] Concretely, the termination probability is the behavior

$$[\![\mathbf{g}, \Delta, \mathbf{t}]\!],$$

where $\mathbf{t}$ are the immediate termination probabilities of $\mathfrak{A}_\Delta$.

B. Δ is *AST* (almost surely terminating) from $\mathbf{g}$ (or $\mathbf{g}$-AST) if

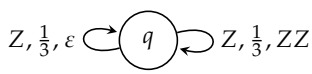
$$[\![\mathbf{g}, \Delta, \mathbf{t}]\!] = 1,$$

that is, the termination probability is one.

C. Δ is called *universally AST* if Δ is AST from *all* initial distributions on the set of configurations $Q \times \Gamma^*$. △

73: See Definition 2.40 on page 37.

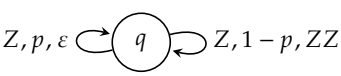
► **Example 3.24** (An almost surely terminating pPDA) Consider the following improper pPDA Δ with $Q = \{q\}$ and $\Gamma = \{Z\}$:



Intuitively, this pPDA is universally AST because in every step there is at least a $1:3$ chance of terminating immediately due the missing probability mass of $1/3$. We will establish universal AST formally in Example 3.27 on page 74. ◄

► **Example 3.25** (Non-almost surely terminating pPDA) There also exist pPDA that are not AST. Trivial examples include proper pPDA without any pop transitions; such pPDA have zero termination probability from any configuration with nonempty stack.

A more interesting class of non-AST examples arises from pPDA that mimic a random walk on the stack height, with a bias toward *increasing* the stack. More precisely, let Δ_p be the following pPDA where $p \leq 1/2$:



74: See Example 3.25.

It can be shown[74] that Δ_p terminates only with probability $\left(\frac{p}{1-p}\right)^n$ from configuration qZ^n, where $n \in \mathbb{N}$. For example, if $p = 1/3$, then the termination probability from qZ is $1/2$. ◄

Next, we observe that the immediate termination probabilities $\mathbf{t}$ of an induced DTMC constitute a *simple* final weight vector (i.e., immediate termination depends only on the head of a configuration). We can therefore apply Theorem 3.15 to obtain the following equation system characterization of the termination probabilities:

Corollary 3.26 (Equation system for termination probabilities) *Let $\Delta = (Q, \Gamma, P)$ be a pPDA and let $\mathbf{t}$ be the immediate termination probabilities*[75] *of the induced DTMC of Δ.*

75: See Definition 2.40 on page 37.

76: See Definition 3.14 on page 60.

A. *$\mathbf{t}$ is simple,*[76] *the restriction of $\mathbf{t}$ to the heads $Q \times \Gamma$ is $\mathbf{1} - P\mathbf{1}$, and the restriction of $\mathbf{t}$ to the terminal configurations $Q \times \{\varepsilon\}$ is $\mathbf{1}$.*

B. *The probabilities*

$$\langle qZ \rangle \quad \stackrel{\text{def}}{=} \quad [\![\mathbf{e}_{qZ}, \Delta, \mathbf{t}^+]\!] \qquad (qZ \in Q \times \Gamma)$$

of terminating before reaching the empty stack *constitute the least solution of the linear equation system*

$$\begin{aligned} \langle qZ \rangle \;=\; & \mathbf{t}(qZ) && \text{(Immediate termination at } qZ\text{)} \\ & + \sum_{qZ \xrightarrow{p} rY} p \cdot \langle rY \rangle && \text{(Swap)} \\ & + \sum_{qZ \xrightarrow{p} rYX} p \cdot \Big(\langle rY \rangle + \sum_{s \in Q} [rY\,s] \cdot \langle sX \rangle \Big) . && \text{(Push)} \end{aligned}$$

The (total) termination probability from head qZ is thus given as

$$[\![\mathbf{e}_{qZ}, \Delta, \mathbf{t}]\!] \;=\; \langle qZ \rangle \;+\; \sum_{r \in Q} [qZ\,r] .$$

C. *If Δ is proper, then $\mathbf{t}^+ = \mathbf{0}$ and the termination probability from qZ can be written more concisely as*

$$[\![\mathbf{e}_{qZ}, \Delta, \mathbf{t}]\!] \;=\; \sum_{r \in Q} [qZ\,r] .$$

D. *Δ is universally AST if and only if Δ is $\mathbf{e}_{qZ}$-AST for all heads $qZ \in Q \times \Gamma$.*

Proof. Item A follows immediately from the definition of the induced DTMC. Items B and C are instances of Theorem 3.15.

For direction $\Longleftarrow$ of Item D (the other direction is trivial) we assume that Δ is *not* universally AST. Then there exists a configuration $q\gamma \in Q \times \Gamma^*$ such that Δ is not $q\gamma$-AST, that is, $[\![\mathbf{e}_{q\gamma}, \Delta, \mathbf{t}]\!] < 1$.

Assume, without loss of generality, that γ has minimal length, that is, Δ is $r\eta$-AST for all $r \in Q$ and η with $|\eta| < |\gamma|$. Moreover, we have $|\gamma| > 0$ since Δ is AST from every terminal configuration; γ can thus be written as $\gamma = Z\eta$. To complete the proof, we argue as follows:

$$\begin{aligned}
[\![\mathbf{e}_{qZ\eta}, \Delta, \mathbf{t}]\!] &= \langle qZ\rangle + \sum_{r\in Q}[qZ\,r]\cdot[\![\mathbf{e}_{r\eta}, \Delta, \mathbf{t}]\!] && \\
&&& \text{(Corollary 3.17)} \\
&= \langle qZ\rangle + \sum_{r\in Q}[qZ\,r] && (\Delta \text{ is } r\eta\text{-AST}) \\
&= [\![\mathbf{e}_{qZ}, \Delta, \mathbf{t}]\!] && \text{(Item B)} \\
&< 1 && \text{(Assumption)}
\end{aligned}$$

Thus Δ is not AST from initial head qZ. □

Note that the linear equation system from Corollary 3.26B includes the return triples $[qZ\,r]$—where $qZ \in Q \times \Gamma$ and $r \in Q$—as coefficients. In the case of pPDA, these triples are actually *probabilities*, i.e., real numbers between zero and one. More precisely, $[qZ\,r]$ denotes the probability of reaching the configuration $r\varepsilon$ from qZ. Accordingly, we refer to the return triples of a pPDA as *return probability triples* or simply *return probabilities*.

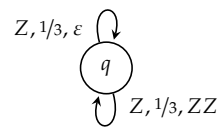


Figure 3.9: Improper pPDA from Example 3.24 restated.

► **Example 3.27** (Proving AST) Reconsider the pPDA from Example 3.24 (restated in the margin). We now prove AST formally by showing that the termination probability from initial distribution $\mathbf{e}_{qZ}$ is one. We proceed in three steps.

1. To determine the return probabilities of Δ we solve its fundamental equation system,[77] which consists of just a single equation since Q and Γ are both singletons:

   $$[qZ\,q] = \frac{1}{3} + \frac{1}{3}[qZ\,q]^2 .$$

   This quadratic equation has two real solutions,

   $$\frac{3-\sqrt{5}}{2} \approx 0.382 \quad \text{and} \quad \frac{3+\sqrt{5}}{2} \approx 2.618 .$$

   Following Theorem 3.11, the return probabilities are the *smallest* solution, that is, $[qZ\,q] = \frac{3-\sqrt{5}}{2}$.[78] Intuitively, the return probability $[qZ\,q]$ is the chance to reach the empty stack starting in configuration qZ.
2. Next, we compute $\langle qZ\rangle = [\![\mathbf{e}_{qZ}, \Delta, \mathbf{t}^+]\!]$ as in Corollary 3.26. Intuitively, $\langle qZ\rangle$ is the probability to terminate *without* reaching the empty stack.

77: See Theorem 3.11 on page 58.

78: In this particular example, the other solution is not a probability. However, there also exist examples with multiple solutions in $[0, 1]$.

We have

$$\begin{aligned}\langle qZ\rangle &= \frac{1}{3} + \frac{1}{3}\Big(\langle qZ\rangle + [qZ\,q]\cdot\langle qZ\rangle\Big)\\ &= \frac{1}{3} + \frac{1}{3}\Big(\langle qZ\rangle + \frac{3-\sqrt{5}}{2}\cdot\langle qZ\rangle\Big)\end{aligned}$$

with unique solution $\langle qZ\rangle = \frac{\sqrt{5}-1}{2}$.

3. Finally, the termination probability of Δ from initial configuration qZ is

$$[\![\mathbf{e}_{qZ}\,,\Delta\,,\mathbf{t}]\!] \;=\; \langle qZ\rangle + [qZ\,q] \;=\; \frac{\sqrt{5}-1}{2} + \frac{3-\sqrt{5}}{2} \;=\; 1\,,$$

as expected. By Corollary 3.26D, this suffices to conclude that Δ is even *universally* AST.

This example also shows that the return probabilities may be *irrational* numbers. ◀

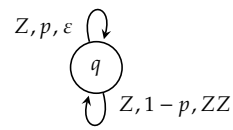


Figure 3.10: The pPDA Δ_p from Example 3.25 restated.

► **Example 3.28** (Computing the termination probabilities) Reconsider Example 3.25 on page 72 in which we claimed that, for $p \le 1/2$, the pPDA Δ_p (restated in the margin) terminates with probability

$$[\![\mathbf{e}_{qZ^n}\,,\Delta_p\,,\mathbf{t}]\!] \;=\; \left(\frac{p}{1-p}\right)^n$$

from the initial configuration qZ^n, where $n \in \mathbb{N}$. We now establish this claim formally. Since Δ_p is proper, the termination probability from head qZ is equal to the return probability triple $[qZ\,q]$; see Corollary 3.26C. The triple $[qZ\,q]$ is the least nonnegative solution of

$$[qZ\,q] \;=\; p + (1-p)\cdot[qZ\,q]^2\,.$$

It can be shown that this quadratic equation has the real solutions $\frac{p}{1-p}$ and 1 for all $p \neq 1$. For $p \le 1/2$ in particular, we have $\frac{p}{1-p} \le 1$, as expected. For initial configurations other than qZ, we use Corollary 3.17 on page 63 to derive that, for $n \in \mathbb{N}$,

$$[\![\mathbf{e}_{qZ^{n+1}}\,,\Delta_p\,,\mathbf{t}]\!] \;=\; [qZ\,q]\cdot[\![\mathbf{e}_{qZ^n}\,,\Delta_p\,,\mathbf{t}]\!]\,.$$

This implies the claim by induction, using the trivial fact that $[\![\mathbf{e}_{q\varepsilon}\,,\Delta_p\,,\mathbf{t}]\!] = 1$ for the base case $n = 0$. ◀

▷ **Remark 3.29** (Making a pPDA proper) Computing the termination probability of improper pPDA is slightly more involved than for proper pPDA.[79] Indeed, in the proper case, we only have to compute the return probability triples, whereas improper pPDA require solving an additional linear equation system. To simplify the handling of the improper case, one could make the pPDA proper by redirecting the missing probabilities (the immediate termination probabilities) to a fresh state with self-looping pop transitions for every symbol.[80] However, since this increases the size of the

79: See Corollary 3.26B *vs* Corollary 3.26C

80: This is an instance of the general construction from the proof of Theorem 3.15.

fundamental equation system and does not generalize to other properties (such as expected runtimes; see Section 3.2.2), we have opted to treat termination for improper pPDA explicitly. ◁

3.2.2 Expected Runtimes of pPDA

The expected runtime of a pPDA is the expected number of transitions taken until termination, which may be ∞. Recall that termination occurs either when the stack becomes empty or due to missing probability mass in improper pPDA, causing immediate termination. As with termination probabilities, we formally define the expected runtime of a pPDA as the expected runtime of its induced DTMC:

Definition 3.30 ***(Expected runtimes of pPDA)*** Let $\Delta = (Q, \Gamma, P)$ be a pPDA with an initial distribution $\mathbf{g}$ on its configurations $Q \times \Gamma^*$.

A. The *expected runtime* of Δ from $\mathbf{g}$ is defined as the expected runtime[81] of the induced DTMC $\mathfrak{M}_\Delta$ from $\mathbf{g}$. Concretely, the expected runtime is the behavior

$$[\![\mathbf{g}, \Delta, \mathbf{1} - \mathbf{t}]\!],$$

where $\mathbf{t}$ are the immediate termination probabilities[82] of $\mathfrak{M}_\Delta$.

B. Δ is called *positively almost surely terminating* (PAST) from $\mathbf{g}$ (or $\mathbf{g}$-PAST, for short) if

$$[\![\mathbf{g}, \Delta, \mathbf{1} - \mathbf{t}]\!] < \infty,$$

that is, the expected runtime of Δ from $\mathbf{g}$ is finite.

C. Δ is called *universally PAST* if Δ is PAST from *all* initial distributions over $Q \times \Gamma^*$. △

81: See Definition 2.45 on page 39.

82: See Definition 2.45 on page 39.

Recall from our termination taxonomy[83] that PAST implies AST. Therefore, every pPDA that is *not* AST from some initial distribution—such as the one from Example 3.25 on page 72—has infinite expected runtime from that distribution.

83: See Theorem 2.54 on page 45.

The next result is an equation system characterization for the expected runtime of a pPDA. It is yet another application of simple final weight vectors:

Corollary 3.31 (Equation system for expected runtimes) *Let* $\Delta = (Q, \Gamma, P)$ *be a pPDA and let* $\mathbf{t}$ *be the immediate termination probabilities*[84] *of the induced DTMC of* Δ.

84: See Definition 2.45 on page 39.

A. *The vector* $\mathbf{1} - \mathbf{t}$ *is simple,*[85] *and we have* $(\mathbf{1} - \mathbf{t})^+ = \mathbf{1} - \mathbf{t}$.
B. *The expected runtimes*

$$\langle qZ \rangle \stackrel{\text{def}}{=} [\![\mathbf{e}_{qZ}, \Delta, \mathbf{1} - \mathbf{t}]\!], \qquad qZ \in Q \times \Gamma,$$

constitute the least solution in $\overline{\mathbb{R}}_{\geq 0}$ *of the linear equation system*

$$\begin{aligned} \langle qZ \rangle = & \sum_{qZ \xrightarrow{p} r\gamma} p && \text{(Probability to continue)} \\ & + \sum_{qZ \xrightarrow{p} rY} p \cdot \langle rY \rangle && \text{(Swap)} \\ & + \sum_{qZ \xrightarrow{p} rYX} p \cdot \Big(\langle rY \rangle + \sum_{s \in Q} [rYs] \cdot \langle sX \rangle \Big). && \text{(Push)} \end{aligned}$$

C. *If* Δ *is proper,*[86] *the linear equation system from Item* B *simplifies to:*

$$\langle qZ \rangle = 1 + \sum_{qZ \xrightarrow{p} rY} p \cdot \langle rY \rangle + \text{... (push case as above)}$$

D. Δ *is universally PAST if and only if* Δ *is* $\mathbf{e}_{qZ}$*-PAST for all heads* $qZ \in Q \times \Gamma$.

85: See Definition 3.14 on page 60.

86: See Definition 3.22 on page 71.

Proof. $\mathbf{1} - \mathbf{t}$ is simple because $\mathbf{t}$ is simple by Corollary 3.26A and simple weight vectors are closed under componentwise operations. Moreover, $\mathbf{t}(r\varepsilon) = 1$ for all $r \in Q$ by definition of the induced DTMC, hence $(\mathbf{1} - \mathbf{t})^+ = \mathbf{1} - \mathbf{t}$.

Item B is an instance of Theorem 3.15A, using that $(\mathbf{1} - \mathbf{t})^+ = \mathbf{1} - \mathbf{t}$.

Item C is an immediate consequence of Item B and the definition of properness.

The proof of Item D is analogous to that of Corollary 3.26D and is thus omitted. □

We can also write the expected runtime equation system from Corollary 3.31B in matrix-vector notation as follows:

$$\mathbf{h} = P\mathbf{1} + A_\Delta(\mathbf{r}) \cdot \mathbf{h}.$$

Here, the matrix $A_\Delta(\mathbf{r})$ is as in Theorem 3.15; in particular, its coefficients depend on the return probability triples $\mathbf{r}$. Also notice that $P\mathbf{1} = \mathbf{1}$ for proper pPDA.

An important consequence of Corollary 3.31 is that Δ is universally PAST if *and only if* the equation system has a nonnegative *finite* solution. In Chapter 4, we exploit this fact to derive sound and *complete* certificates for PAST. These certificates are in essence easy-to-check proofs that the equation system from Corollary 3.31B has a finite solution.[87]

87: But they do not necessarily reveal the exact solution, just an upper bound.

► **Example 3.32** (Computing the expected runtime of a pPDA) Reconsider once again the pPDA Δ from Example 3.24 (restated in the margin). We have already proved in Example 3.27 on page 74 that Δ is universally AST. Here, we show that Δ is even universally *PAST*. We do so by computing the expected runtime, revealing that it is finite.

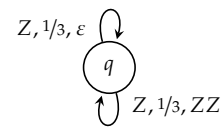


Figure 3.11: pPDA Δ from Example 3.24 restated.

We proceed as in Corollary 3.31B. Since both the state space and the stack alphabet of Δ are singleton sets, we obtain just one linear equation:

$$\langle qZ\rangle \;=\; \frac{2}{3} + \frac{1}{3}\Big(\langle qZ\rangle + [qZ\,q]\cdot\langle qZ\rangle\Big)\,.$$

Using that $[qZ\,q] = \frac{3-\sqrt{5}}{2}$ (as shown in Example 3.27) and solving,[88] we find that the expected runtime from qZ is

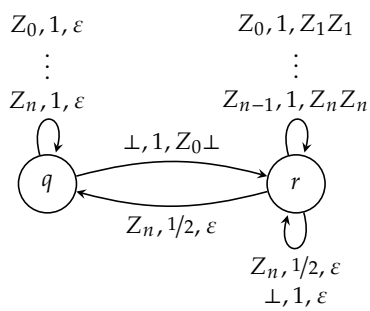
$$\langle qZ\rangle \;=\; 1-\sqrt{5} \;\approx\; 1.236\,.$$

88: The equation also admits the solution ∞, which we discard since it is not the least solution.

In words, the expected number of transitions taken before termination is approximately 1.236 on average. This number may appear too small at first glance. However, notice that Δ has a 1 : 3 chance of terminating immediately, without taking any transitions at all.

Since Δ is qZ-PAST for its only head qZ, it follows with Corollary 3.31D that Δ is universally PAST. This implies in particular that Δ is universally AST, which we had already proved directly in Example 3.27 on page 74. ◄

► **Example 3.33** (A pPDA with doubly exponential expected runtime) For $n \geq 1$, consider the following (proper) pPDA Δ_n with stack alphabet $\Gamma_n = \{\bot, Z_0, \ldots, Z_n\}$:



The overall idea of Δ_n is to push 2^n symbols Z_n onto the stack while in state r. This is achieved via a *repeating squaring* mechanism implemented with

the help of the intermediate symbols $Z_0, \ldots, Z_{n-1}$. To terminate (i.e., reach the empty stack), all 2^n instances of Z_n must be popped without leaving state r, which happens only with doubly exponentially small probability $1/2^{2^n}$. Otherwise, q is entered and the process starts over. On average, Δ_n needs 2^{2^n} attempts to successfully pop all Z_n symbols without leaving state r. The expected runtime from initial configuration $q\bot$ is thus at least 2^{2^n}.

89: See Theorem 3.11 on page 58.

More formally, by the fundamental equation system[89] we have:

$$\begin{aligned}
[rZ_n\, r] &= \frac{1}{2} = \frac{1}{2^{2^0}} \\
[rZ_{n-1}\, r] &= [rZ_n\, r]^2 + [rZ_n\, q] \cdot \underbrace{[qZ_n\, r]}_{=0} = \frac{1}{4} = \frac{1}{2^{2^1}} \\
&\vdots \\
[rZ_0\, r] &= [rZ_1\, r]^2 + [rZ_1\, q] \cdot \underbrace{[qZ_1\, r]}_{=0} = \frac{1}{2^{2^n}}
\end{aligned}$$

Therefore, $[rZ_0\, q] = 1 - 1/2^{2^n}$. By Corollary 3.31c, the expected runtime $\langle q\bot \rangle$ from initial head $q\bot$ satisfies:

$$\begin{aligned}
\langle q\bot \rangle &= 1 + \langle rZ_0 \rangle + \sum_{s \in Q} [rZ_0\, s] \cdot \langle s\bot \rangle \\
&\geq 1 + [rZ_0\, q] \cdot \langle q\bot \rangle \,.
\end{aligned}$$

It follows that

$$\langle q\bot \rangle \geq \frac{1}{1 - [rZ_0\, q]} = 2^{2^n} \,.$$

Therefore, the expected runtime of Δ_n is at least *doubly exponential* (in magnitude) in n, whereas the size of Δ_n is only linear in n. ◀

90: See Definition 3.23 on page 72.

91: See Theorem 3.19 on page 65.

▷ **Remark 3.34** (Expected runtimes via return moment triples) For the subclass of pPDA that are both universally AST[90] and proper, the expected runtime equation system from Corollary 3.31c can also be derived from the equation system of return moment triples.[91] To see this, recall that the latter are the least solution to the following equations:

$$\begin{aligned}
[qZ\, t]^{\langle 1 \rangle} &= [qZ\, t] + \sum_{qZ \xrightarrow{p} rY} p \cdot [rY\, t]^{\langle 1 \rangle} \\
&\quad + \sum_{qZ \xrightarrow{p} rYX} p \cdot \sum_{s \in Q} \Big([rY\, s]^{\langle 1 \rangle} \cdot [sX\, t] + [rY\, s] \cdot [sX\, t]^{\langle 1 \rangle} \Big) \,.
\end{aligned}$$

Summing the equations over $t \in Q$ on both sides, and writing, for every head qZ, the sum $\sum_{t \in Q} [qZ\, t]^{\langle 1 \rangle}$ as the symbol $\langle qZ \rangle$, we obtain the equation

system

$$\langle qZ\rangle = \sum_{t\in Q}[qZ\,t] + \sum_{qZ\xrightarrow{p} rY} p\cdot\langle rY\rangle + \sum_{qZ\xrightarrow{p} rYX} p\cdot\Big(\langle rY\rangle\cdot\sum_{t\in Q}[sX\,t] + \sum_{s\in Q}[rY\,s]\cdot\langle sX\rangle\Big)\,.$$

Since we have assumed universal AST and properness, we have $\sum_{t\in Q}[qZ\,t] = 1$ for every head qZ. This observation transforms the previous equation system into the one from Corollary 3.31c. Moreover, again due to universal AST and properness, we know that the values $\langle qZ\rangle = \sum_{t\in Q}[qZ\,t]^{\langle 1\rangle}$ are indeed the expected runtimes from every head qZ.

To our knowledge, the above argument cannot easily be extended to non-AST or improper pPDA. On the other hand, our appoach based on expressing the expected runtime as a behavior of a simple final weight vector naturally accommodates improper and non-AST pPDA as well. ..... ◁

3.2.3 *Conditional Expected Runtimes of pPDA*

Finally, we can also define the *conditional* expected runtime of a pPDA, again referring to the same notion in the underlying DTMC:

Definition 3.35 ***(Conditional expected runtimes of pPDA)*** Let $\Delta = (Q, \Gamma, P)$ be a pPDA with an initial distribution $\mathbf{g}$ on its configurations $Q\times\Gamma^*$.

A. The *(first) termination moment* of Δ from $\mathbf{g}$ is defined as the (first) termination moment[92] of the induced DTMC $\mathfrak{M}_\Delta$ from $\mathbf{g}$, that is, the behavioral moment

$$[\![\mathbf{g}\,,\Delta\,,\mathbf{t}]\!]^{\langle 1\rangle}\,,$$

where $\mathbf{t}$ are the immediate termination probabilities of $\mathfrak{M}_\Delta$.

B. We define the *conditional expected runtime* of Δ from $\mathbf{g}$ as the conditional expected runtime of $\mathfrak{M}_\Delta$ from $\mathbf{g}$, that is, the quotient

$$\frac{[\![\mathbf{g}\,,\Delta\,,\mathbf{t}]\!]^{\langle 1\rangle}}{[\![\mathbf{g}\,,\Delta\,,\mathbf{t}]\!]}$$

provided the termination probability $[\![\mathbf{g}\,,\Delta\,,\mathbf{t}]\!]$ is nonzero (otherwose, the conditional expected runtime is undefined).

C. Δ is *conditionally positively almost surely terminating* (cPAST) from $\mathbf{g}$ (or $\mathbf{g}$-cPAST, for short) if its conditional expected runtime is either finite or undefined.

92: See Definition 2.51 on page 43.

D. Δ is *universally cPAST* if Δ is cPAST from *all* initial distributions on its configurations $Q \times \Gamma^*$. △

The notion of cPAST in pPDA is of particular technical interest in this work. Indeed, as we will see in Chapter 4, cPAST can be viewed as an important well-behavedness or *nondegeneracy* property of pPDA. In the remainder of this section, we establish a connection between cPAST and the return moment triples[93] $[qZ\,r]^{\langle 1\rangle}$, specifically that universal cPAST is equivalent to all these triples being finite.[94] This result is straightforward for proper pPDA, but nontrivial for improper ones. We begin with two auxiliary lemmas.

93: See Theorem 3.19 on page 65.

94: Formally stated as Theorem 3.38 on the following page.

Lemma 3.36 *A pPDA Δ is universally cPAST if and only if Δ is cPAST from each head $qZ \in Q \times \Gamma$.*

Proof. Similar to the proof of Corollary 3.26D. □

Lemma 3.37 *Let $A \in \mathbb{R}_{\geq 0}^{n\times n}$ be a matrix and let $\mathbf{b}, \mathbf{c} \in \mathbb{R}_{\geq 0}^{n}$ be vectors. Suppose that $\boldsymbol{x}$ and $\mathbf{y}$ are the least solutions to the linear equation systems*

$$\boldsymbol{x} \;=\; A\boldsymbol{x} + \mathbf{b} \quad \textit{and} \quad \boldsymbol{y} \;=\; A\boldsymbol{y} + \mathbf{c}\,,$$

respectively, over the extended *nonnegative reals. Furthermore, assume that $\boldsymbol{x}$ and $\mathbf{y}$ satisfy $\boldsymbol{x}_i = 0$ iff $\mathbf{y}_i = 0$ for all $i \in [1..n]$. Then it holds that*

$$\boldsymbol{x} < \infty \quad \Longleftrightarrow \quad \mathbf{y} < \infty\,;$$

that is, either both systems have a nonnegative finite *solution, or neither does.*

Lemma 3.36 follows from general results on nonnegative matrices discussed in Chapter 4; see Section B.1 for the proof.

We state our characterization of universal cPAST in pPDA in Theorem 3.38 below. The challenging part lies in the $\Longleftarrow$ direction, with the difficulty arising from *improper*[95] pPDA. Notably, there exist improper pPDA where no head can reach the empty stack, yet the conditional expected runtime is well-defined and, in fact, nonzero. A minimal example is provided in the margin. This is in contrast to *proper* pPDA, where any head not reaching the empty stack has *undefined* conditional expected runtime, and is thus trivially cPAST. It is not immediately clear why the conditional expected runtime could not be infinite in such improper pPDA. Indeed, it turns out that this direction only holds for *finite* pPDA:

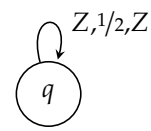


Figure 3.12: An improper pPDA that satisfies $[qZ\,q]^{\langle 1\rangle} = 0$ and has a conditional expected runtime of 2 from qZ.

95: That is, where the probabilities of available transitions sum to less than one for some heads; see Definition 3.22.

Theorem 3.38 (Characterization of universal cPAST) *A finite pPDA is universally cPAST*[96] *if and only if all its return moment triples*[97] *are finite, that is, if for all $qZ \in Q \times \Gamma$ and $r \in Q$ we have $[qZ\,r]^{\langle 1 \rangle} < \infty$.*

96: See Definition 3.35 on page 80.

97: See Theorem 3.19 on page 65.

Proof. For the easier direction $\Longrightarrow$, assume that the pPDA Δ is universally cPAST. By Definition 3.35, Δ is in particular cPAST from each initial head $qZ \in Q \times \Gamma$, that is, either $[\![\mathbf{e}_{qZ}, \Delta, \mathbf{t}]\!] = 0$, meaning that he termination probability from qZ is zero, or $[\![\mathbf{e}_{qZ}, \Delta, \mathbf{t}]\!]^{\langle 1 \rangle} < \infty$. Since termination with probability zero entails a termination moment of zero, we have $[\![\mathbf{e}_{qZ}, \Delta, \mathbf{t}]\!]^{\langle 1 \rangle} < \infty$ in either case. Finally, since $\mathbf{e}_{r\varepsilon} \leq \mathbf{t}$, it follows that

$$[qZ\,r]^{\langle 1 \rangle} \;=\; [\![\mathbf{e}_{qZ}, \Delta, \mathbf{e}_{r\varepsilon}]\!]^{\langle 1 \rangle} \;\leq\; [\![\mathbf{e}_{qZ}, \Delta, \mathbf{t}]\!]^{\langle 1 \rangle} \;<\; \infty\,.$$

As mentioned, the proof of direction $\Longleftarrow$ is more involved. We start by recalling the linear equation systems from Theorems 3.19 and 3.20. They respectively characterize the behaviors and behavioral moments of Δ for the simple final weight vector $\mathbf{t}$, the immediate termination probabilities of Δ:

$$\begin{aligned} \mathbf{h} &= A_\Delta(\mathbf{r})\mathbf{h} + \mathbf{t}\big|_{Q\times\Gamma}\,, \\ \mathbf{h}^{\langle 1 \rangle} &= A_\Delta(\mathbf{r})\mathbf{h}^{\langle 1 \rangle} + A_\Delta(\mathbf{r} + \mathbf{r}^{\langle 1 \rangle})\mathbf{h}\,. \end{aligned}$$

In the first system, $\mathbf{h}$ is a vector such that $\mathbf{h}(qZ) = [\![\mathbf{e}_{qZ}, \Delta, \mathbf{t}^+]\!]$ for each head qZ of Δ. In words, $\mathbf{h}$ contains, for each head, the probability to terminate without emptying the stack. Also recall that $\mathbf{r}$ are the return probability triples of Δ. We have $\mathbf{h} \leq \mathbf{1}$ and $\mathbf{r} \leq \mathbf{1}$ since these vectors contain probabilities only. The second system similarly characterizes the behavioral moments $\mathbf{h}^{\langle 1 \rangle}(qZ) = [\![\mathbf{e}_{qZ}, \Delta, \mathbf{t}^+]\!]^{\langle 1 \rangle}$. Notice that this system contains the vector $\mathbf{h}$ from the first system, as well as the return moment triples $\mathbf{r}^{\langle 1 \rangle}$—the latter are finite by assumption.

In the following argument, we apply Lemma 3.37 to these two equation systems to establish that $\mathbf{h}^{\langle 1 \rangle} < \boldsymbol{\infty}$, which, in turn, implies universal cPAST as claimed:

$$\begin{aligned} & \forall qZ \in Q \times \Gamma, r \in Q\colon [qZ\,r]^{\langle 1 \rangle} < \infty && \text{(Assumption)} \\ \Longleftrightarrow\; & \mathbf{r}^{\langle 1 \rangle} < \infty && \text{(Change of notation; see Theorem 3.19)} \\ \Longrightarrow\; & A_\Delta(\mathbf{r} + \mathbf{r}^{\langle 1 \rangle})\mathbf{h} < \infty && \text{(Since } \mathbf{r}, \mathbf{r}^{\langle 1 \rangle}, \mathbf{h} < \infty) \\ \Longrightarrow\; & \mathbf{h}^{\langle 1 \rangle} < \infty && (\dagger) \end{aligned}$$

$$\begin{aligned}
&\iff \forall qZ \in Q \times \Gamma\colon [\![\mathbf{e}_{qZ}, \Delta, \mathbf{t}^+]\!]^{\langle 1\rangle} < \infty && \text{(Change of notation)}\\
&\iff \forall qZ \in Q \times \Gamma\colon [\![\mathbf{e}_{qZ}, \Delta, \mathbf{t}]\!]^{\langle 1\rangle} < \infty && \text{(Since } [\![\mathbf{e}_{qZ}, \Delta, \mathbf{t}]\!]^{\langle 1\rangle} = [\![\mathbf{e}_{qZ}, \Delta, \mathbf{t}^+]\!]^{\langle 1\rangle} + \textstyle\sum_r [qZ\, r]^{\langle 1\rangle}\text{)}\\
&\iff \forall qZ \in Q \times \Gamma\colon \Delta \text{ is } \mathbf{e}_{qZ}\text{-cPAST} && \text{(Definition 3.35)}\\
&\iff \Delta \text{ is universally cPAST.} && \text{(Lemma 3.36)}
\end{aligned}$$

In the above derivation, (†) is an application of Lemma 3.37 with

- $A = A_\Delta(\mathbf{r})$,
- $\boldsymbol{x} = \mathbf{h}$,
- $\mathbf{b} = \mathbf{t}|_{Q\times\Gamma}$,
- $\boldsymbol{y} = \mathbf{h}^{\langle 1\rangle}$, and
- $\mathbf{c} = A_\Delta(\mathbf{r} + \mathbf{r}^{\langle 1\rangle})\mathbf{h}$.

Since we indeed have $\mathbf{h}(qZ) = 0$ if and only if $\mathbf{h}^{\langle 1\rangle}(qZ) = 0$,[98] and moreover $\mathbf{h} < \boldsymbol{\infty}$, Lemma 3.37 yields that $\mathbf{h}^{\langle 1\rangle} < \boldsymbol{\infty}$. □

98: By Lemma 2.34 on page 33.

3.3 *Open Questions*

1. Does the fundamental equation system[99] extend to infinite (i.e., with infinitely many states or stack symbols) WPDA as well? If so, could this be leveraged to reason about infinite pPDA symbolically, similar to existing approaches that encode infinite Markov decision processes as probabilistic programs [12, 14]?
2. Are there further natural equation systems for WPDA or pPDA? Specifically, while the fundamental equation system characterizes certain (two-sided) behaviors, are there alternative systems that characterize, for example, backward behaviors directly?

99: See Theorem 3.11 on page 58.

[12]: Batz (2024), "Automated deductive verification of probabilistic programs".

[14]: Batz et al. (2024), "J-P: MDP. FP. PP - Characterizing Total Expected Rewards in Markov Decision Processes as Least Fixed Points with an Application to Operational Semantics of Probabilistic Programs".

3.4 *Additional Bibliographic Remarks*

As mentioned, the WPDA[100] we consider here are similar to those studied by Kuich [87, Section 6]. We note the following technical differences: First, we only allow push transitions that increase the stack height by *at most one symbol*, whereas Kuich also allows pushing multiple symbols at once. This is nonessential (and not particularly natural in the context of modeling recursive programs), as pushing multiple symbols can be simulated by introducing auxiliary states. Second, our WPDA are defined without explicit initial and final weights, nor with an initial stack symbol. Finally, unlike Kuich, we

100: See Definition 3.2 on page 51.

[87]: Kuich (1997), "Semirings and Formal Power Series: Their Relevance to Formal Languages and Automata".

do not explicitly consider WPDA over a restricted set of admissible weights.[101]

The idea of viewing WPDA as generators of their—generally infinite—induced weighted automata,[102] as well as the resulting notion of behavior is also due to Kuich [87, Section 6]. The same holds for the fundamental equation system[103] of a WPDA [87, Corollary 6.5]. The *notation* for the fundamental system that we use is due to Esparza, Kucera, and Mayr [44]. To our knowledge, the equation systems for behaviors relative to simple final weight vectors and behavioral moments[104] have not appeared explicitly in prior work on the general weighted setting.

Expected runtimes, and more generally, expected rewards in pPDA have been studied extensively [21, 22, 45]. However, these works focus on expectations *conditioned* on reaching the empty stack. Our notion of *conditional* expected runtime[105] is a special case of such conditional expected rewards and was explicitly studied by Esparza, Kucera, and Mayr [45, Example 3.3]. Indeed, our equation system for return moment triples[106] in pPDA can be derived from a similar system in [45, Theorem 3.1] (see [136, Appendix C] for details). However, our generating functions-based proof of Theorem 3.19 differs substantially from that in [45, Theorem 3.1] and generalizes to arbitrary WPDA.

The equation system characterization of (unconditional) expected runtimes in pPDA[107] is, to our knowledge, an original contribution of our previous paper [135]. It cannot easily be derived from the known equations for conditional expectations; see the discussion in Remark 3.34 on page 79.

The terminology *simple* weight vectors[108] was first introduced in the context of pPDA as a reward function that depends only on the state of a configuration [45]. It was later generalized to reward functions that depend only on the head of a configuration [21]. Our definition mildly generalizes this, from real-valued reward functions to weight vectors over an arbitrary semiring.

101: See Section 2.4 for a discussion on weighted automata where only a subset of the semiring elements are allowed as weights.

102: See Definition 3.5 on page 54.

103: See Theorem 3.11 on page 58.

[44]: Esparza et al. (2004), "Model Checking Probabilistic Pushdown Automata".

104: See overview on Figure 3.7 on page 70.

[21]: Brázdil et al. (2013), "Analyzing probabilistic pushdown automata".

[22]: Brázdil et al. (2015), "Runtime analysis of probabilistic programs with unbounded recursion".

[45]: Esparza et al. (2005), "Quantitative Analysis of Probabilistic Pushdown Automata: Expectations and Variances".

105: See Definition 3.35 on page 80.

106: See Theorem 3.19 on page 65.

107: See Corollary 3.31 on page 76.

[135]: Winkler et al. (2023).

108: See Definition 3.14 on page 60.

[45]: Esparza et al. (2005).

[21]: Brázdil et al. (2013).

Certificates for Probabilistic Pushdown Automata

4

4.1 Background . . 87
4.1.1 Polynomials . . 87
4.1.2 Positive Polynomial Systems . 88
4.1.3 Feasible and Clean PPSs . . . 92
4.1.4 Linear PPSs . . 95
4.2 Certificates for PPSs 95
4.2.1 Upper Bounds 96
4.2.2 Lower Bounds . 110
4.3 Certificates for pPDA 114
4.3.1 Return Probabilities 114
4.3.2 PAST 117
4.4 Open Questions 121
4.5 Bibliographic Remarks 122

The leading question of this chapter is as follows:

> *What sort of information must we exhibit to convince someone—a person or computer program—that a given pPDA (probabilistic pushdown automaton) has a certain property, such as being PAST (positively almost surely terminating)?*

It is clear that merely claiming the fact is insufficient—the other party would rightfully demand some proof. To increase our chances of convincing them, we should moreover seek proofs that are *simple*: small and easy to check. This principle is the idea of *certificates*.

Certificates Certifying a property is different from verifying it algorithmically. For example, a Boolean satisfiability (SAT) solver might (correctly) output `sat` or `unsat` on any given instance. However, such yes–no answers leave users no other choice but to trust the solver blindly. In contrast, a *certifying* SAT solver would accompany every answer with an easy-to-check proof (e.g., a satisfying assignment for the answer `sat`).

To a certain extent, certificates are a soft, informal concept. The study of rigorously defined notions of certificates belongs to the field of complexity theory.[1] The problem with too rigorous definitions is that they may easily exclude practically useful certificates. A prominent example are *resolution proofs* in SAT solving: Although they occasionally violate the requirement of being small, they are widely used in practice as certificates for `unsat` [68].

Certificate Design Goals For the reasons outlined in the previous paragraph, I refrain from giving a formal definition of certificates. Let us, however, agree on the following design goals:[2]

1. *Soundness*: The existence of a valid certificate implies that the claimed property actually holds.[3]
2. *Ease of verification*: Certificates should be easy to verify, both in terms of theoretical complexity (e.g., in linear time[4]) and implementation effort of the verifier.
3. *Succinctness*: Certificates should be reasonably small.
4. *Completeness*: Every true property—or at least most of them—should admit a certificate.

1: For example, a decision problem belongs to NP ∩ coNP if and only if all yes-instances and all no-instances admit certificates of polynomial size that can be verified in polynomial time.

[68]: Heule et al. (2015), "Proofs for satisfiability problems".

2: This list is roughly ordered by importance.

3: This requirement is the only fully formal one.

4: As mentioned, we shall not be too strict about this.

5. *Computability*: Certificates should be computable by reasonable extensions to existing (verification) algorithms, thereby making the latter *certifying* [96].

[96]: McConnell et al. (2011), "Certifying algorithms".

What constitutes a good certificate is often a nontrivial question. For instance, in the SAT solving example, certificates for `unsat` (e.g., resolution proofs) are less obvious than for `sat`.

Positive Polynomial Systems What, then, are suitable certificates for pPDA properties such as PAST? We saw in Chapter 3 that the analysis of pPDA revolves around its *return triples*, that is, the probabilities $[qZr]$ for states q, r and a stack symbol Z. These probabilities are the least solution of the pPDA's fundamental equation system.[5] Thus, if we can certify the return probabilities—or sufficiently tight bounds on them—, we should be able to certify a great deal of pPDA properties.

5: See Theorem 3.11 on page 58.

Recall that the fundamental system of a pPDA is a polynomial equation system of a certain shape: It involves only polynomials with nonnegative coefficients on one side of the equations, and individual variables on the other side. Following Esparza, Kiefer, and Luttenberger [43], we shall call these systems *positive polynomial systems*.[6,7] Because of their central importance to pPDA, a significant portion of this chapter treats certificates for bounds on the (least) solution of such systems.

[43]: Esparza et al. (2010), "Computing the Least Fixed Point of Positive Polynomial Systems".

6: Other authors have used the term *monotone system of polynomial equations* [50].

7: pPDA generate only a certain subclass of these systems. However, our certificates apply to the general class.

Chapter Outline, Contributions, and Origins After introducing some preliminaries in Section 4.1, this chapter proceeds as follows:

Section 4.2 (Certificates for Positive Polynomial Systems) We develop certificates for upper and lower bounds on the least solution of positive polynomial systems. A key question we investigate is *completeness*—to what extent do valid certificates exist for all true bounds?

Section 4.3 (Certificates for Probabilistic Pushdown Automata) We apply the results from Section 4.2 to pPDA, developing (mildly incomplete) certificates for bounds on the return probabilities as well as *complete* certificates for PAST—the latter is a main contribution of this thesis.

As in the preceding chapters, we defer open questions and additional bibliographic remarks to the end (Sections 4.4 and 4.5). Most of the material presented in this chapter relies on our prior publication [135].

[135]: Winkler et al. (2023), "On Certificates, Expected Runtimes, and Termination in Probabilistic Pushdown Automata".

4.1 Positive Polynomial Systems: Background

In this section, we introduce some concepts needed in the remainder of this chapter: polynomials (Section 4.1.1), positive polynomial systems along with some of their basic properties (Section 4.1.2 on the following page), the notions of feasibility and cleanness (Section 4.1.3 on page 92), and positive linear systems (Section 4.1.4 on page 95).

Throughout the whole chapter, we shall use the following notation for comparisons between vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ over some index set I:

1. $\boldsymbol{x} \leq \boldsymbol{y}$ means that $\boldsymbol{x}_i \leq \boldsymbol{y}_i$ for all $i \in I$.
2. $\boldsymbol{x} < \boldsymbol{y}$ means that $\boldsymbol{x} \leq \boldsymbol{y}$ and $\boldsymbol{x} \neq \boldsymbol{y}$.
3. $\boldsymbol{x} \lneq \boldsymbol{y}$ means that $\boldsymbol{x}_i < \boldsymbol{y}_i$ for *all* $i \in I$.

For example, if $\boldsymbol{x} > \mathbf{0}$, then $\boldsymbol{x}$ is *componentwise positive*.

4.1.1 Polynomials

Let $X = \{x_1, \ldots, x_n\}$ be a finite set of variables. The set $\mathbb{R}_{\geq 0}[X]$ of *polynomials* with nonnegative real coefficients and variables in X is the smallest set of functions $f : \mathbb{R}_{\geq 0}^X \to \mathbb{R}_{\geq 0}$ containing the constant functions $c \in \mathbb{R}_{\geq 0}$, the projection functions[8]

$$x_i : \ \mathbb{R}_{\geq 0}^X \to \mathbb{R}_{\geq 0}\,, \ (x_1 \ \ldots \ x_n)^\top \mapsto x_i$$

for all $x_i \in X$, and that is closed under addition and multiplication.

8: In the univariate case $X = \{x\}$, projection degenerates to the identity function x.

We may lift every polynomial $f \in \mathbb{R}_{\geq 0}[X]$ to a function

$$\overline{f} : \ \overline{\mathbb{R}}_{\geq 0}^X \to \overline{\mathbb{R}}_{\geq 0}$$

over the *extended* nonnegative reals $\overline{\mathbb{R}}_{\geq 0}$. Thus, we may plug ∞ into polynomials with nonnegative real coefficients.[9] We shall not strictly distinguish between f and $\overline{f}$ in the remainder.

9: This is not easily possibly for polynomials with arbitrary real coefficients: It would require defining expressions like $\infty - \infty$.

We often use the standard notation $\mathbb{R}_{\geq 0}[x_1, \ldots, x_n]$ instead of $\mathbb{R}_{\geq 0}[X]$. Similarly, for a given argument $\boldsymbol{x} = (x_1 \ \ldots \ x_n)^\top \in \mathbb{R}_{\geq 0}^X$, we may write $f(x_1, \ldots, x_n)$ instead of $f(\boldsymbol{x})$.

To illustrate these notions, consider the polynomial

$$f \ = \ (x+1)(y+2) \ \in \ \mathbb{R}_{\geq 0}[x, y]\,.$$

We have $f(3,4) = (3+1)(4+2) = 24$, and $f(\infty, y) = \infty$ for all $y \in \overline{\mathbb{R}}_{\geq 0}$.

We can (re)write every polynomial as a finite weighted sum of pairwise distinct *monomials* (i.e., products of variables raised to nonnegative integer powers). This may include the empty monomial $x_1^0 \cdots x_n^0 = 1$, whose weight—if present—we call the *constant coefficient* of the polynomial. For example, we can rewrite the polynomial f from the previous paragraph as

$$f \;=\; xy + 2x + y + 2\,,$$

where 2 is the constant coefficient. The (total) *degree* of a polynomial is the maximum degree of its monomials, where the degree of a monomial is the sum of the exponents occurring in it. Polynomials of degree 1 are called *linear*; polynomials of degree at least 2 are called *nonlinear*. Our example polynomial f has degree 2 and is thus nonlinear.

The *partial derivative* operator with respect to a variable $x \in X$ is the unique function $\partial_x \colon \mathbb{R}_{\geq 0}[X] \to \mathbb{R}_{\geq 0}[X]$ that preserves addition and scalar multiplication,[10] and acts on monomials according to the standard rule

$$\partial_{x_i}\left(x_1^{e_1} \cdots x_n^{e_n}\right) \;=\; e_i \cdot x_1^{e_1} \cdots x_i^{e_i - 1} \cdots x_n^{e_n}\,.$$

This rule entails that the partial derivative is zero if $e_i = 0$. Notably, partial derivatives preserve nonnegative coefficients; thus, ∂_x is indeed a well-defined function with codomain $\mathbb{R}_{\geq 0}[X]$. The example polynomial $f = xy+2x+y+2$ satisfies $\partial_x f = y+2$ and $\partial_y f = x+1$.

10: Formally, $\partial_x\left(f+g\right) = \partial_x f + \partial_x g$, and $\partial_x\left(cf\right) = c\partial_x f$ for all $f, g \in \mathbb{R}_{\geq 0}[X]$ and $c \in \mathbb{R}_{\geq 0}$.

4.1.2 The Basics of Positive Polynomial Systems

Intuitively, a positive polynomial system over variables $x_1, \ldots, x_n$ is an equation system

$$\left.\begin{array}{rcl} x_1 & = & f_1(x_1, \ldots, x_n) \\ & \vdots & \\ x_n & = & f_n(x_1, \ldots, x_n) \end{array}\right\}\,, \tag{4.1}$$

where, for all $i \in [1..n]$, $f_i \in \mathbb{R}_{\geq 0}[x_1, \ldots, x_n]$ is a polynomial with nonnegative[11] real coefficients. For example, the fundamental system of a pPDA is a positive polynomial system.[12]

11: Even though we speak of *positive* polynomial systems, it is technically convenient to allow zero as a coefficient.

12: See Theorem 3.11 on page 58.

For our purposes, it is convenient to identify Equation (4.1) with a function whose *fixed points* coincide with the system's solutions over the extended nonnegative reals:

Definition 4.1 (PPS: *Positive polynomial system* [43]) Let $X = \{x_1, \ldots, x_n\}$ be a finite set of variables.[13] A PPS over X is a function

$$\boldsymbol{f} : \overline{\mathbb{R}}_{\geq 0}^{X} \to \overline{\mathbb{R}}_{\geq 0}^{X}, \quad \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \mapsto \begin{pmatrix} f_1(x_1, \ldots, x_n) \\ \vdots \\ f_n(x_1, \ldots, x_n) \end{pmatrix}$$

with polynomials $f_i \in \mathbb{R}_{\geq 0}[X]$ for all $i \in [1..n]$. We call f_i the x_i*-component* of $\boldsymbol{f}$ and denote it by $\boldsymbol{f}_{x_i}$. △

13: For technical reasons, it is convenient to allow $X = \emptyset$, which yields the trivial *empty* PPS.

Note that a PPS contains as many polynomials as variables. Therefore, we could have defined PPSs as vectors of polynomials of type $\mathbb{R}_{\geq 0}[X]^X$. However, the direct definition as a function is preferable: The functional view more closely resembles the way we use PPSs in the sequel.

► **Example 4.2** (One- and multi-dimensional PPSs) We call a PPS *n-dimensional* if it has n variables (i.e., $|X| = n$).

– *One-dimensional* PPSs are just univariate polynomials with nonnegative coefficients. Examples of one-dimensional PPSs include:

$x \mapsto 42$,	which is constant;
$x \mapsto x + 1$,	which is linear, but not constant;
$x \mapsto \frac{1}{4}x^2 + \frac{1}{2}x + \frac{1}{4}$,	which is quadratic (i.e., nonlinear).

– For PPSs with $n > 1$ variables we prefer vector notation. The following are examples of two-dimensional PPSs over $X = \{x, y\}$:

$$\begin{pmatrix} x \\ y \end{pmatrix} \mapsto \begin{pmatrix} x + y + 1 \\ \frac{1}{2}x + y \end{pmatrix}, \quad \text{which is linear;}$$

$$\begin{pmatrix} x \\ y \end{pmatrix} \mapsto \begin{pmatrix} xy + 1 \\ y^2 + y \end{pmatrix}, \quad \text{which is nonlinear.}$$

Notice that, apart from nonnegative coefficients, we do not make any other assumptions about the polynomials in a PPS. In particular, every component may depend only on a (possibly empty) subset of the variables. For instance, in the preceding nonlinear two-dimensional example PPS, the y-component does not depend on x.[14] ◄

14: We shall consider *strongly connected* PPSs [43] in Chapter 5, where every variable depends (directly or indirectly) on every other variable.

We summarize some standard—yet crucial—PPS properties in the following lemma:

Lemma 4.3 (Monotonicity, least fixed points, continuity) *Let f be a PPS over variables X.*

A. *f is* monotonic *relative to the componentwise partial order $\leq$ on $\overline{\mathbb{R}}_{\geq 0}^X$; that is, for all $v, w \in \overline{\mathbb{R}}_{\geq 0}^X$ we have*

$$v \leq w \quad \Longrightarrow \quad f(v) \leq f(w) .$$

B. *f has an* LFP *(least fixed point) μf; that is, $f(\mu f) = \mu f$ and for all $v \in \overline{\mathbb{R}}_{\geq 0}^X$ such that $f(v) = v$, we have $\mu f \leq v$.*

C. *f is* continuous *on $\mathbb{R}_{\geq 0}^X$ in the usual sense; that is, for all componentwise converging sequences $v_0, v_1, \ldots$ in $\mathbb{R}_{\geq 0}^X$ we have*[15]

$$\lim_{n\to\infty} f(v_n) = f\left(\lim_{n\to\infty} v_n\right) .$$

15: Of course, f remains continuous if we extend it to a function on all of $\mathbb{R}^X$; however, negative reals do not occur in this chapter.

D. *f is ω-*continuous*; that is, for all* monotonic *sequences $v_0 \leq v_1 \leq \ldots$ in $\overline{\mathbb{R}}_{\geq 0}^X$, we have*

$$\sup_{n\in\mathbb{N}} f(v_n) = f\left(\sup_{n\in\mathbb{N}} v_n\right) ,$$

where the suprema are taken componentwise.[16]

16: These suprema may evaluate to ∞ in some or all components.

E. *The* Kleene sequence[17] *$0 \leq f(0) \leq f^2(0) \leq \ldots$ is monotonic and*

$$\sup_{n\in\mathbb{N}} f^n(0) = \mu f . \tag{4.2}$$

17: Named after Stephen C. Kleene [79].

If $\mu f < \infty$, we may replace $\sup_{n\in\mathbb{N}}$ in Equation (4.2) by $\lim_{n\to\infty}$.[18]

18: PPS satisfying $\mu f < \infty$ are called *feasible*; see Definition 4.5 on page 92.

F. *Item E admits the following variant: For all $0 \leq v \leq \mu f$,*

$$\sup_{n\in\mathbb{N}} f^n(v) = \mu f .$$

However, the sequence $f^0(v), f^1(v), f^2(v), \ldots$ need not be monotonic in this case.

Proof. Item A (monotonicity) follows from the fact that the components of f have nonnegative coefficients only.

For Item B (least fixed point) we observe that $\overline{\mathbb{R}}_{\geq 0}^X$ with the componentwise partial order $\leq$ is a *complete lattice* (i.e., has all suprema) and apply the Knaster-Tarski Fixed Point Theorem, which implies in particular that every monotonic endofunction on a complete lattice has an LFP.[19]

19: See Lemma 4.10 on page 95. We do not yet state the theorem here as it has a prominent application in Section 4.2.1.

Item C holds because polynomials are continuous functions.

For Item D (ω-continuity) we distinguish two cases: If the sequence $\boldsymbol{v}_0, \boldsymbol{v}_1, \ldots$ converges in $\mathbb{R}^X_{\geq 0}$, the statement follows from Item C, using that the limit of a nondecreasing sequence—assuming it exists[20]—is the supremum of that sequence. Otherwise, the sequence diverges in at least one component. Let us consider the one-dimensional case for simplicity. Assume that $v_0, v_1, \ldots$ is a nondecreasing diverging, and thus *unbounded* sequence in $\mathbb{R}_{\geq 0}$. The case where $\boldsymbol{f}$ is constant is trivial, so assume that $\boldsymbol{f}$ is nonconstant. Since nonconstant polynomials are unbounded functions, it follows that $\boldsymbol{f}(v_0), \boldsymbol{f}(v_1), \ldots$ is also unbounded, which already implies the claim. We can extend this argument to the multidimensional case by considering each component separately.

20: The limit of a nondecreasing sequence exists iff the sequence is *bounded*.

Item E is an application of Kleene's Fixed Point Theorem, which states that every ω-continuous endofunction on a complete lattice—by Item D, $\boldsymbol{f}$ is such a function—has an LFP that is equal to the supremum of the Kleene sequence.

Finally, Item F is a consequence of Item E: By monotonicity, it holds for all $n \in \mathbb{N}$ that

$$\boldsymbol{f}^n(\boldsymbol{0}) \leq \boldsymbol{f}^n(\boldsymbol{v}) \leq \boldsymbol{f}^n(\mu\boldsymbol{f}) = \mu\boldsymbol{f} \,.$$

It follows that

$$\mu\boldsymbol{f} \overset{\text{E}}{=} \sup_{n\in\mathbb{N}} \boldsymbol{f}^n(\boldsymbol{0}) \leq \sup_{n\in\mathbb{N}} \boldsymbol{f}^n(\boldsymbol{v}) \leq \mu\boldsymbol{f}$$

and hence the claim. An example where the sequence $(\boldsymbol{f}^n(\boldsymbol{v}))_{n\in\mathbb{N}}$ is *not* monotonic is the two-dimensional PPS $\boldsymbol{f}$ given by

$$\begin{pmatrix} x \\ y \end{pmatrix} \quad \mapsto \quad \begin{pmatrix} \frac{1}{2} + \frac{1}{2}y \\ \frac{1}{2} + \frac{1}{2}x \end{pmatrix}$$

together with initial vector $\boldsymbol{v} = (0\ 1)^\top$. This choice of $\boldsymbol{f}$ and $\boldsymbol{v}$ leads to the sequence

$$\begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 1/2 \end{pmatrix}, \begin{pmatrix} 3/4 \\ 1 \end{pmatrix}, \cdots$$

of iterates where every pair of neighboring vectors is incomparable. □

The properties from Lemma 4.3 are all relevant; however, monotonicity and the existence of LFPs (i.e., the first two items) are particularly crucial for our purposes.

► **Example 4.4** (The LFP of a PPS) The two-dimensional PPS f given by

$$\begin{pmatrix} x \\ y \end{pmatrix} \mapsto \begin{pmatrix} \frac{1}{2}x^3 + \frac{1}{2}y \\ \frac{1}{4}y + \frac{3}{4} \end{pmatrix}$$

has the LFP

$$\mu f = \begin{pmatrix} \frac{\sqrt{5}}{2} - \frac{1}{2} \\ 1 \end{pmatrix}.$$

We can verify this as follows:

- First, the unique real solution of the equation $y = \frac{1}{4}y + \frac{3}{4}$ is $y = 1$. Thus, $(\mu f)_y = 1$.
- Second, to determine $(\mu f)_x$, we may substitute 1 for y in f_x and consider the real solutions of $x = \frac{1}{2}x^3 + \frac{1}{2}$, which are $x_1 = -\frac{\sqrt{5}}{2} - \frac{1}{2}$, $x_2 = \frac{\sqrt{5}}{2} - \frac{1}{2}$, and $x_3 = 1$. Since $x_1 < 0$ and $x_2 < x_3$, we conclude that $(\mu f)_x = x_2$.

The PPS f in this example is solvable by *radicals*; that is, its LFP μf can be expressed using square roots, cubic roots, etc., integer constants, and the four basic arithmetic operators. However, Galois theory shows that this is not possible for all PPSs. For instance, the one-dimensional PPS $x \mapsto \frac{1}{6}x^5 + \frac{1}{2}$ of degree five is not solvable by radicals [50, Theorem 3.2].[21] In general, even PPSs of degree two may not be solvable by radicals. ... ◄

[50]: Etessami et al. (2009), "Recursive Markov chains, stochastic grammars, and monotone systems of nonlinear equations".

21: A numeric approximation of the least nonnegative fixed point is $x \approx 0.50550123$.

Another property of PPSs, less relevant for our discussion but worth noting, is that they are closed under *composition*: If f and g are PPSs over the same set of variables X, then so is their functional composition $f \circ g$. This follows from the closure properties of polynomials. As a special case, the k-fold iteration f^k of a PPS f is again a PPS for all $k \in \mathbb{N}$.

4.1.3 Feasible and Clean PPSs

We shall often deal with PPSs satisfying one or both of the following properties:

Definition 4.5 ***(Feasibility and cleanness)*** Let f be a PPS.

A. f is *feasible* if $\mu f < \infty$. Thus, f has at least one componentwise nonnegative real—hence finite—fixed point.
B. f is *clean* if $\mu f > \mathbf{0}$; that is, the LFP of f is componentwise positive. △

Notice that feasibility and cleanness are dual: The former excludes the *top* element, whereas the latter excludes the *bottom* element of the underlying complete lattice $\overline{\mathbb{R}}_{\geq 0}$ from (all components of) the LFP.

► **Example 4.6** ((In)feasible and (un)clean PPSs) The PPS from Example 4.4 is both feasible and clean, as its LFP is componentwise positive and real (i. e., finite). Let us provide some examples of *infeasible* PPS:

– The prototypical example of an infeasible PPS is the one-dimensional

$$x \mapsto x + 1 \,.$$

It does not have a real fixed point—not even a negative one.[22] Thus, its LFP is ∞.[23] Indeed, the associated Kleene sequence $0, 1, 2, 3, \ldots$ has the supremum ∞.

– A second class of infeasible PPS are those that admit only partially negative fixed points (i. e., negative in at least one component). One of the simplest examples is

$$x \mapsto 2x + 1$$

with the unique real, but negative, fixed point $x = -1$. Again, the associated Kleene sequence $0, 1, 3, 7, \ldots$ diverges to ∞.

We now focus on the cleanness property from Definition 4.5. Whether a PPS is clean generally depends on the presence of nonzero *constant coefficients*:

– A one-dimensional PPS is clean if and only if its constant coefficient is nonzero. Examples include constant PPS such as $x \mapsto 42$, as well as the preceding linear (infeasible) PPSs $x \mapsto x + 1$ and $x \mapsto 2x + 1$. Examples of *unclean* one-dimensional PPSs are $x \mapsto 0$, $x \mapsto x$, and $x \mapsto x^2 + x$, all of which have zero as their constant coefficient.
– The following is an example of a three-dimensional unclean PPS:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \mapsto \begin{pmatrix} \frac{1}{2}x + y + \frac{1}{2} \\ xy \\ x + 5 \end{pmatrix}$$

The LFP is $(1 \;\; 0 \;\; 6)^\top$. Adding x, z, or any positive constant coefficient to the y-component would make this PPS clean.

– There also exist PPS that are both infeasible *and* unclean. The simplest example is

$$\begin{pmatrix} x \\ y \end{pmatrix} \mapsto \begin{pmatrix} x + 1 \\ y \end{pmatrix}$$

with LFP $(\infty \;\; 0)^\top$. ◄

22: Because such a fixed point would satisfy the equation $x = x + 1$, which does not have a solution in $\mathbb{R}$.

23: This must be true because every PPS has at least one fixed point in $\overline{\mathbb{R}}_{\geq 0}$. Notice that $\infty = \infty + 1$ indeed holds.

It is possible to *clean up* a given PPS by removing all variables the LFP assigns to zero. This technique is well-known in the literature (e. g., [50, Proposition 7.4]).[24]

Definition 4.7 ***(Cleaned-up PPS)*** Let $\tilde{f}$ be a (possibly unclean) PPS over variables $\tilde{X}$ and let

$$X = \{x \in \tilde{X} \mid (\mu \tilde{f})_x > 0\} \,.$$

[50]: Etessami et al. (2009), "Recursive Markov chains, stochastic grammars, and monotone systems of nonlinear equations".

24: Although cleanness and feasibility are dual concepts in a certain sense, I am not aware of an analogous technique for making a PPS feasible. Indeed, determining whether a given PPS is feasible is a difficult problem in general [50, Corollary 7.2].

We define the cleaned-up variant of $\tilde{f}$ as the PPS f over variables X that arises from $\tilde{f}$ by replacing all variables $\tilde{x} \in \tilde{X} \setminus X$ by 0.

More formally, assuming that $\tilde{X} = \{x_1, \ldots, x_k, \tilde{x}_{k+1}, \ldots, \tilde{x}_n\}$ where $\{x_1, \ldots, x_k\} = X$ we define, for every $x \in X$ and $v \in \overline{\mathbb{R}}_{\geq 0}^{X}$, $f_x(v) = \tilde{f}_x(v, \mathbf{0})$. △

We sometimes refer to the variables of the cleaned-up PPS as the *clean* variables of the original system. Notice that cleaning up a PPS with LFP $\mathbf{0}$ yields the empty PPS—there are no clean variables in this case.

The next lemma shows that cleaning up a PPS does not change its LFP on the remaining (i.e., clean) variables:

25: As in Definition 4.7.

Lemma 4.8 *Let $\tilde{f}$ be a (possibly unclean) PPS over variables $\tilde{X}$, and let f be its cleaned-up variant over variables $X \subseteq \tilde{X}$.*[25] *Then, for all $x \in X$,*

$$(\mu f)_x \;=\; (\mu \tilde{f})_x \;.$$

Proof. By construction of f, the Kleene sequences of $\tilde{f}$ and f coincide on all variables in X. Thus, the claim follows from Lemma 4.3E. □

There is a simple *marking algorithm* to determine the clean variables of a given PPS (i.e., the set X in Definition 4.7). We may view this algorithm as a rephrasing of the standard routine for checking emptiness of a *context-free grammar* by marking its productive nonterminals. The algorithm is best explained assuming that the PPS is written in equation system form $x_1 = f_1(x_1, \ldots, x_n), \ldots, x_n = f_n(x_1, \ldots, x_n)$:

1. Initially mark all variables x_i for which the polynomial f_i has a nonzero constant coefficient.
2. Next, mark all variables x_i for which the polynomial f_i contains a monomial consisting *only* of already-marked variables.
3. Repeat Step 2 until no new variables are marked.

Evidently, we can implement this algorithm with polynomial time complexity: It suffices to repeat step 2 at most n times. Because of this fact (and Lemma 4.8), practitioners can usually restrict attention to clean PPS. In the context of certificates, however, being able to deal with clean and unclean PPSs alike is beneficial: It relieves the certificate checker from having to clean up the PPS itself.

4.1.4 Linear PPSs

We shall occasionally consider *linear* PPSs as a special case of general PPSs. A linear PPS comprises only polynomials of degree at most one, that is, forbids multiplication of variables.[26] Therefore, linear PPSs admit a matrix–vector representation:

26: It would be logical to adopt a term such as *positive linear system*; I prefer *linear PPS* to emphasize the connection to general PPSs.

Definition 4.9 ***(Linear PPS)*** A PPS f over variables X is called *linear* if there exists a matrix $A \in \mathbb{R}_{\geq 0}^{X \times X}$ and a vector $\mathbf{b} \in \mathbb{R}_{\geq 0}^{X}$ such that f is identical to the function $x \mapsto Ax + \mathbf{b}$. △

The matrix A and the vector $\mathbf{b}$ from Definition 4.9 are unique if they exist. Moreover, it is not difficult to see that f is linear if and only if its Jacobi matrix[27] f' is a *constant* matrix, in which case we have

$$f : \quad x \mapsto f' \cdot x + f(\mathbf{0}) .$$

27: See Definition 4.14 on page 98.

In other words, if f is linear, then $A = f'$ and $\mathbf{b} = f(\mathbf{0})$.

As we would expect, linear PPSs have much stronger properties and are easier to handle than nonlinear PPSs.[28] We shall make this concrete in Section 4.3.

28: After all, linear PPS correspond to (a certain class of) *linear equation systems*.

4.2 Certificates for Positive Polynomial Systems

In this section, we present our certificates for upper and lower bounds on the LFP of a given PPS. Our basic certification mechanism relies on the well-known Knaster-Tarski Fixed Point Theorem [122]:

[122]: Tarski (1955), "A lattice-theoretical fixpoint theorem and its applications".

Lemma 4.10 (Knaster-Tarski) *Let $(L, \sqsubseteq)$ be a complete lattice*[29] *and let $f : L \to L$ be monotonic. Then f has an LFP $\mu f \in L$ given by*

$$\mu f \;=\; \inf \{x \in L \mid f(x) \sqsubseteq x\} .$$

29: That is, a partial order with all suprema and infima.

Dually, f has a greatest fixed point (GFP) *$\nu f \in L$ given by*

$$\nu f \;=\; \sup \{x \in L \mid x \sqsubseteq f(x)\} .$$

In Lemma 4.10, we call the lattice elements $x \in L$ satisfying $f(x) \sqsubseteq x$ *inductive* (or *prefixed points*). Similarly, we refer to the elements $x \in L$ such that $x \sqsubseteq f(x)$ as *coinductive* (or *postfixed points*). For us, inductive elements are far more important than coinductive ones, as our focus is on LFPs rather than on GFPs.

4.2.1 Certificates for Upper Bounds on the LFP

A direct consequence of Lemma 4.10 is that every inductive element is an *upper bound* on the LFP of f. In symbols,

$$\forall x \in L: \quad f(x) \sqsubseteq x \quad \Longrightarrow \quad \mu f \sqsubseteq x\,.$$

Our upper bound certificates for PPS rely on exactly this observation:

Proposition 4.11 (Inductive upper bounds) *Let $\boldsymbol{f}$ be a PPS over variables X and let $\boldsymbol{u} \in \overline{\mathbb{R}}^X_{\geq 0}$.*

A. *If $\boldsymbol{u}$ is inductive, that is, $\boldsymbol{f}(\boldsymbol{u}) \leq \boldsymbol{u}$, then $\mu\boldsymbol{f} \leq \boldsymbol{u}$.*
B. *If $\boldsymbol{u}$ is* strictly *inductive, that is, $\boldsymbol{f}(\boldsymbol{u}) \prec \boldsymbol{u}$, then $\mu\boldsymbol{f} \prec \boldsymbol{u}$.*

Proof. Item A follows from Lemma 4.10 (Knaster-Tarski) because $\boldsymbol{f}$ is a monotonic function on the complete lattice of extended nonnegative real vectors.[30]

30: See Lemma 4.3A on page 90.

For Item B we argue as follows:

$$\begin{aligned} & \boldsymbol{f}(\boldsymbol{u}) \prec \boldsymbol{u} \\ \Longrightarrow \quad & \boldsymbol{f}(\boldsymbol{f}(\boldsymbol{u})) \leq \boldsymbol{f}(\boldsymbol{u}) \prec \boldsymbol{u} && \text{(Monotonicity of } \boldsymbol{f}) \\ \Longrightarrow \quad & \mu\boldsymbol{f} \leq \boldsymbol{f}(\boldsymbol{u}) \prec \boldsymbol{u}\,. && (\boldsymbol{f}(\boldsymbol{u}) \text{ is inductive}) \end{aligned}$$

Thus, $\mu\boldsymbol{f} \prec \boldsymbol{u}$ as claimed. □

Because of Proposition 4.11, we shall refer to (strict) inductive vectors more descriptively as *(strict) inductive upper bounds*. Notably, inductive upper bounds are *self-certifying*—they already carry a proof of being an upper bound within themselves. To check the proof, one only has to verify the inductivity condition $\boldsymbol{f}(\boldsymbol{u}) \leq \boldsymbol{u}$.

► **Example 4.12** (Inductive upper bounds) Consider the following PPS:

$$f: \quad \begin{pmatrix} x \\ y \end{pmatrix} \mapsto \begin{pmatrix} \frac{1}{2}xy + \frac{1}{2} \\ \frac{2}{3}xy + \frac{1}{3} \end{pmatrix}$$

$x = 7/8$ and $y = 5/6$ constitute an inductive upper bound for $\boldsymbol{f}$ because

$$\begin{aligned} 0.875 \;=\; \frac{7}{8} \;\geq\; \frac{1}{2}\cdot\frac{7}{8}\cdot\frac{5}{6} + \frac{1}{2} \;\approx\; 0.865 \qquad &\text{and} \\ 0.833 \;\approx\; \frac{5}{6} \;\geq\; \frac{2}{3}\cdot\frac{7}{8}\cdot\frac{5}{6} + \frac{1}{3} \;\approx\; 0.819\,. \end{aligned}$$

Notice that this bound is even *strictly* inductive. It is not difficult to verify that the (exact) LFP of $\boldsymbol{f}$ is $x = 3/4 < 7/8$ and $y = 2/3 < 5/6$. ◄

Proposition 4.11 admits an optimization: If $\boldsymbol{u}$ is inductive, then so is $\boldsymbol{f}(\boldsymbol{u})$. Thus, we can actually infer the—possibly tighter—inequality $\mu\boldsymbol{f} \leq \boldsymbol{f}(\boldsymbol{u})$. This is sometimes useful in practice because, to check if $\boldsymbol{f}(\boldsymbol{u}) \leq \boldsymbol{u}$ holds, we have to compute $\boldsymbol{f}(\boldsymbol{u})$ anyway.

The next example shows that the reverse implications from Proposition 4.11 are false: Not *all* upper bounds on the LFP of a PPS are inductive.

► **Example 4.13** (Non-inductive upper bounds) Consider the two-dimensional linear PPS

$$\begin{pmatrix} x \\ y \end{pmatrix} \mapsto \begin{pmatrix} \frac{1}{2}y + \frac{1}{2} \\ x \end{pmatrix}$$

with LFP $(1\ 1)^\top$. Evidently, the vector $(1\ 2)^\top$ is an upper bound on the LFP. However, the bound is not inductive because

$$\begin{pmatrix} \frac{1}{2} \cdot 2 + \frac{1}{2} \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{3}{2} \\ 1 \end{pmatrix} \not\leq \begin{pmatrix} 1 \\ 2 \end{pmatrix}.$$

Being inductive is thus strictly stronger than merely being an upper bound on the LFP. ◄

The next sections (Sections 4.2.1.1 to 4.2.1.4) are devoted to the following question: *Which PPSs admit* practically useful *inductive upper bounds?* We begin by clarify what *useful* means in this context.

4.2.1.1 Degenerate and Nondegenerate PPSs

Every PPS $\boldsymbol{f}$ has at least one inductive upper bound, namely the LFP $\mu\boldsymbol{f}$ itself—it satisfies $\boldsymbol{f}(\mu\boldsymbol{f}) \leq \mu\boldsymbol{f}$ by definition. However, $\mu\boldsymbol{f}$ may be *irrational* (i. e., at least one of its components is an irrational number), even if the coefficients of $\boldsymbol{f}$ are all rational.[31] Therefore, the LFP does not necessarily serve as a practically useful certificate, since computations with irrational algebraic numbers, albeit possible, are far from trivial (see, e. g., [91] and the references therein).

Rational inductive upper bounds, on the other hand, are desirable in practice. They can be checked exactly using standard rational arithmetic. Unfortunately, not every PPS admits such bounds.[32]

> *Our goal throughout the following pages is thus to establish sufficient conditions for the existence of* rational *inductive upper bounds with arbitrary precision.*

Fortunately, we shall see that such bounds exist under a rather mild condition.

31: See, for instance, Example 4.4 on page 92.

[91]: Loup et al. (2011), "GiNaCRA: A C++ Library for Real Algebraic Computations".

32: See Proposition 4.25 on page 108 for a counter example.

We begin with some geometric intuition. Consider Figure 4.1. It shows six different two-dimensional PPSs over variables x and y. We visualize the PPSs by plotting the solution sets of the two equations $x = f_x(x, y)$ and $y = f_y(x, y)$ in the upper right quadrant of the plane.[33] For example, the plot in the top left corner is the linear PPS

$$\begin{pmatrix} x \\ y \end{pmatrix} \mapsto \begin{pmatrix} \frac{3}{4}y + \frac{1}{2} \\ \frac{1}{2}x + 1 \end{pmatrix} .$$

33: This form of visualization is standard in the literature (e.g., [43]).

The blue curve is therefore the set of points (x, y) satisfying $x = \frac{3}{4}y + \frac{1}{2}$, whereas the red curve is the set of points satisfying $y = \frac{1}{2}x + 1$. The hatched areas indicate the region of inductive upper bounds, if any.

Figure 4.1 suggests that the inductive region has a nonzero area if and only if the two curves *intersect* at the LFP and do not merely touch each other. Due to standard properties of the rational numbers, such areas always contain rational points. Furthermore, we know from elementary geometry that the curves intersect properly if and only if their tangent lines have different slopes at the LFP. To generalize this to arbitrary dimensions, it is convenient to consider instead *normal vectors* of the tangent lines (or planes, hyperplanes, etc.). It is known that a normal vector at a given point is determined by the *gradient* of the function defining the curve (or surface, or hypersurface, etc.) at that point. This suggests that, to determine whether the inductive region of a PPS $\boldsymbol{f}$ over X has a nonempty area, we should check whether the gradient vectors of the polynomials $x - \boldsymbol{f}_x$, where $x \in X$, are linearly independent at the LFP. We shall call this property *nondegeneracy* in the remainder.[34] It turns out that we can define nondegeneracy conveniently in terms of the *Jacobi matrix* of the PPS:

34: See Definition 4.15 on the next page for the formal details.

Definition 4.14 ***(Jacobi matrix)*** Let $\boldsymbol{f}$ be a PPS over variables X and let $v \in \mathbb{R}^X$. The Jacobi matrix $\boldsymbol{f}'(v)$ of $\boldsymbol{f}$ at v is the $X \times X$ matrix whose coefficients are given by the partial derivatives

$$\boldsymbol{f}'(v)_{x,y} = \partial_y \boldsymbol{f}_x(v) , \quad \text{where } x, y \in X . \qquad \triangle$$

In matrix notation, assuming $X = \{x_1, \ldots, x_n\}$ and writing ∂_i and f_i for ∂_{x_i} and $\boldsymbol{f}_{x_i}$, respectively, we thus have

$$\boldsymbol{f}'(v) = \begin{pmatrix} \partial_1 f_1(v) & \cdots & \partial_n f_1(v) \\ \vdots & \ddots & \vdots \\ \partial_1 f_n(v) & \cdots & \partial_n f_n(v) \end{pmatrix} .$$

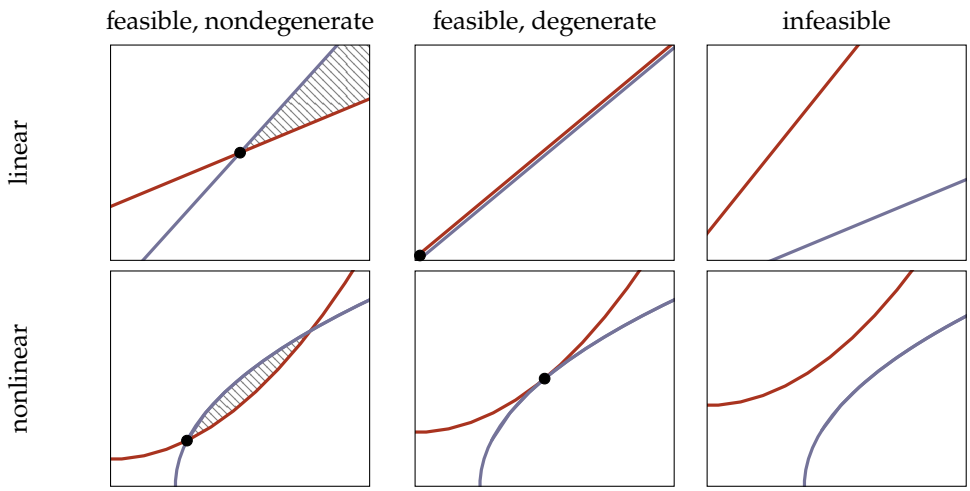


Figure 4.1: Graphical representations of six qualitatively different two-dimensional PPSs. *Legend:* Black dots: LFPs (does not apply to the 3rd column); blue curves: solutions of the x equations; red curves: solutions of the y equations; hatched areas: inductive regions (applies only to the 1st column).

Notice that the *gradient* of f_i at v is defined as the row vector

$$\nabla f_i(v) \;=\; \begin{pmatrix} \partial_1 f_i(v) & \cdots & \partial_n f_i(v) \end{pmatrix} .$$

The gradients thus constitute the rows of the Jacobi matrix. Consequently, the gradients are linearly independent at a point if and only if the Jacobi matrix is invertible at that point. We can now state the definition of nondegeneracy in terms of the Jacobi matrix:

Definition 4.15 ***(Nondegenerate PPS)*** We call a feasible PPS f *nondegenerate* if the matrix $Id - f'(\mu f)$ is nonsingular (i.e., invertible) over $\mathbb{R}$. Otherwise, f is called *degenerate*. △

Notice that we consider the matrix $Id - f'(\mu f)$ instead of $f'(\mu f)$ in Definition 4.15—after all, we are interested in the gradients of the functions $x - f_x$, where X is a variable of f, rather than in the gradients of the functions f_x.

$$\begin{pmatrix} x \\ y \end{pmatrix} \;\mapsto\; \begin{pmatrix} \frac{1}{2}xy \;+\; \frac{1}{2} \\ \frac{2}{3}xy \;+\; \frac{1}{3} \end{pmatrix}$$

The PPS f from Example 4.12 restated.

► **Example 4.16** (Nondegenerate and degenerate PPS) Reconsider the PPS f from Example 4.12 (restated in the margin). Let us prove that f is nondegenerate. To this end, we first determine its Jacobi matrix:

$$f' \;=\; \begin{pmatrix} \frac{1}{2}y & \frac{1}{2}x \\ \frac{2}{3}y & \frac{2}{3}x \end{pmatrix} .$$

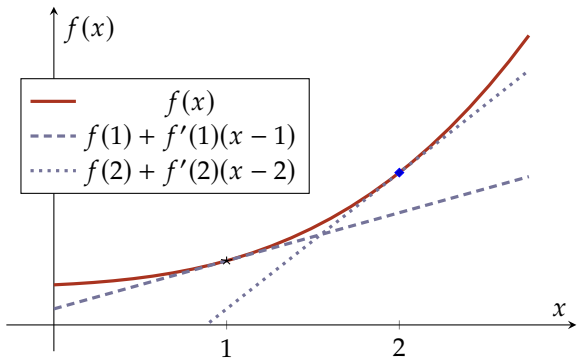


Figure 4.2: Illustration of Taylor's Theorem (Lemma 4.17) for a one-dimensional PPSs. We can see in the plot that every tangent line lower-bounds the polynomial on $\mathbb{R}_{\geq 0}$.

Using that $\mu f = \begin{pmatrix} 3/4 \\ 2/3 \end{pmatrix}$ we find that

$$Id - f'(\mu f) \;=\; \begin{pmatrix} 2/3 & 3/8 \\ 4/9 & 1/2 \end{pmatrix},$$

which is invertible.[35]

35: The determinant is $2/3 \cdot 1/2 - 4/9 \cdot 3/8 = 1/6 \neq 0$.

The simplest example of a degenerate PPS is the identity $f : x \mapsto x$; its Jacobi matrix is $f' = Id$, the (constant) identity matrix; thus, $Id - f' = 0$. Notice that the identity PPS is linear and *unclean*—its LFP is $\mathbf{0}$. We shall see in Lemma 4.31 on page 114 that every linear, feasible, and *clean* PPS is already nondegenerate. In the nonlinear case, on the other hand, feasible, clean, *and degenerate* PPS do exist; the prototypical example is

$$x \;\mapsto\; \frac{1}{2}x^2 + \frac{1}{2}$$

with LFP 1 and derivative x. ◀

In the following we shall prove formally that—as suggested by the examples in Figure 4.1—nondegenerate PPS admit arbitrarily tight rational inductive upper bounds on their LFP.[36] Our proof strategy relies on linearizing the PPS around its LFP. We therefore study a variant of Taylor's theorem for PPS next.

36: We shall see that it actually suffices that the *cleaned-up* variant of the PPS is nondegenerate (Corollary 4.24 on page 107).

4.2.1.2 Taylor's Theorem for PPSs

Informally, Taylor's theorem for differentiable functions $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$ states that

$$\mathbf{f}(\mathbf{x}) \quad \approx \quad \mathbf{f}(\mathbf{v}) \;+\; \mathbf{f}'(\mathbf{v}) \cdot (\mathbf{x} - \mathbf{v})$$

for $\mathbf{v} \approx \mathbf{x}$. This applies in particular to PPSs. More importantly to us, however, the first-order Taylor expansion of a PPS is a function *lower-bounding* the PPS in a certain sense. This phenomenon is illustrated

in Figure 4.2 for a one-dimensional PPS. We can formalize it as follows:

Lemma 4.17 (Taylor's Theorem for PPSs) *Let $\boldsymbol{f}$ be a PPS over X and let $\boldsymbol{v} \in \mathbb{R}^X_{\geq 0}$ be arbitrary. Then, for all $\boldsymbol{x} \in \mathbb{R}^X_{\geq 0}$ such that $\boldsymbol{x} \leq \boldsymbol{v}$ or $\boldsymbol{x} \geq \boldsymbol{v}$, we have*

$$\boldsymbol{f}(\boldsymbol{x}) \quad \geq \quad \boldsymbol{f}(\boldsymbol{v}) \;+\; \boldsymbol{f}'(\boldsymbol{v}) \cdot (\boldsymbol{x} - \boldsymbol{v})\,.$$

Proof. The following proof is similar to and inspired by the proof of Lemma 2.3 in [43]. It is based on the Fundamental Theorem of Calculus[37].

It suffices to show that

$$f(\boldsymbol{x}) \quad \geq \quad f(\boldsymbol{v}) \;+\; \nabla f(\boldsymbol{v}) \cdot (\boldsymbol{x} - \boldsymbol{v}) \tag{4.3}$$

for an arbitrary multivariate polynomial $f \in \mathbb{R}_{\geq 0}[X]$ with nonnegative coefficients. Note that the partial derivatives of f are again polynomials with nonnegative coefficients. The gradient ∇f is thus a monotonic function of type $\mathbb{R}^X_{\geq 0} \to \mathbb{R}^X_{\geq 0}$.[38]

For $t \in [0,1]$ define $g(t) = f(t\boldsymbol{x} + (1-t)\boldsymbol{v})$. By monotonicity of ∇f, we have:

1. If $\boldsymbol{x} \leq \boldsymbol{v}$, then $\nabla f(t\boldsymbol{x} + (1-t)\boldsymbol{v}) \leq \nabla f(\boldsymbol{v})$ and $\boldsymbol{x} - \boldsymbol{v} \leq \boldsymbol{0}$.
2. If $\boldsymbol{x} \geq \boldsymbol{v}$, then $\nabla f(t\boldsymbol{x} + (1-t)\boldsymbol{v}) \geq \nabla f(\boldsymbol{v})$ and $\boldsymbol{x} - \boldsymbol{v} \geq \boldsymbol{0}$.

In either case,

$$\nabla f(t\boldsymbol{x} + (1-t)\boldsymbol{v}) \cdot (\boldsymbol{x} - \boldsymbol{v}) \quad \geq \quad \nabla f(\boldsymbol{v}) \cdot (\boldsymbol{x} - \boldsymbol{v})\,. \tag{4.4}$$

To complete the proof, we argue as follows:

$$\begin{aligned}
& f(\boldsymbol{x}) - f(\boldsymbol{v}) \\
= \;& g(1) - g(0) && \text{(Definition of } g\text{)} \\
= \;& \int_0^1 g'(t)\,dt && \text{(Fundamental Theorem of Calculus)} \\
= \;& \int_0^1 \nabla f(t\boldsymbol{x} + (1-t)\boldsymbol{v}) \cdot (\boldsymbol{x} - \boldsymbol{v})\,dt && \text{(Chain rule)} \\
\geq \;& \int_0^1 \nabla f(\boldsymbol{v}) \cdot (\boldsymbol{x} - \boldsymbol{v})\,dt \\
& && \text{(Monotonicity of the integral and Equation (4.4))} \\
= \;& \nabla f(\boldsymbol{v}) \cdot (\boldsymbol{x} - \boldsymbol{v}) \int_0^1 dt && \text{(Linearity of integration)}
\end{aligned}$$

[43]: Esparza et al. (2010), "Computing the Least Fixed Point of Positive Polynomial Systems".

37: See, e. g., [53, §§18–19] for a proof of the Fundamental Theorem as well as for the other elementary properties of integration used further below.

38: In fact, the gradient is a PPS itself, but we shall not further exploit this fact here.

$$= \quad \nabla f(v) \cdot (x - v) \,. \qquad \left(\int_0^1 dt = 1\right)$$

Adding $f(v)$ on both sides of the resulting inequality yields Equation (4.3). □

Note that Lemma 4.17 does not hold in general if x is incomparable to v.[39]

We cannot only use Lemma 4.17 to derive lower bounds on a PPS, but also upper bounds:

Corollary 4.18 (Bounding a PPS near a point) *Let f be a PPS over variables X and let $v \in \mathbb{R}^X_{\geq 0}$.*

A. *For all $\varepsilon \in \mathbb{R}^X_{\geq 0}$ we have*[40,41]

$$f(v) + f'(v) \cdot \varepsilon \quad \leq \quad f(v + \varepsilon) \quad \leq \quad f(v) + f'(v + \varepsilon) \cdot \varepsilon \,.$$

B. *For all $\varepsilon \in \mathbb{R}^X_{\geq 0}$ with $v - \varepsilon \geq \mathbf{0}$ we similarly have*

$$f(v) - f'(v) \cdot \varepsilon \quad \leq \quad f(v - \varepsilon) \quad \leq \quad f(v) - f'(v - \varepsilon) \cdot \varepsilon \,.$$

Proof. The left inequalities in Items A and B follow from Lemma 4.17 instantiated with $x = v + \varepsilon \geq v$ and $x = v - \varepsilon \leq v$, respectively.

For the right inequalities, observe that we can restate Lemma 4.17 equivalently as follows: For all $v, x \in \mathbb{R}^X_{\geq 0}$ with $x \leq v$ or $x \geq v$,

$$f(x) \quad \leq \quad f(v) + f'(x) \cdot (x - v) \,.$$

To see this, swap the names of x and v and rearrange.[42] Setting $x = v + \varepsilon$ and $x = v - \varepsilon$ yields the rightmost inequalities in Items A and B, respectively. □

Corollary 4.18A reduces our problem of finding an inductive upper bound on a PPS f to finding a vector $\varepsilon \geq \mathbf{0}$ such that $f'(\mu f + \varepsilon) \cdot \varepsilon \leq \varepsilon$: If we have such an ε, then, by the right inequality of Corollary 4.18A,

$$f(\mu f + \varepsilon) \quad \leq \quad f(\mu f) + f'(\mu f + \varepsilon) \cdot \varepsilon \quad \leq \quad \mu f + \varepsilon \,.$$

Our goal is thus to understand when such vectors ε exist. To this end, we take a closer look at the Jacobi matrix of a PPS next.

39: A counter example is the two-dimensional PPS

$$\begin{pmatrix} x \\ y \end{pmatrix} \mapsto \begin{pmatrix} xy \\ xy \end{pmatrix}$$

together with $v = (1\ 0)^\top$ and $x = (0\ 1)^\top$. Here, we have $f(x) = (0\ 0)^\top$, but $f(v) + f'(v) \cdot (x - v) = (1\ 1)^\top$.

40: This statement is identical to Lemma 2.3 in [43].

41: Although I write ε, the vector ε need not be small for the inequalities to hold.

42: The right hand side of the resulting inequality is in general *not* a linearization, that is, a linear function in x; this shall not trouble us here, however.

4.2.1.3 Convergence of the Jacobi Matrix

Recall that a PPS $\boldsymbol{f}$ is by definition nondegenerate if the matrix $Id - \boldsymbol{f}'(\mu\boldsymbol{f})$ is invertible.[43] Throughout the following pages, we prove an alternative characterization of nondegeneracy for the case of a *clean* $\boldsymbol{f}$, namely that the latter is nondegenerate if and only if the Jacobi matrix $\boldsymbol{f}'(\mu\boldsymbol{f})$ is *convergent* in the following sense:[44]

43: See Definition 4.15 on page 99.

44: This characterization is stated as Lemma 4.22 on page 105, but we first need to introduce some preliminary notions.

Definition 4.19 ***(Convergent matrix)*** $A \in \mathbb{R}^{n\times n}$ is *convergent* if

$$\lim_{n\to\infty} A^n \;=\; 0\,,$$

where the limit is taken coefficientwise. △

Recall the following standard notions from linear algebra: An *eigenvalue* of $A \in \mathbb{R}^{n\times n}$ is a number $\lambda \in \mathbb{C}$ such that there exists a vector $\boldsymbol{v} \in \mathbb{C}^n \setminus \{\mathbf{0}\}$ satisfying

$$A\boldsymbol{v} \;=\; \lambda\boldsymbol{v}\,.$$

Every such $\boldsymbol{v}$ is an *eigenvector* corresponding to the eigenvalue λ. The *spectral radius* of A is

$$\rho(A) \;=\; \max\{\,|\lambda| \mid \lambda \text{ is an eigenvalue of } A\,\}\,,$$

where $|\cdot|$ denotes the modulus[45] of a complex number.

45: The *modulus* of a complex number is defined as $|a+ib| = \sqrt{a^2+b^2}$. It generalizes the absolute value of real numbers.

With the definition of the spectral radius at hand, we can characterize convergent matrices as follows:

Lemma 4.20 (Characterization of convergent matrices [113]) *Let $A \in \mathbb{R}^{n\times n}$. The following are equivalent:*

A. *A is convergent, that is,* $\lim_{n\to\infty} A^n = 0$.
B. $\rho(A) < 1$.
C. *The series* $\sum_{n=0}^{\infty} A^n$ *converges.*
D. *The series* $\sum_{n=0}^{\infty} A^n$ *converges to* $(Id - A)^{-1}$ *(which thus exists).*

For nonnegative $A \in \mathbb{R}^{n\times n}_{\geq 0}$, *Items* A *to* D *are moreover equivalent to:*

E. *There exists* $\boldsymbol{v} > \mathbf{0}$ *such that* $A\boldsymbol{v} < \boldsymbol{v}$.
F. $(Id - A)^{-1}$ *exists and is nonnegative.*

Intuitively, convergent matrices share the contractive properties of the real numbers with absolute value < 1, such as convergence of the associated geometric series. In fact, the one-dimensional convergent matrices are the real numbers from the open interval $(-1, 1)$.

Regarding Item F of Lemma 4.20, we note that the requirement of the inverse being nonnegative is generally necessary for the matrix to be convergent. For example, for the one-dimensional matrix $A = 2$, the inverse $(1 - 2)^{-1} = -1$ exists (and is negative), but A is not convergent. In the following, we shall see that the nonnegativity condition in Item F can be dropped in the case where $A = f'(\mu f)$ is the Jacobi matrix of a clean and feasible PPS f. In other words, we shall prove that existence of the inverse $(Id - f'(\mu f))^{-1}$—that is, nondegeneracy of f—already implies its nonnegativity.

The following lemma is implied by [43, Theorem 4.1]. We provide a compact and mostly self-contained proof below.

[43]: Esparza et al. (2010), "Computing the Least Fixed Point of Positive Polynomial Systems".

Lemma 4.21 (Bound on the spectral radius) *For every feasible and clean PPS f it holds that*

$$\rho\left(f'(\mu f)\right) \leq 1 .$$

Proof. We first show the following auxiliary statement: For all $v < \mu f$ such that $v \leq f(v)$ (i.e., v is co-inductive) the Jacobi matrix $f'(v)$ is convergent.

To prove the auxiliary statement, let $\varepsilon > \mathbf{0}$ such that $v + \varepsilon = \mu f$. By Corollary 4.18A, we have

$$f(v) + f'(v) \cdot \varepsilon \leq f(v + \varepsilon) = f(\mu f) = \mu f .$$

Further, by monotonicity of f,

$$f\left(f(v) + f'(v) \cdot \varepsilon\right) \leq \mu f .$$

We can establish a lower bound on the left hand side of the previous inequality by using Corollary 4.18A again:

$$\begin{aligned}
& f\left(f(v) + f'(v) \cdot \varepsilon\right) \\
\geq\ & f^2(v) + f'(f(v)) \cdot f'(v) \cdot \varepsilon && \text{(Corollary 4.18A)} \\
\geq\ & f^2(v) + f'(v) \cdot f'(v) \cdot \varepsilon && (f(v) \geq v \text{ by assumption}) \\
\geq\ & f^2(v) + f'(v)^2 \cdot \varepsilon
\end{aligned}$$

Iterating this argument shows that, for all $n \in \mathbb{N}$,

$$f^n(v) + f'(v)^n \cdot \varepsilon \leq \mu f .$$

Since $\lim_{n\to\infty} f^n(v) = \mu f$, it follows that $\lim_{n\to\infty} f'(v)^n = 0$ as $\varepsilon > \mathbf{0}$, i.e., $f'(v)$ is convergent as claimed.

We now construct a sequence $(v_n)_{n\in\mathbb{N}}$ of vectors such that:

1. $\lim_{n\to\infty} v_n = \mu f$
2. $v_n < \mu f$ for all $n \in \mathbb{N}$
3. $v_n \leq f(v_n)$ for all $n \in \mathbb{N}$

This is sufficient to complete the proof: By the auxiliary statement from above, $f'(v_n)$ is convergent and thus $\rho\left(f'(v_n)\right) < 1$ for all $n \in \mathbb{N}$. Since the spectral radius is continuous with respect to the matrix coefficients, we have

$$\rho\left(f'(\mu f)\right) \;=\; \rho\left(\lim_{n\to\infty} f'(v_n)\right) \;=\; \lim_{n\to\infty} \rho\left(f'(v_n)\right) \;\leq\; 1\,.$$

It remains to construct the sequence $(v_n)_{n\in\mathbb{N}}$. Notice that we cannot take the Kleene sequence of f here, since it does not necessarily satisfy condition 2.[46] Instead, we take as $(v_n)_{n\in\mathbb{N}}$ the Kleene sequence of the modified PPS

$$x \;\mapsto\; \frac{1}{2}x + \frac{1}{2}f(x)\,.$$

46: Because convergence to the LFP in some or all components may occur already after finitely many steps.

The modified PPS has exactly the same (pre-/post-)fixed points as f. Moreover, it satisfies condition 2 from above which can be shown by induction on n, exploiting the fact that f is clean for the base case $v_0 = \mathbf{0} < \mu f$. □

We can now prove the promised characterization of nondegeneracy for clean PPS:

Lemma 4.22 *Let f be a feasible and clean PPS. Then*

$$f \text{ is nondegenerate} \quad\Longleftrightarrow\quad f'(\mu f) \text{ is convergent}\,.$$

Proof. For $\Longrightarrow$ recall that nondegeneracy of f is by definition equivalent to $Id - f'(\mu f)$ being invertible. This means that 1 is *not* an eigenvalue of $f'(\mu f)$, as otherwise there would exist a nonzero vector v with $f'(\mu f)\cdot v = v$, i.e., $(Id - f'(\mu f))\cdot v = \mathbf{0}$ contradicting invertibility of $Id - f'(\mu f)$. Since the spectral radius of any nonnegative matrix is one of its eigenvalues[47], Lemma 4.21 implies that $\rho\left(f'(\mu f)\right) < 1$. Thus, $f'(\mu f)$ is convergent by Lemma 4.20.

47: This is a consequence of the Perron-Frobenius Theorem; see, for instance, [113, Section 10.3, Fact 1a].

The other direction $\Longleftarrow$ follows immediately from Lemma 4.20, which, among other things, states that every convergent matrix A has the property that $Id - A$ is invertible. □

It is worth noting that the implication $\Longrightarrow$ in Lemma 4.22 is generally false for unclean PPS. For instance, the unclean (but feasible) PPS $f = 2x$ is nondegenerate, yet $f'(\mu f) = 2$ is not convergent.

4.2.1.4 Existence of Rational Inductive Upper Bounds

We now combine the foregoing lemmas to prove one of our main results, the sought-after characterization of the existence of rational inductive upper bounds. The result is particularly natural in the case of clean PPSs; see Corollary 4.24 for the unclean case:

Theorem 4.23 (Upper bound certificates for clean PPSs) *Let f be a feasible and clean[48] PPS over variables X. The following are equivalent:*

A. *f is nondegenerate; that is, the matrix $Id - f'(\mu f)$ is invertible.*
B. *$\exists u \in \mathbb{R}^X_{>0}\colon f(u) \prec u$ (and thus $\mu f \prec u$).[49]*
C. *$\forall \varepsilon > 0\colon \exists u \in \mathbb{Q}^X_{>0}\colon f(u) \prec u$ and $u \leq \mu f + \varepsilon$ (and thus $\mu f \prec u$).*

48: See Definition 4.5 on page 92.

49: See Proposition 4.11 on page 96.

Proof. Recall that Item A (nondegeneracy) is equivalent to $f'(\mu f)$ being a convergent matrix.[50]

50: See Lemma 4.22 on the preceding page.

To prove A $\implies$ B, assume that $f'(\mu f)$ is convergent. By Lemma 4.20E there exists $v > 0$ such that $f'(\mu f)v \prec v$. Since the coefficients of the Jacobi matrix $f'(x)$ are continuous functions of x, there exists $\delta > 0$ such that

$$f'(\mu f + \delta v)v \prec v\,. \tag{4.5}$$

Define $u = \mu f + \delta v$. Now Item B follows[51] because

51: Notice that this argument crucially relies on Taylor's theorem for PPS (Lemma 4.17).

$$\begin{aligned} & f(\mu f + \delta v) \\ \leq\ & \mu f + f'(\mu f + \delta v)\cdot \delta v \\ & \qquad \text{(Corollary 4.18A, right inequality)} \\ \prec\ & \mu f + \delta v\,. \qquad \text{(By Equation (4.5))} \end{aligned}$$

We can show A $\implies$ C similarly: Let $\varepsilon > 0$ and choose δ as in Equation (4.5) and moreover sufficiently small such that $\delta v \leq \varepsilon$. Defining $u' = \mu f + \delta v$, we have $f(u') \prec u'$. Since f is a continuous function, the inequality $f(u) \prec u$ remains valid for all u in a sufficiently small neighborhood of u' (e.g., an open ball around u). Choosing any *rational* vector $u \leq u'$ from this neighborhood completes the proof, as such a u satisfies

$$u \quad \leq \quad u' \quad = \quad \mu f + \delta v \quad \leq \quad \mu f + \varepsilon\,.$$

As C $\implies$ B is trivial, it only remains to show B $\implies$ A. Let $f(u) \prec u$. By Proposition 4.11, $\mu f \prec u$. Thus, there exists $\varepsilon > 0$ such that

$\boldsymbol{u} = \mu\boldsymbol{f} + \boldsymbol{\varepsilon}$. We can now argue as follows:

$$\begin{aligned} & \mu\boldsymbol{f} + \boldsymbol{f}'(\mu\boldsymbol{f}) \cdot \boldsymbol{\varepsilon} \\ \leq\ & \boldsymbol{f}(\mu\boldsymbol{f} + \boldsymbol{\varepsilon}) \\ & \qquad \text{(Corollary 4.18A, left inequality)} \\ <\ & \mu\boldsymbol{f} + \boldsymbol{\varepsilon} \qquad\qquad \text{(As } \mu\boldsymbol{f} + \boldsymbol{\varepsilon} = \boldsymbol{u}\text{)} \\ \implies \boldsymbol{f}'(\mu\boldsymbol{f}) \cdot \boldsymbol{\varepsilon}\ <\ & \boldsymbol{\varepsilon}\,. \end{aligned}$$

Since $\boldsymbol{\varepsilon} > \mathbf{0}$, it follows that $\boldsymbol{f}'(\mu\boldsymbol{f})$ is convergent by Lemma 4.20. □

Let us illustrate the meaning of Theorem 4.23 by means of a concrete instance. In Example 4.12 we saw a feasible and clean PPS for which we verified a strictly inductive upper bound; subsequently, we proved nondegeneracy of the same PPS in Example 4.16. We now know, by Theorem 4.23, that nondegeneracy was in fact already implied by the existence of the strictly inductive upper bound that we found. Conversely, had we first established nondegeneracy, we could have directly concluded the existence of such a bound—even though we would still have to find one. We shall discuss an algorithm for computing inductive upper bounds for nondegenerate PPSs in Chapter 5.

We can apply Theorem 4.23 to obtain a result for *unclean* PPSs as well:

Corollary 4.24 (Upper bound certificates for general PPSs) *Let $\tilde{\boldsymbol{f}}$ be a feasible PPS over variables $\tilde{X}$. If the cleaned-up variant*[52] *$\boldsymbol{f}$ of $\tilde{\boldsymbol{f}}$ is nondegenerate, then, for all $\boldsymbol{\varepsilon} > \mathbf{0}$,*

52: See Definition 4.7 on page 93.

$$\exists \boldsymbol{u} \in \mathbb{Q}_{\geq 0}^{\tilde{X}} \colon \quad \underbrace{\tilde{\boldsymbol{f}}(\boldsymbol{u}) \leq \boldsymbol{u}}_{\text{Thus, } \mu\tilde{\boldsymbol{f}} \leq \boldsymbol{u}.} \quad \text{and} \quad \boldsymbol{u} \leq \mu\tilde{\boldsymbol{f}} + \boldsymbol{\varepsilon}\,;$$

that is, $\tilde{\boldsymbol{f}}$ admits arbitrarily precise rational *inductive upper bounds.*

53: See Definition 4.7 on page 93.

Proof. The idea is to take an ε-precise rational inductive upper bound for the cleaned-up PPS[53] and pad it with 0's for the unclean variables.

Formally, by Theorem 4.23, the nondegenerate and clean PPS $\boldsymbol{f}$ over the clean variables X admits ε-precise strictly inductive upper bounds, that is, for all $\boldsymbol{\varepsilon} > \mathbf{0}$ there exists $\boldsymbol{u}' \in \mathbb{Q}_{>0}^{X}$ such that $\boldsymbol{f}(\boldsymbol{u}') < \boldsymbol{u}'$

and $\boldsymbol{u}' \leq \mu\boldsymbol{f} + \boldsymbol{\varepsilon}$. Define the vector $\boldsymbol{u} \in \mathbb{Q}_{\geq 0}^{\tilde{X}}$ such that, for all $x \in \tilde{X}$,

$$\boldsymbol{u}(x) \;=\; \begin{cases} \boldsymbol{u}'(x) & \text{if } x \in X \\ 0 & \text{if } x \notin X \end{cases}.$$

It is easy to verify that $\tilde{\boldsymbol{f}}(\boldsymbol{u}) \leq \boldsymbol{u}$ and $\boldsymbol{u} \leq \mu\tilde{\boldsymbol{f}} + \boldsymbol{\varepsilon}$, as desired. □

Finally, we show that Corollary 4.24 becomes false in general if we drop the nondegeneracy assumption, even for PPSs with rational coefficients:

Proposition 4.25 *There exists a feasible and clean PPS with* rational *coefficients which does not have any rational inductive upper bounds (and is thus necessarily degenerate by Theorem 4.23).*

Proof. Consider the following PPS $\boldsymbol{f}$ over variables x, y, z:[54]

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \mapsto \begin{pmatrix} x^2 + \frac{1}{4}y^2 + xz \\ \frac{1}{4}y^2 + \frac{1}{2} \\ \frac{1}{4}yz + \frac{1}{4}z + \frac{1}{4} \end{pmatrix}$$

It can be verified[55] that $\boldsymbol{f}$ has two fixed points, the least of which is

$$\mu\boldsymbol{f} \;=\; \begin{pmatrix} 1 - \frac{\sqrt{2}}{2} \\ 2 - \sqrt{2} \\ \sqrt{2} - 1 \end{pmatrix}.$$

Thus, $\boldsymbol{f}$ is feasible and clean.

We now show that $\boldsymbol{f}$ has no rational inductive upper bounds. Toward contradiction, assume that there exists $\boldsymbol{u} = \begin{pmatrix} x & y & z \end{pmatrix}^\top \in \mathbb{Q}_{\geq 0}^3$ such that $\boldsymbol{u} \geq \boldsymbol{f}(\boldsymbol{u})$. Since $\boldsymbol{u} \geq \mu\boldsymbol{f}$ and $\mu\boldsymbol{f}$ is irrational, we must have $\boldsymbol{u} > \mu\boldsymbol{f}$, and thus in particular $y > 2 - \sqrt{2}$ and $z > \sqrt{2} - 1$. Let us now inspect the x-component of the inequality $\boldsymbol{u} \geq \boldsymbol{f}(\boldsymbol{u})$:

$$\begin{aligned} x \;&\geq\; x^2 + \frac{1}{4}y^2 + xz \\ &>\; x^2 + \frac{1}{4}(2 - \sqrt{2})^2 + x(\sqrt{2} - 1) \\ \iff \quad 0 \;&>\; 4x^2 + (2 - \sqrt{2})^2 + 4(\sqrt{2} - 2)x \\ &\qquad\qquad \text{(Subtract } x\text{, then multiply by 4)} \\ &=\; (2x + \sqrt{2} - 2)^2 \end{aligned}$$

54: This PPS was carefully designed so that it has the desired property. In my experience, such examples are quite rare.

55: By hand, or using a computer algebra system. I used Wolfram Mathematica.

This is a contradiction since the inequality $(2x + \sqrt{2} - 2)^2 < 0$ has no real solutions. □

Theorem 4.23 implies that the PPS $\boldsymbol{f}$ from the proof of Proposition 4.25 is degenerate. It is instructive to verify this directly as well: The Jacobi matrix of $\boldsymbol{f}$ is

$$\boldsymbol{f}'(x,y,z) \;=\; \begin{pmatrix} 2x+z & \frac{1}{2}y & x \\ 0 & \frac{1}{2}y & 0 \\ 0 & \frac{1}{4}z & \frac{1}{4}y + \frac{1}{4} \end{pmatrix}.$$

We have to show that the matrix $Id - \boldsymbol{f}'(\mu\boldsymbol{f})$ is singular (i.e., not invertible). To see this, it suffices to inspect the leftmost column of $Id - \boldsymbol{f}'(\mu\boldsymbol{f})$: The topmost entry in this column is

$$1 - \left(2\left(1 - \frac{\sqrt{2}}{2}\right) + \sqrt{2} - 1\right) \;=\; 0\,.$$

The other two entries are also 0. Thus, the leftmost column of $Id - \boldsymbol{f}'(\mu\boldsymbol{f})$ is 0 everywhere, showing that $Id - \boldsymbol{f}'(\mu\boldsymbol{f})$ is not invertible.

On the other hand, there also exist *degenerate* PPS with a rational LFP. The simplest example is $x \mapsto \frac{1}{2}x^2 + \frac{1}{2}$, which has LFP 1. Therefore, Corollary 4.24 really provides just a *sufficient* condition for the existence of (tight) rational inductive upper bounds. A complete characterization would be even more desirable, but is left it for future work as it is not as crucial for our applications. Indeed, we shall see in Section 4.3 that inductive upper bounds for nondegenerate PPSs suffice for constructing a *complete* certification mechanism for PAST (positive almost sure termination) of pPDA.

▷ **Remark 4.26** (On the complexity of checking certificates) Assume that rational numbers are encoded as fractions of integers in binary. Then, for any given rational PPS $\boldsymbol{f}$ with *unary-encoded exponents*[56] and rational vector $\boldsymbol{u}$ of suitable dimension, we can perform the inductivity check

$$\boldsymbol{f}(\boldsymbol{u}) \overset{?}{\leq} \boldsymbol{u}$$

in polynomial time in the size of $\boldsymbol{f}$ plus the size of $\boldsymbol{u}$.[57] This is because evaluating $\boldsymbol{f}(\boldsymbol{u})$ requires at most linearly many (in the size of $\boldsymbol{f}$) multiplications and additions that involve the rational entries of $\boldsymbol{u}$ and the coefficients of $\boldsymbol{f}$. It follows that the entries of $\boldsymbol{f}(\boldsymbol{u})$ have polynomial size. Thus, we can evaluate the inductivity check in polynomial time.

Note that this argument does not work if the exponents in $\boldsymbol{f}$ are encoded in binary. In this case, evaluating $\boldsymbol{f}(\boldsymbol{u})$ may yield numbers of exponential bit size (and thus of doubly-exponential magnitude). For example, if we allow

56: Alternatively, with polynomials of fixed maximum degree.

57: We can understand this as the size of the input given to a certificate checker.

binary-encoded exponents, then the size of the PPS $f : x \mapsto x^{2^n}$ is only linear in n (as $\log_2 2^n = n$), yet $f(2) = 2^{2^n}$ has exponential bit size in n. ◁

4.2.2 Certificates for Lower Bounds on the LFP

Certifying lower bounds on the LFP of a PPS is more involved than certifying upper bounds. In particular, we cannot simply dualize the arguments from the previous section. The asymmetry between upper and lower bounds stems from the fact that the Knaster-Tarski Theorem[58] only provides a certification mechanism for lower bounds on the *greatest* fixed point νf of a PPS f: For all nonnegative vectors v, the theorem assures that

$$v \leq f(v) \quad \Longrightarrow \quad v \leq \nu f \,.$$

58: See Lemma 4.10 on page 95.

Since $\mu f < \nu f$ in general, a vector v merely satisfying $v \leq f(v)$—that is, a *coinductive* vector—is not necessarily a lower bound on μf.[59]

59: For instance, take v as the GFP of any PPS f with a non-unique fixed point.

To overcome this issue, we have to impose additional conditions besides coinductivity. Concretely, our approach is to certify a lower bound together with a (strictly) inductive *upper bound*; that is, we certify a two-sided bound.[60] We shall see that this method relies crucially on the following lemma:

60: I am not aware of an approach that would certify one-sided lower bounds only. However, in practice, two-sided bounds are often desirable anyway.

Lemma 4.27 (Iterating a strictly inductive upper bound) *Let f be a PPS with a finite strictly inductive upper bound u, that is, $f(u) < u < \infty$. Then*

$$\lim_{n\to\infty} f^n(u) \;=\; \inf_{n\in\mathbb{N}} f^n(u) \;=\; \mu f \,,$$

where the limit and the infimum are taken componentwise.

Proof. The limit exists and is equal to the infimum because, by monotonicty of f,

$$u \;>\; f(u) \;\geq\; f^2(u) \;\geq\; \ldots$$

is a monotonic sequence bounded from below by $\mathbf{0}$.

In the remainder of the proof, we write $\tilde{u} = \lim_{n\to\infty} f^n(u) = \inf_{n\in\mathbb{N}} f^n(u)$. Thus, the goal is to show that $\tilde{u} = \mu f$. Notice that $\tilde{u} < u$.

First, $\tilde{\boldsymbol{u}}$ is a fixed point of $\boldsymbol{f}$ because, by continuity of $\boldsymbol{f}$,

$$\boldsymbol{f}(\tilde{\boldsymbol{u}}) \;=\; \boldsymbol{f}\left(\lim_{n\to\infty} \boldsymbol{f}^n(\boldsymbol{u})\right) \;=\; \lim_{n\to\infty} \boldsymbol{f}^{n+1}(\boldsymbol{u}) \;=\; \lim_{n\to\infty} \boldsymbol{f}^n(\boldsymbol{u}) \;=\; \tilde{\boldsymbol{u}}\,.$$

It only remains to show that $\tilde{\boldsymbol{u}}$ is the *least* fixed point of $\boldsymbol{f}$. To this end, we first show, using Taylor's Theorem, that the matrix $\boldsymbol{f}'(\tilde{\boldsymbol{u}})$ is convergent[61]. Indeed, we have

61: See Definition 4.19 on page 103.

$$\begin{aligned}
\tilde{\boldsymbol{u}} \;&=\; \boldsymbol{f}(\tilde{\boldsymbol{u}}) && (\tilde{\boldsymbol{u}} \text{ is a fixed point of } \boldsymbol{f}\text{; see above})\\
&=\; \boldsymbol{f}(\boldsymbol{u} - (\boldsymbol{u} - \tilde{\boldsymbol{u}}))\\
&\leq\; \boldsymbol{f}(\boldsymbol{u}) - \boldsymbol{f}'(\tilde{\boldsymbol{u}})(\boldsymbol{u} - \tilde{\boldsymbol{u}}) && \text{(Corollary 4.18B)}\\
&<\; \boldsymbol{u} - \boldsymbol{f}'(\tilde{\boldsymbol{u}})(\boldsymbol{u} - \tilde{\boldsymbol{u}}) && (\boldsymbol{u} \text{ is strictly inductive})
\end{aligned}$$

so that $\boldsymbol{f}'(\tilde{\boldsymbol{u}})(\boldsymbol{u} - \tilde{\boldsymbol{u}}) < \boldsymbol{u} - \tilde{\boldsymbol{u}}$. Since $\boldsymbol{f}'(\tilde{\boldsymbol{u}})$ is a nonnegative matrix and $\boldsymbol{u} - \tilde{\boldsymbol{u}} > \boldsymbol{0}$, it follows that $\boldsymbol{f}'(\tilde{\boldsymbol{u}})$ is convergent by Lemma 4.20.

Applying Taylor's Theorem again, we also have

$$\begin{aligned}
\tilde{\boldsymbol{u}} \;&=\; \boldsymbol{f}(\tilde{\boldsymbol{u}})\\
&=\; \boldsymbol{f}(\mu\boldsymbol{f} + (\tilde{\boldsymbol{u}} - \mu\boldsymbol{f}))\\
&\leq\; \mu\boldsymbol{f} + \boldsymbol{f}'(\tilde{\boldsymbol{u}})(\tilde{\boldsymbol{u}} - \mu\boldsymbol{f}) && \text{(Corollary 4.18A)}
\end{aligned}$$

and thus $\tilde{\boldsymbol{u}} - \mu\boldsymbol{f} \leq \boldsymbol{f}'(\tilde{\boldsymbol{u}})(\tilde{\boldsymbol{u}} - \mu\boldsymbol{f})$. Since multiplication by nonnegative matrices is monotonic, we obtain the inequalities

$$\begin{aligned}
\boldsymbol{f}'(\tilde{\boldsymbol{u}})(\tilde{\boldsymbol{u}} - \mu\boldsymbol{f}) \;&\leq\; \boldsymbol{f}'(\tilde{\boldsymbol{u}})^2(\tilde{\boldsymbol{u}} - \mu\boldsymbol{f})\,,\\
\boldsymbol{f}'(\tilde{\boldsymbol{u}})^2(\tilde{\boldsymbol{u}} - \mu\boldsymbol{f}) \;&\leq\; \boldsymbol{f}'(\tilde{\boldsymbol{u}})^3(\tilde{\boldsymbol{u}} - \mu\boldsymbol{f})\,, \quad \text{etc.}
\end{aligned}$$

It follows that, for all $n \in \mathbb{N}$,

$$\tilde{\boldsymbol{u}} - \mu\boldsymbol{f} \;\leq\; \boldsymbol{f}'(\tilde{\boldsymbol{u}})^n(\tilde{\boldsymbol{u}} - \mu\boldsymbol{f})\,,$$

which is equivalent to

$$\tilde{\boldsymbol{u}} - \mu\boldsymbol{f} \;\leq\; \inf_{n\in\mathbb{N}} \boldsymbol{f}'(\tilde{\boldsymbol{u}})^n(\tilde{\boldsymbol{u}} - \mu\boldsymbol{f})\,.$$

However, since $\boldsymbol{f}'(\tilde{\boldsymbol{u}})$ is convergent, $\lim_{n\to\infty} \boldsymbol{f}'(\tilde{\boldsymbol{u}})^n$ exists and is equal to 0. Thus

$$\tilde{\boldsymbol{u}} - \mu\boldsymbol{f} \;\leq\; \inf_{n\in\mathbb{N}} \boldsymbol{f}'(\tilde{\boldsymbol{u}})^n(\tilde{\boldsymbol{u}} - \mu\boldsymbol{f}) \;\leq\; \lim_{n\to\infty} \boldsymbol{f}'(\tilde{\boldsymbol{u}})^n(\tilde{\boldsymbol{u}} - \mu\boldsymbol{f}) \;=\; \boldsymbol{0}\,,$$

that is, $\tilde{\boldsymbol{u}} \leq \mu\boldsymbol{f}$. Since $\tilde{\boldsymbol{u}}$ is a fixed point of $\boldsymbol{f}$, it follows that $\tilde{\boldsymbol{u}} = \mu\boldsymbol{f}$. □

Lemma 4.27 is a peculiar property of PPS. It does not hold for general

monotonic functions of type $\overline{\mathbb{R}}_{\geq 0}^n \to \overline{\mathbb{R}}_{\geq 0}^n$, not even for continuous ones:

► **Example 4.28** (Counter example to Lemma 4.27 for non-PPS functions) The piecewise linear function $f : \overline{\mathbb{R}}_{\geq 0} \to \overline{\mathbb{R}}_{\geq 0}$ defined as

$$f(x) = \begin{cases} 0 & \text{if } x \in [0,1) \\ 2x - 2 & \text{if } x \in [1,3) \\ \frac{1}{2}x + \frac{5}{2} & \text{if } x \geq 3 \end{cases}$$

is monotonic with least fixed point $\mu f = 2$. Moreover, $u = 6$ is a strict inductive upper bound as $f(u) = \frac{11}{2} < u$. However, iterating f on $u = 6$ converges to the fixed point 5, that is, *not* to the LFP $\mu f = 2$. ..... ◄

We can now state our certification mechanism for lower bounds on the LFP of a PPS:

Theorem 4.29 (Lower bound certificates for PPSs) *Let $\boldsymbol{f}$ be a PPS and let $\mathbf{0} \leq \boldsymbol{\ell}, \boldsymbol{u} < \boldsymbol{\infty}$ be vectors satisfying the following three conditions:*

1. $\boldsymbol{\ell} \leq \boldsymbol{u}$.
2. $\boldsymbol{\ell} \leq \boldsymbol{f}(\boldsymbol{\ell})$, *that is,* $\boldsymbol{\ell}$ *is coinductive.*
3. $\boldsymbol{f}(\boldsymbol{u}) < \boldsymbol{u}$, *that is,* $\boldsymbol{u}$ *is strictly inductive.*[62]

Then it holds that

$$\boldsymbol{\ell} \leq \mu \boldsymbol{f} < \boldsymbol{u} .$$

62: Any $\boldsymbol{u}$ satifying this condition witnesses that $\boldsymbol{f}$ is feasible; moreover, if $\boldsymbol{f}$ is clean, then the existence of such a $\boldsymbol{u}$ is equivalent to $\boldsymbol{f}$ being feasible and nondegenerate by Theorem 4.23 on page 106.

Proof. Using conditions 1 and 2 as well as monotonicity of $\boldsymbol{f}$, we find that, for all $n \in \mathbb{N}$,

$$\boldsymbol{\ell} \leq \boldsymbol{f}(\boldsymbol{\ell}) \leq \boldsymbol{f}^2(\boldsymbol{\ell}) \leq \ldots \leq \boldsymbol{f}^n(\boldsymbol{\ell}) \leq \boldsymbol{f}^n(\boldsymbol{u})$$

and thus

$$\boldsymbol{\ell} \leq \inf_{n \in \mathbb{N}} \boldsymbol{f}^n(\boldsymbol{u}) .$$

But since $\boldsymbol{u}$ is strictly inductive by assumption 3, it follows that $\inf_{n \in \mathbb{N}} \boldsymbol{f}^n(\boldsymbol{u}) = \mu \boldsymbol{f}$ by Lemma 4.27. □

63: This remark applies similarly to Lemma 4.27.

Note that the *strict* inductivity condition on $\boldsymbol{u}$ in Theorem 4.29 is crucial. Otherwise, we could simply choose $\boldsymbol{\ell} = \boldsymbol{u} = \nu \boldsymbol{f}$, which is not a valid lower bound in general.[63]

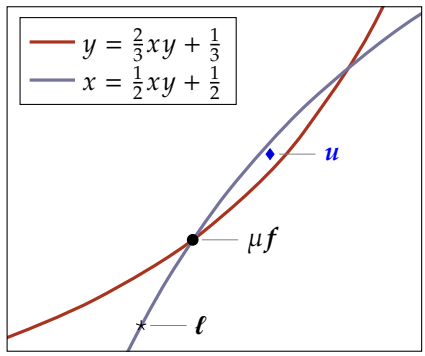


Figure 4.3: Visualization of the PPS f from Example 4.30. We certify the (coinductive) lower bound ℓ together with a strictly inductive upper bound u as suggested by Theorem 4.29.

► **Example 4.30** (Certifying a lower bound) Reconsider the PPS f from Example 4.12 on page 96:

$$\begin{pmatrix} x \\ y \end{pmatrix} \mapsto \begin{pmatrix} \frac{1}{2}xy + \frac{1}{2} \\ \frac{2}{3}xy + \frac{1}{3} \end{pmatrix}$$

We already verified that $u = (7/8 \;\; 5/6)^\top$ is a strictly inductive upper bound for f. We claim that $\ell = (2/3 \;\; 1/2)^\top \leq u$ is a lower bound. By Theorem 4.29, it suffices to check that ℓ is coinductive. The latter is indeed true, because

$$\begin{array}{rcccl} \frac{2}{3} & \leq & \frac{1}{2}\cdot\frac{2}{3}\cdot\frac{1}{2} + \frac{1}{2} & = & \frac{2}{3} \\ \frac{1}{2} & \leq & \frac{2}{3}\cdot\frac{2}{3}\cdot\frac{1}{2} + \frac{1}{3} & = & \frac{5}{9}. \end{array} \quad \text{and}$$

Thus, the LFP μf is sandwiched between ℓ and u:

$$\ell = \begin{pmatrix} 2/3 \\ 1/2 \end{pmatrix} \leq \mu f = \begin{pmatrix} 3/4 \\ 2/3 \end{pmatrix} < u = \begin{pmatrix} 7/8 \\ 5/6 \end{pmatrix}.$$

We may visualize this situation as in Figure 4.3 Note that the verification of these bounds only involves basic rational number arithmetic: Addition, multiplication, and comparison. ◄

To conclude this section, we remark that arbitrarily tight *rational* coinductive lower bounds $\ell \leq f(\ell) \leq \mu f$ on the LFP of a feasible PPS f with rational coefficients always exist, even for degenerate PPSs.[64] This is an immediate consequence of the fact that the Kleene sequence

$$\mathbf{0} \leq f(\mathbf{0}) \leq f^2(\mathbf{0}) \leq \ldots$$

only comprises coinductive rational vectors and converges to μf.

64: This is in contrast to inductive *upper* bounds; see Proposition 4.25 on page 108.

4.3 Certificates for Probabilistic Pushdown Automata

In this section, we study two certification problems for pPDA:[65] for bounds on the *return probabilities*, and, as an extension, for PAST and bounds on the expected runtime. The essence of these certificates is to prove suitable bounds on the LFPs of the pPDA's fundamental PPSs.[66] Regarding the existence of such certificates (i.e., completeness), another crucial ingredient is our characterization of cPAST from Chapter 3.[67] In particular, we shall see that pPDA that are (universally) cPAST give rise to nondegenerate PPSs, for which our certification mechanisms from Section 4.2 are complete.

65: Refer to Section 3.2 on page 71 for the relevant material on pPDA.

66: See Theorem 3.11 on page 58.

67: See Theorem 3.38 on page 82.

4.3.1 Certificates for Bounds on the Return Probabilities

In Chapter 3 we saw that, besides the fundamental PPS of a pPDA, also several *linear* equation systems—linear PPSs, to be precise—are relevant in connection with pPDA. We therefore start with a useful lemma on linear PPSs.[68]

68: We defined linear PPS in Section 4.1.4 on page 95.

Lemma 4.31 (Feasibility of clean linear PPSs) *Let $f : \boldsymbol{x} \mapsto A\boldsymbol{x} + \mathbf{b}$ be a clean*[69] *linear PPS. The following are equivalent:*

69: f clean implies $\mathbf{b} \neq \mathbf{0}$.

A. *f is feasible, that is, it has a finite nonnegative fixed point.*
B. *f has a* unique *finite nonnegative fixed point.*
C. *A is convergent, that is,* $\lim_{n\to\infty} A^n = 0$.[70]

70: This is moreover equivalent to the other 5 conditions from Lemma 4.20.

Proof. The most important implication is A $\implies$ C. In the proof of Lemma 4.21 on page 104, we showed that, in feasible PPSs, every $v \prec \mu f$ satisfying $v \leq f(v)$ has the property that $f'(v)$ is convergent.[71] Since $f'(v) = A$ is a constant matrix for our linear PPS f, the claim follows choosing $v = \mathbf{0}$: We trivially have $\mathbf{0} \leq f(\mathbf{0})$ and $\mathbf{0} \prec \mu f$ because f is clean by assumption.

71: In fact, the condition $v \prec \mu f$ already suffices for $f'(v)$ to be convergent; see [43, Theorem 4.1].

For C $\implies$ B it suffices to note that since A is nonnegative and convergent by assumption, the matrix $Id - A$ has a nonnegative inverse by Lemma 4.20. The equation system $\boldsymbol{x} = A\boldsymbol{x} + \mathbf{b}$ thus has the unique solution $\boldsymbol{x} = (Id - A)^{-1}\mathbf{b}$ in the nonnegative reals. □

Lemma 4.31 implies that degeneracy—which may arise in nonlinear PPSs—is not an issue for linear PPSs: Every clean linear PPS is already nondegenerate (provided it is feasible).

Notice that if f in Lemma 4.31 is unclean, then Item C still implies the other two conditions, and Item B trivially implies Item A. However, the remaining implications fail in general for unclean linear PPSs.[72]

Since we are interested in the return probabilities of pPDA—the triples $[qZr]$ for states q, r and a stack symbol Z—we now consider the fundamental PPS of a pPDA. Recall that the LFP of this PPS comprises the pPDA's return probabilities.[73] The following lemma shows that feasibility is guaranteed for this class of PPSs:

Lemma 4.32 *The fundamental PPS f_Δ of any pPDA Δ is feasible with $\mu f_\Delta \leq \mathbf{1}$.*

Proof. For all heads $qZ \in Q \times \Gamma$, the sum $\sum_{r \in Q} [qZr]$ is at most the termination probability of Δ starting in configuration qZ.[74] The termination probabilities are indeed probabilities (i.e., real numbers between zero and one) as shown in Lemma 2.41 on page 37. It follows that each return probability triple is at most 1, and thus $\mu f_\Delta \leq \mathbf{1}$. □

The fundamental PPS of a pPDA is is, however, not necessarily clean; see Example 4.35 on the following page.

We can now state and prove one of our main results for pPDA:

Theorem 4.33 (Nondegeneracy is equivalent to cPAST) *Let Δ be a pPDA with cleaned-up[75] fundamental PPS f_Δ. Then it holds that*

$$f_\Delta \text{ is nondegenerate} \iff \Delta \text{ is universally cPAST}.$$

Proof. Let $\tilde{f}_\Delta$ be the (possibly unclean) fundamental PPS of Δ, and let f_Δ its cleaned-up version. Recall that $\tilde{f}_\Delta$ has the return probability triples $[qZr]$ of Δ as its LFP[76], whereas the LFP of f_Δ comprises only the triples with *nonzero* probability.

To prove the theorem, we first note that the *linear* PPS

$$\boldsymbol{x} \mapsto \mu\tilde{f}_\Delta + \tilde{f}'_\Delta(\mu\tilde{f}_\Delta) \cdot \boldsymbol{x}$$

which encodes the return *moment* triples $[qZr]^{\langle 1 \rangle}$ of Δ as its LFP[77], is also possibly unclean. Its cleaned-up version is precisely the linear

72: A counter example for the implications from A to C and from B to C is $x \mapsto 2x$. A counter example for the implication from A to B is the identity PPS.

73: See Example 3.13 on page 59 and Example 4.35 on the next page for examples.

74: See Corollary 3.26 on page 73. For proper pPDA, the sum is equal to the termination probability.

75: See Definition 4.7 on page 93.

76: See Theorem 3.11 on page 58.

77: See Theorem 3.19 on page 65.

PPS $\boldsymbol{x} \mapsto \mu\boldsymbol{f}_\Delta + \boldsymbol{f}'_\Delta(\mu\boldsymbol{f}_\Delta)\cdot\boldsymbol{x}$, because $[qZ\,r] = 0$ iff $[qZ\,r]^{\langle 1\rangle} = 0$, as shown in Lemma 2.34 on page 33. We can now argue as follows:

$$
\begin{aligned}
& \Delta \text{ is universally cPAST} \\
\iff\ & \text{All return moments } [qZ\,r]^{\langle 1\rangle} \text{ of } \Delta \text{ are finite.} \\
& \qquad\qquad \text{(Theorem 3.38 on page 82)} \\
\iff\ & \boldsymbol{x} \mapsto \mu\tilde{\boldsymbol{f}}_\Delta + \tilde{\boldsymbol{f}}'_\Delta(\mu\tilde{\boldsymbol{f}}_\Delta)\cdot\boldsymbol{x} \text{ is feasible.} \\
& \qquad\qquad \text{(Theorem 3.19 on page 65)} \\
\iff\ & \boldsymbol{x} \mapsto \mu\boldsymbol{f}_\Delta + \boldsymbol{f}'_\Delta(\mu\boldsymbol{f}_\Delta)\cdot\boldsymbol{x} \text{ is feasible.} \\
\iff\ & \text{The matrix } \boldsymbol{f}'_\Delta(\mu\boldsymbol{f}_\Delta) \text{ is convergent.} \\
& \quad \text{(Lemma 4.31, using that } \boldsymbol{x} \mapsto \mu\boldsymbol{f}_\Delta + \boldsymbol{f}'_\Delta(\mu\boldsymbol{f}_\Delta)\cdot\boldsymbol{x} \text{ is clean)} \\
\iff\ & \boldsymbol{f}_\Delta \text{ is nondegenerate.} \\
& \qquad\qquad \text{(Lemma 4.22, using that } \boldsymbol{f}_\Delta \text{ is clean)}
\end{aligned}
$$

This completes the proof. □

The geometric interpretation of Theorem 4.33 is that the curves, surfaces, or hypersurfaces defined by the equations of the cleaned-up fundamental PPS of a pPDA intersect properly (i.e., do not merely touch each other) pairwise at the LFP if and only if the pPDA is universally cPAST.[78]

78: See Figure 4.1 on page 99 and the accompanying discussion for details on the geometric interpretation.

An immediate practical implication of Theorem 4.33 and the results from Section 4.2 is that every universally cPAST pPDA Δ admits easy-to-check certificates for bounds on its return probability triples:

Corollary 4.34 (Certificates for upper bounds on return triples) *Let the pPDA $\Delta = (Q, \Gamma, P)$ be universally cPAST[79] and let $\boldsymbol{f}_\Delta$ be the fundamental PPS[80] of Δ. Then there exist arbitrarily tight rational inductive upper bounds on the return probabilities $\mu\boldsymbol{f}_\Delta$ of Δ, that is, for all $\boldsymbol{\varepsilon} > \boldsymbol{0}$,*

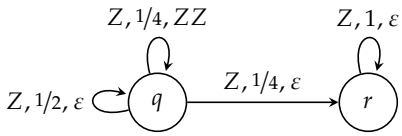
$$\exists \boldsymbol{u} \in \mathbb{Q}_{\geq 0}^{Q\times\Gamma\times Q}\colon \quad \underbrace{\boldsymbol{f}_\Delta(\boldsymbol{u}) \leq \boldsymbol{u}}_{\text{Thus, } \mu\boldsymbol{f}_\Delta \leq \boldsymbol{u}.} \quad \text{and} \quad \boldsymbol{u} \leq \mu\boldsymbol{f}_\Delta + \boldsymbol{\varepsilon}\,.$$

79: See Definition 3.35 on page 80.

80: See Theorem 3.11 on page 58.

Proof. The statement is a direct application of Theorem 4.33 and Corollary 4.24. Notice that $\boldsymbol{f}_\Delta$ is allowed to be unclean. □

► **Example 4.35** (Certifying upper bounds on return probabilities) Consider the following (proper) pPDA Δ with states q and r and stack alphabet $\Gamma = \{Z\}$:



The (unclean) fundamental PPS f_Δ of Δ is given by

$$\begin{pmatrix} [qZq] \\ [qZr] \\ [rZq] \\ [rZr] \end{pmatrix} \mapsto \begin{pmatrix} \frac{1}{4}[qZq]^2 + \frac{1}{2} \\ \frac{1}{4}[qZq][qZr] + \frac{1}{4}[qZr][rZr] + \frac{1}{4} \\ 0 \\ 1 \end{pmatrix}.$$

It is intuitively clear that Δ is universally cPAST (in fact, universally PAST) because from q there is a positive probability of entering r where the automaton empties its stack completely.[81] By Corollary 4.34, we thus know that f_Δ admits rational inductive upper bounds of arbitrary precision. For instance, we can verify that

81: Universal PAST will be proved formally in Example 4.37 on page 119.

$$\boldsymbol{u} = \begin{pmatrix} 3/5 & 1/2 & 0 & 1 \end{pmatrix}^\top$$

is inductive:

$$f_\Delta(\boldsymbol{u}) = \begin{pmatrix} \frac{1}{4} \cdot \left(\frac{3}{5}\right)^2 + \frac{1}{2} \\ \frac{1}{4} \cdot \frac{3}{5} \cdot \frac{1}{2} + \frac{1}{4} \cdot \frac{1}{2} \cdot 1 + \frac{1}{4} \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{59}{100} \\ \frac{9}{20} \\ 0 \\ 1 \end{pmatrix} \overset{\checkmark}{\leq} \boldsymbol{u}\,.$$

The exact LFP of f_Δ is

$$\mu f_\Delta = \begin{pmatrix} 2-\sqrt{2} & \sqrt{2}-1 & 0 & 1 \end{pmatrix}^\top \approx \begin{pmatrix} 0.586 & 0.414 & 0 & 1 \end{pmatrix}^\top,$$

on which $\boldsymbol{u}$ is indeed an upper bound. ◀

We can devise similar certificates for *lower bounds* on the return probabilities of universally cPAST pPDA by instantiating the corresponding result for PPSs.[82] I omit the technical details here, as they add little additional insight to the discussion. See Proposition 4.38 on page 120 for an alternative approach to certifying lower bounds.

82: See Theorem 4.29 on page 112.

4.3.2 Certificates for PAST

We now turn to certificates for (universal) PAST of pPDA, that is, termination in finite expected time (from every initial configuration). We begin by stating our certification result, which is one of the main contributions of this work. Unlike the various certificates from

the previous sections, our certificates for PAST are *unconditionally complete*:

Theorem 4.36 (Certificates for PAST) *Let $\Delta = (Q, \Gamma, P)$ be a pPDA. Then Δ is universally PAST*[83] *if and only if there exist* rational *vectors $\boldsymbol{u} \in \mathbb{Q}_{\geq 0}^{Q \times \Gamma \times Q}$ and $\mathbf{h} \in \mathbb{Q}_{\geq 0}^{Q \times \Gamma}$ such that*

$$\boldsymbol{f}_\Delta(\boldsymbol{u}) \leq \boldsymbol{u} \quad \text{and} \quad A_\Delta(\boldsymbol{u}) \cdot \mathbf{h} + P\mathbf{1} \leq \mathbf{h} ,$$

where $\boldsymbol{f}_\Delta$ is the fundamental PPS of Δ and $A_\Delta(\cdot)$ is the matrix encoding the linear expected runtime equation system[84] *of Δ.*

Moreover, any pair of vectors[85] *$\boldsymbol{u}, \mathbf{h} \geq \mathbf{0}$ satisfying the above inequalities witnesses that the expected runtime from every head $qZ \in Q \times \Gamma$ is at most $\mathbf{h}(qZ)$.*

83: See Definition 3.30 on page 76.

84: See Corollary 3.31B (page 76). Recall that the coefficients of $A_\Delta(\cdot)$ depend on an argument vector indexed by $Q \times \Gamma \times Q$; here the argument is $\boldsymbol{u}$.

85: Not necessarily rational.

Proof. We start with the easier direction $\Longleftarrow$ (soundness). Recall that to prove universal PAST, it suffices to show that the expected runtime from every head $qZ \in Q \times \Gamma$ is finite, which is the case if (and only if) the linear runtime PPS

$$\boldsymbol{y} \mapsto A_\Delta(\mu \boldsymbol{f}_\Delta) \cdot \boldsymbol{y} + P\mathbf{1}$$

is feasible.[86] But since $\boldsymbol{f}_\Delta(\boldsymbol{u}) \leq \boldsymbol{u}$, we have $\mu \boldsymbol{f}_\Delta \leq \boldsymbol{u}$ and thus

$$A_\Delta(\mu \boldsymbol{f}_\Delta) \cdot \mathbf{h} + P\mathbf{1} \leq A_\Delta(\boldsymbol{u}) \cdot \mathbf{h} + P\mathbf{1} \leq \mathbf{h} ,$$

where the first inequality holds because the matrix $A_\Delta(\cdot)$ depends nonnegatively on its argument. The vector $\mathbf{h} < \boldsymbol{\infty}$ is thus an inductive upper bound witnessing feasibility of the runtime system.

86: See Corollary 3.31 on page 76.

The converse direction $\Longrightarrow$ (completeness) is more involved. First, since Δ is universally PAST, it is also universally cPAST.[87] Thus, we can find arbitrarily tight rational inductive upper bounds $\boldsymbol{u}$ on the return probabilities $\mu \boldsymbol{f}_\Delta$ of Δ.[88] Furthermore, since Δ is universally PAST, the linear runtime PPS $\boldsymbol{y} \mapsto A_\Delta(\mu \boldsymbol{f}_\Delta) \cdot \boldsymbol{y} + P\mathbf{1}$ is feasible.[89]

87: See Theorem 2.54 on page 45.

88: By Corollary 4.34. Recall that this relies crucially on our characterization of nondegeneracy of the fundamental PPS from Theorem 4.33.

89: See Corollary 3.31 on page 76.

Assume for the moment that the runtime PPS is clean. Then, by Lemma 4.31, the matrix $A_\Delta(\mu \boldsymbol{f}_\Delta)$ is convergent, which is equivalent to $\rho\left(A_\Delta(\mu \boldsymbol{f}_\Delta)\right) < 1$.[90] Since both $\rho(\cdot)$ and $A_\Delta(\cdot)$ depend continuously on (the coefficients of) their arguments, we can choose $\boldsymbol{u} \geq \mu \boldsymbol{f}_\Delta$ sufficiently close to $\mu \boldsymbol{f}_\Delta$ so that also $\rho(A_\Delta(\boldsymbol{u})) < 1$. The overapproximated runtime system

$$\boldsymbol{y} \mapsto A_\Delta(\boldsymbol{u}) \cdot \boldsymbol{y} + P\mathbf{1} \tag{4.6}$$

90: See Lemma 4.20 on page 103.

is thus also clean and feasible (hence nondegenerate). We can now pick $\mathbf{h}$ as any rational inductive upper bound for the linear PPS in Equation (4.6), which exists according to Theorem 4.23.[91]

If the runtime PPS is unclean[92] then we consider its cleaned-up variant. Cleaning preserves feasibility. We can thus apply the above argument to the cleaned-up runtime PPS to obtain a rational inductive upper bound on its overapproximated version. Lastly, we can pad this upper bound with additional zeros[93] to obtain the desired vector $\mathbf{h}$. □

91: Such a rational $\mathbf{h}$ exists even if the matrix $A_\Delta(\boldsymbol{u})$ and the vector $P\mathbf{1}$ have *irrational* entries, which could be the case if Δ has irrational transition probabilities.

92: This can only happen in improper pPDA, as $P\mathbf{1} = \mathbf{1}$ for proper pPDA.

93: As in the proof of Corollary 4.24 on page 107.

Let us now consider an example of PAST certification.

► **Example 4.37** (Certifying PAST) Reconsider the pPDA Δ from Example 4.35 on page 116 (restated in the margin). We prove that Δ is universally PAST by verifying a certificate as in Theorem 4.36. We have already verified that

$$\boldsymbol{u} \;=\; \begin{pmatrix} 3/5 & 1/2 & 0 & 1 \end{pmatrix}^\top$$

is an inductive upper bound on the return probabilities of Δ. It remains to reveal a rational vector $\mathbf{h} \in \mathbb{Q}_{\geq 0}^{Q\times\Gamma}$ such that

$$A_\Delta(\boldsymbol{u}) \cdot \mathbf{h} + P\mathbf{1} \;\leq\; \mathbf{h}\,.$$

Notice that $P\mathbf{1} = \mathbf{1}$ as Δ is proper. We claim that

$$\mathbf{h} \;=\; \begin{pmatrix} 2 & 1 \end{pmatrix}^\top$$

is such a vector. To verify this, we write the overapproximated linear runtime system $\boldsymbol{y} \mapsto A_\Delta(\boldsymbol{u})\boldsymbol{y} + \mathbf{1}$ over variables $\boldsymbol{y} = \begin{pmatrix}\langle qZ\rangle & \langle rZ\rangle\end{pmatrix}^\top$ explicitly as[94]

$$\begin{pmatrix}\langle qZ\rangle \\ \langle rZ\rangle\end{pmatrix} \;\mapsto\; \begin{pmatrix}\frac{1}{4}\left(\langle qZ\rangle + \boldsymbol{u}(q,Z,q)\cdot\langle qZ\rangle + \boldsymbol{u}(q,Z,r)\cdot\langle rZ\rangle\right) + 1 \\ 1\end{pmatrix}.$$

94: These expressions are obtained by writing out the equation-wise definition of $\boldsymbol{y} = A_\Delta(\boldsymbol{u})\boldsymbol{y} + \mathbf{1}$ from Corollary 3.31 on page 76.

We can now check that $\mathbf{h}$ is inductive for this system:

$$\begin{pmatrix}\frac{1}{4}\left(2 + \frac{3}{5}\cdot 2 + \frac{1}{2}\cdot 1\right) + 1 \\ 1\end{pmatrix} \;=\; \begin{pmatrix}\frac{77}{40} \\ 1\end{pmatrix} \;\overset{\checkmark}{\leq}\; \mathbf{h}\,.$$

This concludes the verification that Δ is universally PAST, and moreover proves that the expected runtime from initial configuration qZ is at most $\mathbf{h}(qZ) = 2$. ◄

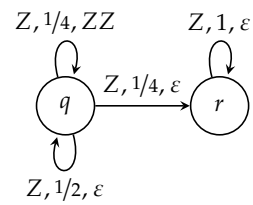


Figure 4.4: The pPDA Δ from Example 4.35 restated.

It is also possible to extract *lower bounds* from a termination proof. Indeed, if a proper pPDA $\Delta = (Q, \Gamma, P)$ is universally AST, then we must have

$$\sum_{r\in Q} [qZ\,r] \;=\; 1 \tag{4.7}$$

for all heads $qZ \in Q \times \Gamma$. This observation leads to the following result:

Proposition 4.38 (From termination to lower bounds) *Let the proper*[95] *pPDA Δ be universally AST and let $\boldsymbol{u} \geq \mu \boldsymbol{f}_\Delta$ be an upper bound on the return probabilities of Δ. Then we have*

$$[qZ\,r] \;\geq\; 1 - \sum_{r' \neq r} \boldsymbol{u}(q, Z, r')$$

for all states $q, r \in Q$ and stack symbols $Z \in \Gamma$.

95: It is possible for slightly technical to extend this statement to improper pPDA.

Proof. The statement is an immediate consequence of Equation (4.7). □

Proposition 4.38 entails that our PAST certificates also witness lower bounds on the return probabilities.

► **Example 4.39** (Extracting lower bounds from a PAST certificate) Consider the proper pPDA Δ from Examples 4.35 and 4.37 once again. We have already verified that Δ is universally PAST (and thus AST[96]) and that the following upper bounds on the return probabilities are valid:

96: See Theorem 2.54 on page 45.

$$\mu \boldsymbol{f}_\Delta \;=\; \begin{pmatrix}[qZ\,q] & [qZ\,r] & [rZ\,q] & [rZ\,r]\end{pmatrix}^\top \;\leq\; \begin{pmatrix}\frac{3}{5} & \frac{1}{2} & 0 & 1\end{pmatrix}^\top \;=\; \boldsymbol{u}$$

Thus, we conclude by Proposition 4.38 that

$$\mu \boldsymbol{f}_\Delta \;\geq\; \begin{pmatrix}1-\frac{1}{2} & 1-\frac{3}{5} & 1-1 & 1-0\end{pmatrix}^\top \;=\; \begin{pmatrix}\frac{1}{2} & \frac{2}{5} & 0 & 1\end{pmatrix}^\top$$

is a valid lower bound. ◄

With Proposition 4.38, we could even go one step further and extend our PAST certificates with additional *lower bounds on the expected runtimes*. I omit the technical details as this extension requires little additional insight.

▷ **Remark 4.40** (On the size of certificates) The size of a PAST certificate as in Theorem 4.36 on page 118 may be exponential in the size of the pPDA Δ, even if we encode rational numbers in binary. To see this, note that the vector $\mathbf{h}$ in Theorem 4.36 upper-bounds the expected runtimes from all heads of Δ. We saw already in Example 3.33 on page 78 that expected runtimes can be doubly exponential in magnitude in the size of Δ; their binary encoding is thus still exponentially large.

It is plausible that either there are no polynomially sized—and PTIME-verifiable—certificates for (universal) PAST, or such certificates are extraordinarily complex. Indeed, the existence of such certificates would entail that deciding PAST for pPDA is in NP. Currently, the best known upper

Description	Certificate data	Condition(s)	Certified property	Remarks
Upper bounds [Corollary 4.34]	$\boldsymbol{u} \in \mathbb{Q}_{\geq 0}^{Q\times\Gamma\times Q}$	$\boldsymbol{f}(\boldsymbol{u}) \leq \boldsymbol{u}$	$\forall qZr \in Q\times\Gamma\times Q$: $[qZ\,r] \leq \boldsymbol{u}_{qZr}$	Certificate exists if Δ is cPAST
Lower bounds [Theorem 4.29]	$\boldsymbol{\ell}, \boldsymbol{u} \in \mathbb{Q}_{\geq 0}^{Q\times\Gamma\times Q}$	$\boldsymbol{\ell} \leq \boldsymbol{f}(\boldsymbol{\ell})$ $\boldsymbol{f}(\boldsymbol{u}) < \boldsymbol{u}$ $\boldsymbol{\ell} \leq \boldsymbol{u}$	$\forall qZr \in Q\times\Gamma\times Q$: $\boldsymbol{\ell}_{qZr} \leq [qZ\,r] < \boldsymbol{u}_{qZr}$	Certificate exists if Δ is cPAST
PAST [Theorem 4.36]	$\boldsymbol{u} \in \mathbb{Q}_{\geq 0}^{Q\times\Gamma\times Q}$ $\mathbf{h} \in \mathbb{Q}_{\geq 1}^{Q\times\Gamma}$	$\boldsymbol{f}(\boldsymbol{u}) \leq \boldsymbol{u}$ $A_\Delta(\boldsymbol{u})\mathbf{h} + P\mathbf{1} \leq \mathbf{h}$	$\forall qZ \in Q\times\Gamma$: $\mathrm{ert}(qZ) \leq \mathbf{h}_{qZ}$	Certificate exists iff Δ is PAST

Figure 4.5: $\Delta = (Q, \Gamma, P)$ is a pPDA with fundamental PPS $\boldsymbol{f}$ (see Theorem 3.11 on page 58). $A_\Delta(\boldsymbol{u})$ is a matrix whose coefficients can be read off Δ and $\boldsymbol{u}$ (see Corollary 3.31B on page 76). $\mathrm{ert}(qZ)$ denotes the *expected runtime* of Δ from initial configuration qZ (formally defined in Definition 3.30 on page 76). The first two rows apply more generally to *arbitrary* PPS (certificate existence is then guaranteed if $\boldsymbol{f}$ is nondegenerate; see Corollary 4.24 on page 107).

bound for this problem is PSPACE [45]. There seems to be no known lower bound for the PAST problem; however, deciding AST is known to be hard for the *square root sum problem* [50, Section 5], whose membership in NP is a long-standing open question. We conjecture that PAST is also square root sum-hard. If this conjecture was correct, then placing the PAST problem in NP would imply that the square root sum problem is in NP as well. .... ◁

We conclude the technical portion of this chapter with an overview of our various pPDA certificates (Figure 4.5). So far, we have described only what certificates are. We shall study a *certifying algorithm* in Chapter 5.

4.4 Open Questions

1. Does the GFP of the fundamental PPS of a pPDA have a natural interpretation?
2. What are (complete) certificates for AST of pPDA? Could we exploit or adapt existing proof rules for probabilistic programs [93, 98] to the pPDA setting?
3. Does there exist a pPDA with rational transition probabilities whose fundamental PPS does not have any rational inductive upper bounds?[97] (By Corollary 4.24 on page 107, such a pPDA cannot be universally cPAST.)
4. Is it feasible to formalize the theory of this chapter in a proof assistant such as Lean or Isabelle/HOL and, based on the formalization, derive a verified certificate checker implementation?
5. What is the precise complexity of deciding PAST for pPDA? Is the problem square root sum-hard as we conjectured in Remark 4.40?

[93]: Majumdar et al. (2025), "Sound and Complete Proof Rules for Probabilistic Termination".

[98]: McIver et al. (2018), "A new proof rule for almost-sure termination".

97: In Proposition 4.25 on page 108 we exhibited a rational PPS without a rational inductive upper bounds; however, this example does not obviously correspond to the fundamental PPS of any pPDA.

4.5 Additional Bibliographic Remarks

The general idea of certificates—and of algorithms for computing them—pervades computer science [96]. However, apart from our own publications [134, 135], I am not aware of any prior work studying certificates for pPDA explicitly. A notable source of motivation for our studies were pre-existing works on certificates for other probabilistic and quantitative models, such as Markov decision processes [54, 63] and timed automata [129]. Succinct certificates for reachability in traditional (i.e., nonprobabilistic) pushdown systems[98] have also been studied [32] previously.

Various authors have explored general PPSs in depth, including Etessami and Yannakakis [50, Section 7]; and Esparza, Kiefer, and Luttenberger [43]. Among other topics, these researchers have studied the theoretical properties of *Newton's method*, an iterative technique for approximating the LFP of a PPS. (We can view Newton iteration as a more sophisticated, accelerated form of Kleene iteration.[99]) Although not considering certificates explicitly, the theory developed in the aforementioned papers overlaps in part with the material presented in Section 4.1, especially Section 4.2.1. See the bibliographic references within these sections for details.

The techniques described in this chapter have influenced our later work on certificates for Markov decision processes [26].

[96]: McConnell et al. (2011), "Certifying algorithms".

[54]: Funke et al. (2020), "Farkas Certificates and Minimal Witnesses for Probabilistic Reachability Constraints".

[63]: Hartmanns et al. (2020), "Optimistic Value Iteration".

[129]: Wimmer et al. (2020), "Verified Certification of Reachability Checking for Timed Automata".

98: A traditional pushdown system is a WPDA over the Boolean semiring; see Definition 3.2 on page 51.

[32]: Chistikov et al. (2022), "Subcubic certificates for CFL reachability".

[50]: Etessami et al. (2009), "Recursive Markov chains, stochastic grammars, and monotone systems of nonlinear equations".

[43]: Esparza et al. (2010), "Computing the Least Fixed Point of Positive Polynomial Systems".

99: See Lemma 4.3E on page 90.

[26]: Chatterjee et al. (2025), "Fixed Point Certificates for Reachability and Expected Rewards in MDPs".

Optimistic Value Iteration for Positive Polynomial Systems

5

5.1 Reduction to Strongly Connected PPSs 125
5.1.1 Irreducible Matrices 125
5.1.2 Strongly Connected PPSs . . 126
5.1.3 Decomposed PPS Analysis . . 128
5.2 Perron-Frobenius Cones 130
5.3 The OVI Algorithm 133
5.3.1 Subroutines . . 135
5.3.2 Correctness and Termination . . 137
5.4 OVI in Practice 139
5.4.1 Implementation 140
5.4.2 Experimental Evaluation . . . 143
5.5 Open Questions 146
5.6 Bibliographic Remarks 147

In Chapter 4, we established that *inductive upper bounds* are crucial for certifying pPDA (probabilistic pushdown automata). The present chapter addresses the algorithmic computation of such bounds—and thus of pPDA certificates.

Optimistic Value Iteration In 2020, Hartmanns and Kaminski [63] introduced *Optimistic Value Iteration* (OVI), a heuristic-driven, practically efficient [64] algorithm for approximating expected rewards in Markov decision processes (a class of models not considered in this thesis) up to a given error threshold. The key idea of OVI is, however, more broadly applicable: We can view it as a heuristic for computing a two-sided bound—where the upper bound is inductive—on the LFP (least fixed point) of any ω-continuous function $\boldsymbol{F} \colon \overline{\mathbb{R}}^n_{\geq 0} \to \overline{\mathbb{R}}^n_{\geq 0}$, such as, for instance, a PPS (positive polynomial system; see Lemma 4.3 on page 90), or the *Bellman operator* of a Markov decision process as in the original OVI paper [63]. The basic OVI scheme alternates between the following two stages:

(1) Compute an arbitrary lower bound $\boldsymbol{\ell}$ on the LFP of $\boldsymbol{F}$, for example, $\boldsymbol{\ell} = \boldsymbol{F}^i(\mathbf{0})$ for some $i \geq 0$. (In Markov decision processes, the latter approach is known as *Value Iteration*; see e.g., Puterman's book [112, Section 7.2.4].)

(2) *Guess* a candidate upper bound $\boldsymbol{u} \stackrel{\text{def}}{=} \boldsymbol{\ell} + \boldsymbol{\varepsilon}$ on the LFP of $\boldsymbol{F}$, where $\boldsymbol{\varepsilon} > \mathbf{0}$ is reasonably small, and *check* whether $\boldsymbol{u}$ is inductive (i.e., $\boldsymbol{F}(\boldsymbol{u}) \leq \boldsymbol{u}$, and thus $\mu\boldsymbol{F} \leq \boldsymbol{u}$). If yes, return $(\boldsymbol{\ell}, \boldsymbol{u})$ as the desired two-sided bound; otherwise, go back to step (1) and improve $\boldsymbol{\ell}$.

This basic scheme is not guaranteed to terminate; ensuring that it finds an inductive upper bound eventually is a key challenge in OVI-style algorithms and generally requires problem-specific modifications or additions to steps (1) or (2).

Being Optimistic In step (2) of the foregoing paragraph, the algorithm is *optimistic* in two ways: it trusts that $\boldsymbol{\ell}$ is actually $\boldsymbol{\varepsilon}$-close to the LFP, and that the guessed candidate $\boldsymbol{u}$ is not only an upper bound but also *inductive*. In general, there is no guarantee that either condition holds. However, since $\boldsymbol{F}$ is ω-continuous, the lower bound will eventually become sufficiently tight. To ensure inductivity, the original OVI algorithm refines the candidate $\boldsymbol{u}$ in an

iterative fashion [63]. We shall adopt a different strategy, which we outline in the next paragraph.

Guessing Inductive Upper Bounds in PPSs In this chapter, we present an OVI-style algorithm tailored to PPSs. As is common practice [43, 50], our method first decomposes the given PPS into *strongly connected components* to be processed individually. However, the main novelty of our algorithm is a guided, more informed guessing strategy to obtain inductive upper bounds. Specifically, our OVI maintains and improves both a lower bound $\boldsymbol{\ell}$ on the LFP of the input PPS $\boldsymbol{f}$, and an approximation $\boldsymbol{v}$ of a certain (normalized) *eigenvector* of the Jacobi matrix of $\boldsymbol{f}$ at the current bound $\boldsymbol{\ell}$. Simply put, the algorithm guesses candidate upper bounds $\boldsymbol{u}$ according to the rule[1]

$$\boldsymbol{u} \;=\; \boldsymbol{\ell} + \varepsilon \cdot \boldsymbol{v} \;;$$

that is, each candidate $\boldsymbol{u}$ arises by moving from the current lower bound $\boldsymbol{\ell}$ by a small amount $\varepsilon > 0$ in the direction of the approximate eigenvector $\boldsymbol{v}$. We shall prove that this strategy ensures that some candidate eventually hits the inductive region, provided that the PPS is nondegenerate.[2]

[43]: Esparza et al. (2010), "Computing the Least Fixed Point of Positive Polynomial Systems".

[50]: Etessami et al. (2009), "Recursive Markov chains, stochastic grammars, and monotone systems of nonlinear equations".

1: The actual algorithm also considers variations of this rule with different values of ε; see Section 5.3 on page 133 for details.

2: See Definition 4.15 on page 99.

Chapter Outline, Contributions, and Origins This chapter is structured as follows:

Section 5.1 (Reduction to Strongly Connected PPSs) We recall the (existing) notion of strongly connected PPSs and explain how the decomposed, component-wise analysis proceeds.

Section 5.2 (Perron-Frobenius Cones) We establish the theoretical foundations for our OVI algorithm by proving a geometric property of inductive regions: under suitable assumptions, these regions contain certain *cone-shaped* polytopes.

Section 5.3 (The Optimistic Value Iteration Algorithm) We present our OVI algorithm for PPSs, prove its correctness, and characterize its termination behavior.

Section 5.4 (OVI in Practice) We present an experimental evaluation of our OVI algorithm (and of two baseline approaches) on PPSs derived from pPDA models of recursive probabilistic programs and stochastic context-free grammars.

See Section 5.5 for open questions and Section 5.6 for additional bibliographic remarks.

The material presented in this chapter is based on our TACAS 2023 paper [134], which includes Sections 5.2 to 5.4 as original contributions. This chapter extends the conference version with several additional examples, figures, explanations, and more streamlined proofs.

[134]: Winkler et al. (2023), "Certificates for Probabilistic Pushdown Automata via Optimistic Value Iteration".

5.1 Reduction to Strongly Connected PPSs

In this section, we reduce the problem of computing bounds on the LFP of an arbitrary PPS to the same problem on its *strongly connected components*: sub-PPSs in which every variable depends—directly or indirectly—on every other variable.

Such a modular analysis has two advantages: first, it enables us to exploit the theoretical properties of strongly connected PPSs; second, it typically leads to improved practical performance, as smaller components are easier to handle algorithmically.

5.1.1 Irreducible Matrices

We shall formally define strong connectedness of PPSs via the *irreducibility* of their Jacobi matrices. Irreducible matrices are a standard concept in linear algebra, especially in the theory of nonnegative matrices [113].

Let us define irreducibility: Recall that a directed graph is *strongly connected* if there is a (directed) path from every vertex to every other vertex. Formally, assuming that I is the vertex set and $A \in \{0,1\}^{I\times I}$ is the adjacency matrix, the graph is strongly connected if and only if for every pair of vertices $i, j \in I$, there exists $k \in \mathbb{N}$ such that j is reachable from i via a path of length k, that is, $A^k(i,j) \neq 0$. Notice that, by definition, the empty graph and the two graphs with only one vertex are strongly connected (Figure 5.1).



Figure 5.1: The two one-vertex graphs are both strongly connected.

Definition 5.1 ***(Irreducible matrix)*** Let I be a finite index set. A square matrix $A \in \mathbb{R}^{I\times I}$ is *irreducible* if the directed graph with vertex set I and adjacency matrix $B \in \{0,1\}^{I\times I}$ defined as

$$B(i,j) \;=\; \begin{cases} 1 & \text{if } A(i,j) \neq 0\,, \\ 0 & \text{otherwise}\,, \end{cases}$$

is strongly connected. △

The standard example of an irreducible matrix is

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad \text{inducing the graph} \quad B = \bullet \rightleftarrows \bullet \,,$$

which is strongly connected. A square matrix that is not irreducible is called *reducible*; every identity matrix of dimension at least two is a simple example of a reducible matrix.

The next lemma characterizes reducible matrices:

Lemma 5.2 (Characterization of reducible matrices [113]) *Let $A \in \mathbb{R}^{I \times I}$ be a finite square matrix. The following are equivalent:*

A. *A is reducible.*
B. *$|I| \geq 2$ and we can write A in a block upper-triangular form:*[3]

$$A = \begin{pmatrix} B & D \\ 0 & C \end{pmatrix},$$

where B and C are nonempty square matrices, possibly of different dimensions.

3: That is, we can impose a corresponding order on I.

Refer to Examples 5.5 and 5.9 on the next page and on page 129 for concrete applications of the preceding result.

5.1.2 Strongly Connected PPSs

With the notion of irreducible matrices at hand, we can now define strongly connected PPSs formally.

A variable x of a PPS $\boldsymbol{f}$ is said to *depend directly on* another variable y if the latter appears in the x-component of $\boldsymbol{f}$; that is, if $\partial_y \boldsymbol{f}_x \neq 0$ [43]. The following definition thus ensures that, in strongly connected PPSs, every variable x depends—directly or indirectly—on every other variable $y \neq x$:[4]

4: If there are at least two variables, this definition implies that x also depends—at least indirectly—on itself; if there is just one variable, then x may or may not depend (directly) on itself.

Definition 5.3 ***(Strongly connected PPS [43])*** A PPS $\boldsymbol{f}$ is *strongly connected* if its Jacobi matrix $\boldsymbol{f}'(\boldsymbol{x})$ is irreducible for some (or, equivalently, for every) $\boldsymbol{x} > \boldsymbol{0}$. △

Note that every one-dimensional PPS as well as the empty PPS are strongly connected because all graphs with either zero or one vertex are strongly connected by definition.

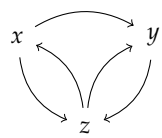
We call a strongly connected PPS *trivial* if it is either empty or one-dimensional of the form $x \mapsto$ constant. For example, the one-dimensional PPSs $x \mapsto 42$ and $x \mapsto \frac{1}{2}x^2 + \frac{1}{2}$ are both strongly connected, but only the second one is nontrivial.

► **Example 5.4** (A strongly connected PPS) The PPS

$$f\colon \begin{pmatrix} x \\ y \\ z \end{pmatrix} \mapsto \begin{pmatrix} y + z^2 + 1 \\ 2z + 2 \\ x^3 + y \end{pmatrix}$$

is strongly connected because, intuitively, x depends directly on y, y depends directly on z, and z depends directly on x; every variable thus depends—directly or indirectly—on every other variable. More formally, we can determine the Jacobi matrix of f and the induced directed graph:

$$f'(x) = \begin{pmatrix} 0 & 1 & 2z \\ 0 & 0 & 2 \\ 3x^2 & 1 & 0 \end{pmatrix}$$



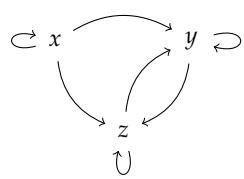
Evidently, the graph is strongly connected. ◄

► **Example 5.5** (A non-strongly connected PPS) The PPS

$$f\colon \begin{pmatrix} x \\ y \\ z \end{pmatrix} \mapsto \begin{pmatrix} \frac{1}{4}x + y^2 + z + 1 \\ \frac{1}{2}y^2 + \frac{1}{2}z \\ \frac{1}{3}y + \frac{1}{3}z + \frac{1}{3} \end{pmatrix}$$

is not strongly connected because neither y nor z depend on x, not even indirectly. Indeed, the Jacobi matrix of f and its induced graph are:

$$f'(x) = \left(\begin{array}{c|cc} \frac{1}{4} & 2y & 1 \\ \hline 0 & y & z \\ 0 & \frac{1}{3} & \frac{1}{3} \end{array}\right)$$



The graph is not strongly connected as there is no path from neither y nor z to x. Also note that, as Lemma 5.2 asserts, $f'(x)$ is indeed a block upper-triangular matrix (with the two blocks indicated by the horizontal and vertical lines). ◄

5.1.3 Decomposed PPS Analysis

The last ingredient toward componentwise PPS analysis is the notion of *sub-PPSs*:

Definition 5.6 ***(Sub-PPS)*** Let f be a PPS over variables $X \uplus Y$ and let $v \in \mathbb{R}^Y_{\geq 0}$. We define the *sub-PPS of f over X at v* as

$$f_{X,v} : \quad x \mapsto f(x, v)\big|_X \ .$$

If no $x \in X$ depends on any $y \in Y$ in f, we may omit the subscript v and call f_X a *simple* sub-PPS. △

► **Example 5.7** (Sub-PPS) Reconsider the PPS f from Example 5.5 (restated in the margin). The *simple* sub-PPS of f over variables $\{y, z\}$ is

$$f_{\{y,z\}} : \quad \begin{pmatrix} y \\ z \end{pmatrix} \mapsto \begin{pmatrix} \frac{1}{2}y^2 + \frac{1}{2}z \\ \frac{1}{3}y + \frac{1}{3}z + \frac{1}{3} \end{pmatrix} .$$

Notice that neither y nor z depend on x in f. Notice further that $f_{\{y,z\}}$ is strongly connected, but this need not be the case in general.

Next, let $v = (1\ 1)^\top$. An example of a *non-simple* sub-PPS of f is

$$f_{\{x\},v} : \quad x \mapsto \frac{1}{4}x + 3 \ .$$

The polynomial $\frac{1}{4}x + 3$ results from substituting $y = 1$ and $z = 1$ into the x-component of f, that is, into the polynomial $\frac{1}{4}x + y^2 + z + 1$. ..... ◄

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \mapsto \begin{pmatrix} \frac{1}{4}x+y^2+z+1 \\ \frac{1}{2}y^2 + \frac{1}{2}z \\ \frac{1}{3}y + \frac{1}{3}z + \frac{1}{3} \end{pmatrix}$$

Figure 5.2: The PPS f from Example 5.5 on the previous page restated.

The following result, slightly technical but not difficult to prove, is the cornerstone of componentwise PPS analysis:

Proposition 5.8 (PPS decomposition) *Let the PPS f be <u>not</u> strongly connected*[5] *and write its Jacobi matrix in block upper-triangular form*[6]

$$f'(x) = \begin{pmatrix} B & \cdots \\ 0 & C \end{pmatrix} \begin{matrix} \} Y \\ \} Z \end{matrix} \ ;$$

that is, B and C are nonempty square matrices indexed by disjoint sets of variables Y and Z, respectively. Further, let $\bowtie \in \{<, \leq, \geq, >\}$, $v \in \mathbb{R}^Y_{\geq 0}$, and $w \in \mathbb{R}^Z_{\geq 0}$ be such that the sub-PPSs[7] *$f_{Y,w}$ and f_Z satisfy*

$$f_{Y,w}(v) \bowtie v \qquad \text{and} \qquad f_Z(w) \bowtie w \ . \tag{5.1}$$

Then, for $u = (v\ \ w)^\top$ it holds that $f(u) \bowtie u$.

5: See Definition 5.3 on page 126.

6: Such a form exists by Lemma 5.2 on page 126; it is not necessarily unique.

7: See Definition 5.6. Notice that f_Z is a *simple* sub-PPS.

Proof. To prove the statement, we unfold the definitions and exploit the assumptions as follows:

$$\begin{aligned} \boldsymbol{f}(\boldsymbol{u}) &= \boldsymbol{f}(\boldsymbol{v}, \boldsymbol{w}) \\ &= \left(\boldsymbol{f}_{Y,\boldsymbol{w}}(\boldsymbol{v}) \;\; \boldsymbol{f}_Z(\boldsymbol{w})\right)^\top && \text{(by Definition 5.6)} \\ &\bowtie (\boldsymbol{v} \;\; \boldsymbol{w})^\top && \text{(by Equation (5.1))} \\ &= \boldsymbol{u} \,. && \square \end{aligned}$$

Proposition 5.8 yields the following algorithm that reduces the problem of computing (strictly) inductive or coinductive vectors for an arbitrary PPS $\boldsymbol{f}$ to the strongly connected case:

(1) If $\boldsymbol{f}$ is already strongly connected, analyze it directly.
(2) Else, decompose the set of variables of $\boldsymbol{f}$ into Y and Z as in Proposition 5.8 and proceed as follows:
   (A) First, analyze the simple sub-PPS $\boldsymbol{f}_Z$ recursively; let $\boldsymbol{w}$ be the result.
   (B) Second, analyze the sub-PPS $\boldsymbol{f}_{Y,\boldsymbol{w}}$ (again recursively) to obtain the result $\boldsymbol{v}$.
   (C) Return the final result $\boldsymbol{u} = (\boldsymbol{v} \;\; \boldsymbol{w})^\top$ for $\boldsymbol{f}$.

This procedure terminates because it decreases the systems' dimensions in each recursive call. Proposition 5.8 asserts that, if the vectors computed in step (1) are, say, inductive, then the final result $\boldsymbol{u}$ is inductive as well; this observation applies analogously to strictly inductive as well as to (strictly) coinductive vectors.

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \mapsto \begin{pmatrix} \frac{1}{4}x + y^2 + z + 1 \\ \frac{1}{2}y^2 + \frac{1}{2}z \\ \frac{1}{3}y + \frac{1}{3}z + \frac{1}{3} \end{pmatrix}$$

Figure 5.3: The PPS $\boldsymbol{f}$ from Example 5.5 restated.

► **Example 5.9** (Componentwise PPS analysis) Reconsider once again the PPS $\boldsymbol{f}$ from Example 5.5 (restated in the margin). We can write the Jacobi matrix of $\boldsymbol{f}$ in block upper-triangular form as in Example 5.5:[8]

$$\boldsymbol{f}'(\boldsymbol{x}) = \left(\begin{array}{c|cc} \frac{1}{4} & 2y & 1 \\ \hline 0 & y & z \\ 0 & \frac{1}{3} & \frac{1}{3} \end{array}\right).$$

8: In this example, the block upper-triangular form is unique.

Consequently, we decompose the variables of $\boldsymbol{f}$ into $Y = \{x\}$ and $Z = \{y, z\}$. Next, we analyze the—in this example strongly connected—simple sub-PPS of $\boldsymbol{f}$ over Z:

$$\boldsymbol{f}_Z : \quad \begin{pmatrix} y \\ z \end{pmatrix} \mapsto \begin{pmatrix} \frac{1}{2}y^2 + \frac{1}{2}z \\ \frac{1}{3}y + \frac{1}{3}z + \frac{1}{3} \end{pmatrix}.$$

Suppose that we have obtained the inductive upper bound $\boldsymbol{w} = (1/2 \;\; 3/4)^\top$ for $\boldsymbol{f}_Z$. The next step is to analyze the one-dimensional (hence strongly

connected) sub-PPS $f_{Y,w}$:

$$f_{Y,w}: \quad x \mapsto \frac{1}{4}x + \left(\frac{1}{2}\right)^2 + \frac{3}{4} + 1 = \frac{1}{4}x + 2\,.$$

Suppose that we have found the inductive upper bound $3/8$ for $f_{Y,w}$. By Proposition 5.8, the vector $\boldsymbol{u} = (3/8\ 1/2\ 3/4)^\top$ is then an inductive upper bound for the original PPS $\boldsymbol{f}$. ◀

5.2 Perron-Frobenius Cones

In this section, we establish a geometric property of the inductive regions of strongly connected[9], nondegenerate, feasible and clean PPSs, namely that such regions contain certain *cones*, which we define as follows:

9: See Definition 5.3 on page 126.

Definition 5.10 ***(Cone)*** Let $\mathbf{a}, \boldsymbol{v} \in \mathbb{R}^n$ be vectors such that $\|\boldsymbol{v}\|_\infty = 1$, that is, $\boldsymbol{v}$ is normalized with respect to the maximum norm. A *cone* with apex $\mathbf{a}$, axis direction $\boldsymbol{v}$, length $\hat{\delta}$, and width ε is a set

$$C(\mathbf{a}, \boldsymbol{v}, \hat{\delta}, \varepsilon) = \left\{ \mathbf{a} + \delta\tilde{\boldsymbol{v}} \mid 0 \leq \delta \leq \hat{\delta} \wedge \|\tilde{\boldsymbol{v}} - \boldsymbol{v}\|_\infty \leq \varepsilon \right\},$$

where $\hat{\delta}, \varepsilon > 0$. △

Thus, what we call *cone*—a set $C(\mathbf{a}, \boldsymbol{v}, \hat{\delta}, \varepsilon)$ as in Definition 5.10—is the set of all line segments connecting an apex $\mathbf{a}$ to a hypercube with side length 2ε centered around the point $\mathbf{a} + \hat{\delta}\boldsymbol{v}$ (see Figure 5.4 for an illustration in two dimensions). Notice that such a set is not a cone in the strict geometrical sense: in geometry, cones are formed by the line segments connecting an apex to every point of a base shape—often a circle—lying in a *plane* not containing the apex (this definition generalizes naturally to higher-dimensional *hypercones*). We could turn our cones into proper geometric ones by adding additional constraints to the definition of the set $C(\mathbf{a}, \boldsymbol{v}, \hat{\delta}, \varepsilon)$; however, this would unnecessarily complicate the theory.

The next lemma is a classical result about irreducible nonnegative matrices, which—as we shall see shortly—plays a key role in determining the axis directions of the sought-after cones:

Lemma 5.11 (Perron-Frobenius Theorem) *Let $A \geq 0$ be an irreducible matrix.*[10] *Then A has a right eigenvector $\boldsymbol{v} > \mathbf{0}$ with associated eigenvalue $\rho = \rho(A)$, the spectral radius*[11] *of A, that is,*

$$A\boldsymbol{v} = \rho\boldsymbol{v}\,.$$

10: See Definition 5.1 on page 125.

11: See Section 4.2.1.3 on page 103.

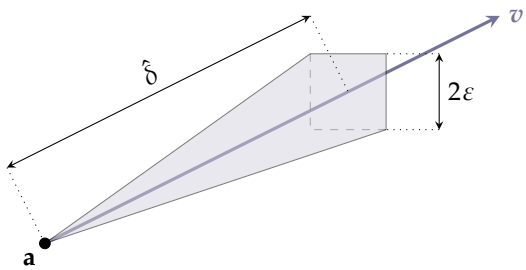


Figure 5.4: Illustration of a cone $C(\mathbf{a}, v, \hat{\delta}, \varepsilon)$ in two dimensions (Definition 5.10). The point $\mathbf{a}$ is the apex, the axis direction is v, the cone extends up to $\hat{\delta}$, and the base is a square of side length 2ε centered at $\mathbf{a} + \hat{\delta}v$.

Furthermore, every right eigenvector $v' > \mathbf{0}$ of A is a scalar multiple of v.

We call the *unique* normalized—we work only with the maximum norm $\|\cdot\|_\infty$ in this chapter—eigenvector $v > \mathbf{0}$ of an irreducible matrix $A \geq 0$ the *Perron-Frobenius eigenvector* of A.

► **Example 5.12** (Perron-Frobenius eigenvector) Consider the matrix

$$A = \begin{pmatrix} 0 & 1 \\ 1/2 & 0 \end{pmatrix}.$$

Its spectral radius is $\rho = \sqrt{2}/2$ and its Perron-Frobenius eigenvector is $v = \begin{pmatrix} 1 & \sqrt{2}/2 \end{pmatrix}^\top$. Indeed, we have $\|v\|_\infty = 1$ and

$$Av = \begin{pmatrix} 0 & 1 \\ 1/2 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ \sqrt{2}/2 \end{pmatrix} = \begin{pmatrix} \sqrt{2}/2 \\ 1/2 \end{pmatrix} = \frac{\sqrt{2}}{2} \cdot \begin{pmatrix} 1 \\ \sqrt{2}/2 \end{pmatrix} = \rho v .$$

Notably, this example also shows that neither the spectral radius nor the coefficients of the Perron-Frobenius eigenvector need to be rational numbers, even if the matrix entries are all rational. ◄

We are now ready to state one of the most important definitions of this chapter:

Definition 5.13 ***(Perron-Frobenius cone)*** Let f be a feasible, clean, and strongly connected PPS, and let v be the Perron-Frobenius eigenvector of $f'(\mu f)$. A *Perron-Frobenius cone* of f is any cone with apex μf and axis direction v. △

Notice that the foregoing definition is well-defined because the LFP μf of a feasible and clean PPS f satisfies $\mu f > \mathbf{0}$; consequently, by Definition 5.3, the Jacobi matrix $f'(\mu f)$ is irreducible, and thus—by Lemma 5.11—has a Perron-Frobenius eigenvector.

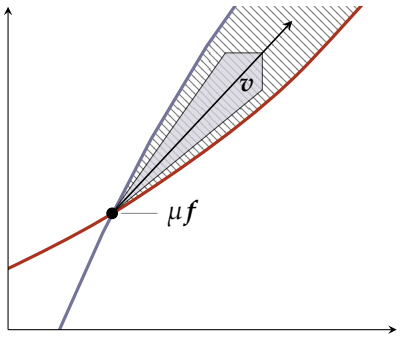


Figure 5.5: Illustration of Theorem 5.14 for a two-dimensional example PPS f. The inductive area (hatched) contains a cone (shaded) whose apex is the LFP μf and whose axis direction is the Perron-Frobenius eigenvector of $f'(\mu f)$.

Theorem 5.14 (Inductive Perron-Frobenius cones) *Let f be a feasible, clean, and strongly connected PPS. Then f is nondegenerate[12] if and only if f has a Perron-Frobenius cone containing—except for the apex μf—only strictly inductive upper bounds.*

12: See Definition 4.15 on page 99.

Proof. Suppose that f is nondegenerate. Since f is clean by assumption, $f'(\mu f)$ is convergent;[13] the spectral radius $\rho = \rho\left(f'(\mu f)\right)$ thus satisfies $\rho < 1$.[14] Moreover, since f is strongly connected and $\mu f > \mathbf{0}$ (f is clean), the Jacobi matrix $f'(\mu f)$ is irreducible.[15]

13: See Definition 4.19 on page 103.

14: See Lemma 4.20 on page 103.

15: See Definition 5.3 on page 126.

Let v be the normalized Perron-Frobenius eigenvector of $f'(\mu f)$; that is, we have $v > \mathbf{0}$, $\|v\|_\infty = 1$, and

$$f'(\mu f) \cdot v \;=\; \rho v \;. \tag{5.2}$$

Fix any number ρ' such that $\rho < \rho' < 1$. Further, let $v_{\min} > 0$ be the minimum entry of v, and define $M = \|f'(\mu f + \mathbf{1})\|_\infty$, the maximum row sum of the Jacobi matrix of f evaluated at the vector $\mu f + \mathbf{1}$.

We can now define the width of the sought-after cone as

$$\varepsilon \;=\; \frac{(1-\rho')v_{\min}}{M+1} \;.$$

Notice that $0 < \varepsilon \leq 1$. For the length of the cone, we choose any sufficiently small value $\hat{\delta} > 0$ such that

$$\forall \tilde{v} : \quad \|v - \tilde{v}\|_\infty \leq \varepsilon \implies f'(\mu f + \hat{\delta}\tilde{v}) \cdot v < \rho' v \;. \tag{5.3}$$

Such a $\hat{\delta}$ exists because the Jacobi matrix of f is continuous in every component, and because $\rho < \rho'$; thus, we can increase the matrix coefficients on the left hand side of Equation (5.2) by a nonzero amount while still keeping the resulting vector strictly smaller than $\rho' v$. We moreover assume without loss of generality that $\hat{\delta} \leq 1/2$.

We now prove that the Perron-Frobenius cone $C(\mu\boldsymbol{f}, \boldsymbol{v}, \hat{\delta}, \varepsilon)$ satisfies the claim of the theorem, that is, contains strictly inductive vectors only (except for the apex $\mu\boldsymbol{f}$, which is inductive but not strictly). To this end, let $0 < \delta \leq \hat{\delta}$ and let $\tilde{\boldsymbol{v}}$ be such that $\|\boldsymbol{v} - \tilde{\boldsymbol{v}}\|_\infty \leq \varepsilon$. We argue as follows:

$$
\begin{aligned}
& \boldsymbol{f}(\mu\boldsymbol{f} + \delta\tilde{\boldsymbol{v}}) \\
\leq\ & \mu\boldsymbol{f} + \boldsymbol{f}'(\mu\boldsymbol{f} + \delta\tilde{\boldsymbol{v}}) \cdot \delta\tilde{\boldsymbol{v}} \\
& \qquad \text{(Taylor's theorem (Corollary 4.18 on page 102))} \\
\leq\ & \mu\boldsymbol{f} + \delta \cdot \boldsymbol{f}'(\mu\boldsymbol{f} + \hat{\delta}\tilde{\boldsymbol{v}}) \cdot (\boldsymbol{v} + \boldsymbol{\varepsilon}) \\
& \qquad \text{(Because } \delta \leq \hat{\delta} \text{ and } \tilde{\boldsymbol{v}} \leq \boldsymbol{v} + \boldsymbol{\varepsilon}) \\
<\ & \mu\boldsymbol{f} + \delta \left(\rho'\boldsymbol{v} + \boldsymbol{f}'(\mu\boldsymbol{f} + \hat{\delta}\tilde{\boldsymbol{v}}) \cdot \boldsymbol{\varepsilon}\right) \qquad \text{(By Equation (5.3))} \\
\leq\ & \mu\boldsymbol{f} + \delta \left(\rho'\boldsymbol{v} + \boldsymbol{f}'(\mu\boldsymbol{f} + \mathbf{1}) \cdot \boldsymbol{\varepsilon}\right) \\
& \qquad \text{(Because } \hat{\delta}\tilde{\boldsymbol{v}} \leq \hat{\delta}(\boldsymbol{v} + \boldsymbol{\varepsilon}) \leq 0.5 \cdot (\mathbf{1} + \mathbf{1}) = \mathbf{1}) \\
\leq\ & \mu\boldsymbol{f} + \delta \left(\rho'\boldsymbol{v} + \|\boldsymbol{f}'(\mu\boldsymbol{f} + \mathbf{1})\|_\infty \cdot \boldsymbol{\varepsilon}\right) \\
& \qquad \text{(Because } \boldsymbol{\varepsilon} \text{ is a constant vector)} \\
\leq\ & \mu\boldsymbol{f} + \delta \left(\rho'\boldsymbol{v} + M\boldsymbol{\varepsilon}\right) \ . \qquad \text{(Definition of } M\text{)}
\end{aligned}
$$

Thus, it suffices to show that

$$\rho'\boldsymbol{v} + M\boldsymbol{\varepsilon} \overset{!}{\leq} \tilde{\boldsymbol{v}} \ .$$

To prove the latter inequality, we exploit the definition of ε:

$$
\begin{aligned}
& \rho'\boldsymbol{v} + M\boldsymbol{\varepsilon} \\
=\ & \rho'\boldsymbol{v} + (M+1)\boldsymbol{\varepsilon} - \boldsymbol{\varepsilon} \\
=\ & \rho'\boldsymbol{v} + (1-\rho')\mathbf{v}_{\min} - \boldsymbol{\varepsilon} \qquad \text{(Definition of } \varepsilon\text{)} \\
\leq\ & \boldsymbol{v} - \boldsymbol{\varepsilon} \qquad \text{(Because } \mathbf{v}_{\min} \leq \boldsymbol{v}\text{)} \\
\leq\ & \tilde{\boldsymbol{v}} \ . \qquad \text{(Because } \|\boldsymbol{v} - \tilde{\boldsymbol{v}}\|_\infty \leq \varepsilon\text{)}
\end{aligned}
$$

This concludes the proof of the *only if* direction. The *if* direction is an immediate consequence of Theorem 4.23 on page 106. □

5.3 *The Optimistic Value Iteration Algorithm*

We are now ready to present our variant of OVI for PPSs; see Algorithm 5.1 on the following page for the pseudocode. It takes as input a clean and strongly connected[16] PPS $\boldsymbol{f}$ and an error bound $\varepsilon > 0$. The goal of the algorithm is to compute an inductive upper bound $\boldsymbol{u}$ within $\|\cdot\|_\infty$-distance ε of the LFP. Notably, the algorithm is

16: Recall that neither assumption is a real loss of generality: we can efficiently clean up any PPS (see Section 4.1.3 on page 92) and conduct a decomposed analysis as outlined in Proposition 5.8 on page 128.

Algorithm 5.1: Optimistic Value Iteration (OVI) for PPS

Input : A clean and strongly connected PPS $\boldsymbol{f}$, an error bound $\varepsilon > 0$
Output: Two vectors $\boldsymbol{\ell}, \boldsymbol{u}$ such that $\boldsymbol{\ell} \leq \mu\boldsymbol{f} \leq \boldsymbol{u}$, $\|\boldsymbol{\ell} - \boldsymbol{u}\|_\infty \leq \varepsilon$, and $\boldsymbol{f}(\boldsymbol{u}) \leq \boldsymbol{u}$
Termination is guaranteed only if $\boldsymbol{f}$ is feasible and nondegenerate.

```
// initialize variables and constants
ℓ ← 0                                                    // lower bound
v ← 1                                  // approximate Perron-Frobenius eigenvector
τ ← ε                                                    // approximation tolerance
n ← 1                                                    // number of guesses per phase
C ← any constant ∈ (0, 1)              // damping factor for approximation tolerance
D ← any constant ∈ (0, 1)                              // damping factor for guessing offset
// main loop
while true do
    ℓ ← improveLB(f, ℓ, τ)                                          // improve lower bound
    v ← approxEV(f'(ℓ), v, τ)                          // improve eigenvector approximation
    for k = 0, 1, ..., n−1 do                                       // guess-and-check phase
        u ← ℓ + D^k ε · v                  // guess a candidate upper bound optimistically
        if f(u) ≤ u then                                            // inductivity check
            return ℓ, u
    n ← n + 1                          // increase number of guesses in the next phase
    τ ← C · τ                                        // decrease approximation tolerance
```

partial: It does not terminate on certain inputs. There are two reasons for partiality: infeasibility and degeneracy of the input PPS $\boldsymbol{f}$ (see Remark 5.17 on page 139 for a detailed discussion of termination).

The basic idea of Algorithm 5.1 is as follows:

1. It maintains and iteratively updates two vectors: $\boldsymbol{\ell}$, a lower bound on $\mu\boldsymbol{f}$, and $\boldsymbol{v}$, a normalized approximation of the Perron-Frobenius eigenvector[17] of $\boldsymbol{f}'(\boldsymbol{\ell})$ (Algorithm 5.1).
2. After each improvement of $\boldsymbol{\ell}$ and $\boldsymbol{v}$, the algorithm optimistically guesses several candidate upper bounds $\boldsymbol{u}$ by adding decreasing multiples of $\varepsilon \cdot \boldsymbol{v}$ to the current lower bound $\boldsymbol{\ell}$ (Algorithm 5.1).
3. Once a candidate $\boldsymbol{u}$ becomes inductive (i.e., satisfies $\boldsymbol{f}(\boldsymbol{u}) \leq \boldsymbol{u}$), the algorithm terminates and returns $\boldsymbol{u}$ (Algorithm 5.1).

17: See Lemma 5.11 on page 130.

See Figure 5.6 on the next page for a visualized example execution of Algorithm 5.1. Termination is guaranteed whenever $\boldsymbol{f}$ is feasible and nondegenerate; in this case, the guessed candidates will eventually hit an inductive Perron-Frobenius cone[18] of $\boldsymbol{f}$. To prevent *overshooting* the cone, it is vital to consider multiple decreasing guesses as mentioned in step 2 in the foregoing algorithmic outline.[19]

18: See Theorem 5.14 on page 132.

19: See Figure 5.6b on the next page for an illustration and Theorem 5.16 on page 138 for the formal proof.

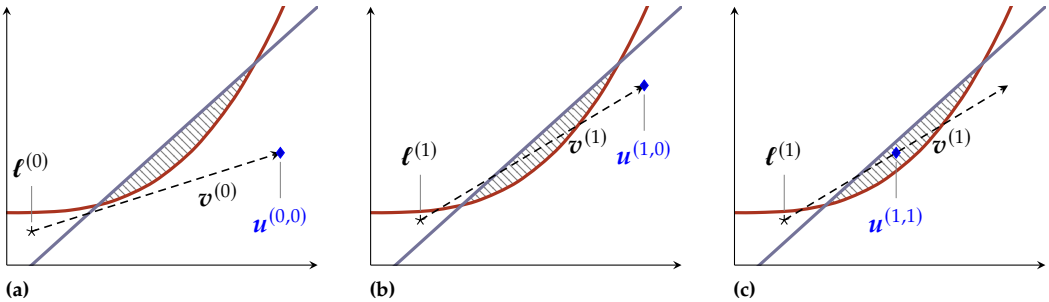


Figure 5.6: Illustration of an example execution of OVI (Algorithm 5.1 on the previous page) on a two-dimensional PPS. **(a)** The first eigenvector approximation $v^{(0)}$ is too coarse; the first optimistic guess $u^{(0,0)}$ misses the inductive region. **(b)** The second eigenvector approximation $v^{(1)}$ is more accurate; however, the second guess $u^{(1,0)}$ overshoots. **(c)** The third guess $u^{(1,1)} = \ell^{(1)} + 0.5v^{(1)}$ (resulting from the concrete parameters $\varepsilon = 1$ and $D = 0.5$ that I choose here for the sake of illustration) is inductive. (*Legend:* solid blue line: x-component; solid red curve: y-component; hatched area: inductive region; dashed arrow: eigenvector approximation.)

5.3.1 Subroutines for Lower Bounds and Eigenvectors

For lower bound and eigenvector approximation (Algorithm 5.1), Algorithm 5.1 relies on two subroutines: `improveLB` and `approxEV`. These functions must satisfy the contracts detailed throughout the upcoming Sections 5.3.1.1 and 5.3.1.2, where we let

$$\tau^{(0)} > \tau^{(1)} > \ldots \to 0$$

be any decreasing sequence of *tolerances* converging to 0. Intuitively, `improveLB` and `approxEV` must be able to approximate the LFP of a PPS and the Perron-Frobenius eigenvector of an irreducible nonnegative matrix, respectively, up to arbitrary precision. The tolerance is not necessarily related to the approximation error; we require only that the approximation error goes to zero as the tolerance does.

We now state the contracts and possible implementations (which rely on standard methods) formally.

5.3.1.1 Computation of Lower Bounds

For every feasible, clean, and strongly connected PPS f, `improveLB` must satisfy that

$$\lim_{i\to\infty} \ell^{(i)} = \mu f\,, \qquad \text{(equivalently: } \sup_{i\in\mathbb{N}} \ell^{(i)} = \mu f\text{)}$$

where $\boldsymbol{\ell}^{(0)} = \mathbf{0}$ and, for all $i \geq 0$,

$$\mu f \;\geq\; \boldsymbol{\ell}^{(i+1)} \;\stackrel{\text{def}}{=}\; \texttt{improveLB}(f, \boldsymbol{\ell}^{(i)}, \tau^{(i)}) \;\geq\; \boldsymbol{\ell}^{(i)} \,.$$

The easiest method to implement $\texttt{improveLB}(f, \boldsymbol{\ell}, \tau)$ according to its contract is via *Kleene iteration*:[20]

20: See Lemma 4.3E on page 90.

```
repeat
    ℓ_old ← ℓ                          // Save the old iterate
    ℓ ← f(ℓ)                           // Kleene update
until ‖ℓ − ℓ_old‖∞ ≤ τ
return ℓ
```

The preceding scheme computes $f^i(\boldsymbol{\ell})$ for a sufficiently large i such that $\|f^i(\boldsymbol{\ell}) - f^{i-1}(\boldsymbol{\ell})\|_\infty \leq \tau$ holds. Kleene's fixed point theorem ensures that this implementation satisfies the contract of `improveLB`.

An alternative implementation of `improveLB` is via *Newton's method*. The latter may converge much faster than Kleene iteration, but requires solving a linear equation system in each iteration, making it more expensive per iteration and more difficult to implement overall. Newton's method for PPSs has been thoroughly studied in the literature [43, 50]; I omit a detailed description here.

[43]: Esparza et al. (2010), "Computing the Least Fixed Point of Positive Polynomial Systems".

[50]: Etessami et al. (2009), "Recursive Markov chains, stochastic grammars, and monotone systems of nonlinear equations".

5.3.1.2 *Approximation of Perron-Frobenius Eigenvectors*

For every irreducible matrix $A \geq 0$ with (exact) normalized Perron-Frobenius eigenvector $\boldsymbol{v}_{\text{exact}}$, we require `approxEV` to satisfy that

$$\lim_{i \to \infty} \boldsymbol{v}^{(i)} \;=\; \boldsymbol{v}_{\text{exact}} \,,$$

where $\boldsymbol{v}^{(0)} > \mathbf{0}$ and, for all $i \geq 0$, $\boldsymbol{v}^{(i+1)} \stackrel{\text{def}}{=} \texttt{approxEV}(A, \boldsymbol{v}^{(i)}, \tau^{(i)})$ is normalized with regard to $\|\cdot\|_\infty$.

A basic approach to implement $\texttt{approxEV}(A, \boldsymbol{v}, \tau)$ is the following *power iteration* scheme:

```
repeat
    v_old ← v                          // Save previous iterate
    v ← (A + Id)v   // Use A + Id to ensure aperiodicity
    v ← v/‖v‖∞                         // Normalize
until ‖v − v_old‖∞ ≤ τ
return v
```

The preceding pseudocode effectively computes the vector

$$\boldsymbol{v}^i \overset{\text{def}}{=} \frac{(A + Id)^i \boldsymbol{v}}{\|(A + Id)^i \boldsymbol{v}\|_\infty}$$

for a sufficiently large i such that $\|\boldsymbol{v}^i - \boldsymbol{v}^{i-1}\|_\infty \leq \tau$. Importantly, adding the identity matrix to A does not alter the eigenvectors[21] of A and ensures that the $\boldsymbol{v}^i$ converge to $\boldsymbol{v}_{\text{exact}}$ even if A is *periodic* [115, Theorem 4.1].

21: It only shifts the eigenvalues by 1 on the real axis.

[115]: Saad (2011), *Numerical Methods for Large Eigenvalue Problems*.

► **Example 5.15** (Power iteration) We illustrate eigenvector approximation via power iteration on the matrix

$$A = \begin{pmatrix} 0 & 1 \\ 1/2 & 0 \end{pmatrix}.$$

Note that A is irreducible but periodic (the induced graph has only one cycle, which has length two). Therefore, direct power iteration on A does not converge for certain initial values of $\boldsymbol{v}$; for example, the initial vector $\boldsymbol{v} = (1\ 1)^\top$ leads to oscillation between $\boldsymbol{v}$ and $(1\ 1/2)^\top$.

To avoid this issue, the proposed implementation of `approxEV` instead performs power iteration on the matrix

$$A + Id = \begin{pmatrix} 1 & 1 \\ 1/2 & 1 \end{pmatrix},$$

which is aperiodic and has the same Perron-Frobenius eigenvector as A. Starting from $\boldsymbol{v} = \boldsymbol{v}^0 = (1\ 0)^\top$, the first few iterates are as follows:

$$\begin{aligned} \boldsymbol{v}^1 &= (1\ 0.5000)^\top \\ \boldsymbol{v}^2 &\approx (1\ 0.6667)^\top \\ \boldsymbol{v}^3 &= (1\ 0.7000)^\top \\ \boldsymbol{v}^4 &\approx (1\ 0.7059)^\top \end{aligned}$$

With $\tau = 0.01$, our implementation of `approxEV` would terminate after the fourth iteration, returning $\boldsymbol{v}^4$ as an approximation of the Perron-Frobenius eigenvector

$$\boldsymbol{v}_{\text{exact}} = \left(1\ \sqrt{2}/2\right)^\top \approx (1\ 0.7071)^\top$$

of A. Decreasing the tolerance τ would lead to further iterations and a more accurate approximation. ◄

5.3.2 Correctness and Termination of OVI

In this section, we prove the correctness and termination of our OVI algorithm under suitable preconditions:

Theorem 5.16 (Partial correctness and termination) *Algorithm 5.1 is partially correct:* If *it terminates on an input* $(\boldsymbol{f}, \varepsilon)$*, where* $\boldsymbol{f}$ *is a clean*[22] *and strongly connected*[23] *PPS and* $\varepsilon > 0$*, then it returns vectors* $\boldsymbol{\ell} \geq \mathbf{0}$ *and* $\boldsymbol{u} > \mathbf{0}$ *such that* $\boldsymbol{\ell} \leq \mu\boldsymbol{f} \leq \boldsymbol{u}$*,* $\|\boldsymbol{\ell} - \boldsymbol{u}\|_\infty \leq \varepsilon$*, and* $\boldsymbol{f}(\boldsymbol{u}) \leq \boldsymbol{u}$*.*

Moreover, Algorithm 5.1 terminates if $\boldsymbol{f}$ *is feasible and nondegenerate.*[24]

22: See Definition 4.5 on page 92.

23: See Definition 5.3 on page 126.

24: See Definition 4.15 on page 99.

Proof. We start with partial correctness, which is the easier part of the proof:

25: See Section 5.3.1.1 on page 135.

- $\boldsymbol{\ell}$ is initially $\mathbf{0}$, the trivial a lower bound, and remains a lower bound on $\mu\boldsymbol{f}$ throughout the main loop because the `improveLB` function preserves lower bounds according to its contract.[25]
- Because of the condition in Algorithm 5.1, $\boldsymbol{u}$ is inductive upon return (i.e., $\boldsymbol{f}(\boldsymbol{u}) \leq \boldsymbol{u}$); as a consequence, $\mu\boldsymbol{f} \leq \boldsymbol{u}$ and thus $\mathbf{0} < \boldsymbol{u}$ because $\boldsymbol{f}$ is clean.
- Since $\mathrm{D}^k \leq 1$ and $\|\boldsymbol{v}\|_\infty = 1$ by the contract of `approxEV`,[26] any vector $\boldsymbol{u}$ constructed in Algorithm 5.1 satisfies

26: See Section 5.3.1.2 on page 136.

$$\|\boldsymbol{\ell} - \boldsymbol{u}\|_\infty \;=\; \|\mathrm{D}^k\varepsilon \cdot \boldsymbol{v}\|_\infty \;=\; \mathrm{D}^k\varepsilon \;\leq\; \varepsilon\,.$$

27: See Theorem 5.14 on page 132.

28: See Definition 5.13 on page 131.

To prove termination under the stated preconditions, we rely on the existence of an inductive Perron-Frobenius cone.[27] Let $C(\mu\boldsymbol{f}, \boldsymbol{v}_{\text{exact}}, \hat{\delta}, e)$ be such a cone. Recall[28] that $\boldsymbol{v}_{\text{exact}}$ is the *exact* (normalized) Perron-Frobenius eigenvector of $\boldsymbol{f}'(\mu\boldsymbol{f})$, and $\hat{\delta}, e > 0$.

We claim that a candidate upper bound $\boldsymbol{u}$ constructed in Algorithm 5.1 falls within the cone eventually, that is, $\boldsymbol{u} = \mu\boldsymbol{f} + \delta\tilde{\boldsymbol{v}}$ for some $0 < \delta \leq \hat{\delta}$ and some $\tilde{\boldsymbol{v}}$ with $\|\boldsymbol{v}_{\text{exact}} - \tilde{\boldsymbol{v}}\|_\infty \leq e$. We now prove this claim.

Fix a sufficiently large $k \geq 0$ such that $0 < \mathrm{D}^k\varepsilon \leq \hat{\delta}$. Toward contradiction, assume that the algorithm does not terminate. Denote by $\boldsymbol{\ell}^{(i)}$ and $\boldsymbol{v}^{(i)}$ the values of variables $\boldsymbol{\ell}$ and $\boldsymbol{v}$, respectively, obtained during the i-th iteration ($i \geq 0$) of the main loop. Due to the contracts of `improveLB` and `approxEV`, we have

$$\lim_{i\to\infty} \boldsymbol{\ell}^{(i)} \;=\; \mu\boldsymbol{f} \quad \text{and} \quad \lim_{i\to\infty} \boldsymbol{v}^{(i)} \;=\; \boldsymbol{v}_{\text{exact}}\,.$$

For $k \leq i$, we can write $\boldsymbol{u}^{(i,k)}$ as constructed in Algorithm 5.1 as

follows:

$$\begin{aligned}\boldsymbol{u}^{(i,k)} &= \boldsymbol{\ell}^{(i)} + \mathrm{D}^k\varepsilon \cdot \boldsymbol{v}^{(i)} \\ &= \mu\boldsymbol{f} + \mathrm{D}^k\varepsilon \underbrace{\left(\boldsymbol{v}^{(i)} + \frac{1}{\mathrm{D}^k\varepsilon}\left(\boldsymbol{\ell}^{(i)} - \mu\boldsymbol{f}\right)\right)}_{=:\,\tilde{\boldsymbol{v}}^{(i)}} . \end{aligned} \tag{5.4}$$

Thus, it remains to show that $\|\boldsymbol{v}_{\mathrm{exact}} - \tilde{\boldsymbol{v}}^{(i)}\|_\infty \leq e$ for some sufficiently large i. We have[29]

$$\begin{aligned}&\lim_{i\to\infty} \|\boldsymbol{v}_{\mathrm{exact}} - \tilde{\boldsymbol{v}}^{(i)}\|_\infty \\ &= \|\boldsymbol{v}_{\mathrm{exact}} - \boldsymbol{v}^{(i)} - \frac{1}{\mathrm{D}^k\varepsilon}\left(\boldsymbol{\ell}^{(i)} - \mu\boldsymbol{f}\right)\|_\infty \qquad \text{(Definition of } \tilde{\boldsymbol{v}}^{(i)}\text{)} \\ &= \|\boldsymbol{v}_{\mathrm{exact}} - \lim_{i\to\infty}\boldsymbol{v}^{(i)} - \frac{1}{\mathrm{D}^k\varepsilon}\left(\lim_{i\to\infty}\boldsymbol{\ell}^{(i)} - \mu\boldsymbol{f}\right)\|_\infty \\ &= 0\,. \quad \text{(because } \lim_{i\to\infty}\boldsymbol{v}^{(i)} = \boldsymbol{v}_{\mathrm{exact}} \text{ and } \lim_{i\to\infty}\boldsymbol{\ell}^{(i)} = \mu\boldsymbol{f}\text{)}\end{aligned}$$

Therefore, since $e > 0$, there must exist $i \geq k$ such that $\|\boldsymbol{v}_{\mathrm{exact}} - \tilde{\boldsymbol{v}}^{(i)}\|_\infty \leq e$. Such an i satisfies $\boldsymbol{u}^{(i,k)} \in C(\mu\boldsymbol{f}, \boldsymbol{v}_{\mathrm{exact}}, \hat{\delta}, e)$ by Equation (5.4). Thus, $\boldsymbol{u}^{(i,k)}$ is (strictly) inductive, the inductivity check in Algorithm 5.1 passes, and the algorithm terminates. □

29: For this argument to work, it is crucial that, thanks to the inner **for** loop, k does not depend on i.

▷ **Remark 5.17** (On termination of OVI) It is possible to run Algorithm 5.1 on input PPSs whose feasibility status is unknown a priori, and—if termination occurs—conclude feasibility.[30] On an infeasible input PPS, OVI will diverge. However, recall that feasibility is guaranteed for certain classes of PPSs, such as those induced by pPDA.[31]

The second cause of nontermination is degeneracy of the input PPS: A degenerate PPS lacks an inductive Perron-Frobenius cone;[32] therefore, there is no guarantee that the optimistic guesses of Algorithm 5.1 will ever hit the inductive region. In theory, it is still possible that OVI guesses an inductive upper bound of a degenerate PPS by chance (e. g., its exact LFP); however, in practice, this is unlikely to happen.[33] Whenever OVI returns a *strictly* inductive upper bound (which is the typical case), we know that the input PPS must have been nondegenerate.[34] ◁

30: In fact, the returned inductive upper bound is a *certificate of feasibility*.

31: See Lemma 4.32 on page 115.

32: See Theorem 5.14 on page 132.

33: We have not observed such lucky guesses in our experiments (Section 5.4).

34: See Theorem 4.23 on page 106.; strict inductive upper bounds certify nondegeneracy in clean PPSs.

5.4 OVI in Practice

In this section, I report on experiments conducted to evaluate the practical performance of OVI for computing certificates for pPDA, that is, inductive upper bounds on the LFPs of their fundamental PPSs.[35] The experiments also include a comparison with two base-

35: See Theorem 3.11 on page 58.

```
bool and() { // entry point
  prob {
    1/2: return (1/2:true | 1/2:false);
    1/2: {
      if(!or()) return false;
      else return or();
    }
  }
}
```

```
bool or() {
  prob {
    1/2: return (1/2:true | 1/2:false);
    1/2: {
      if(and()) return true;
      else return and();
    }
  }
}
```

Figure 5.7: Probabilistic recursive program evaluating a random and-or tree [22]. The `prob` blocks execute the contained statements with the respective probabilities; they are a probabilistic variant of Java's `switch` statement. Our tool automatically translates this program to a pPDA and computes an inductive upper bound certifying that a call to `and()` returns `true` and `false` with probabilities $\leq 382/657 \approx 0.5814$ and $391/933 \approx 0.4191$, respectively.

line approaches: *satisfiability modulo theories* (SMT) solving, and an OVI variant with a more straightforward guessing heuristic adapted from the original OVI algorithm [63].[36]

36: A peer-reviewed software artifact containing all the code necessary to reproduce the experiments has been archived and is available online [133].

5.4.1 Implementation

We begin by describing our concrete OVI implementation, as well as the implementations of the two baselines methods.

5.4.1.1 Our Tool: Pray

We implemented OVI for PPSs in a prototypical Java tool called Pray (**P**robabilistic **r**ecursion **a**nal**y**zer) [133]. The tool supports two input formats: (1) recursive probabilistic programs in a *Java-esque* syntax (e.g., Figure 5.7; Pray translates such programs *automatically* to pPDA), and (2) explicit PPSs in the syntax of the pre-existing PReMo tool [142].[37] If Pray terminates successfully, it outputs a *rational* inductive upper bound on the LFP of the fundamental PPS of the input program's pPDA model, or on the LFP of the explicitly given PPS. The bound's absolute target precision $\varepsilon > 0$ (per component) is configurable; we used $\varepsilon = 10^{-3}$ in our experiments.

[142]: Wojtczak et al. (2007), "PReMo: An Analyzer for Probabilistic Recursive Models".

37: PReMo approximates the LFPs of PPSs associated to various types of probabilistic recursive models. It does, however, not address the computation of inductive upper bounds (certificates); thus, we shall not directly compare against it in our experiments.

Main Routine Upon invocation, Pray proceeds through the following four stages:

(1) It parses the input and, if the latter is a program, constructs a pPDA model along with the associated (cleaned-up) fundamental PPS.

(2) It computes a decomposition of the PPS into strongly connected[38] components using standard algorithms from the JGraphT library [102].
(3) It applies Algorithm 5.1—as detailed in the next paragraph—to the individual strongly connected PPSs in bottom-up order and combines the results as suggested by Proposition 5.8 on page 128.
(4) If the previous step succeeds, Pray verifies the resulting floating point certificate using *exact rational number arithmetic* (see Remark 5.18 for more details).

38: See Definition 5.3 on page 126.

[102]: Michail et al. (2020), "JGraphT—A Java Library for Graph Data Structures and Algorithms".

OVI Implementation The heart of Pray is an implementation of Algorithm 5.1, which the tool instantiates concretely as follows:

- For eigenvector approximation (i.e., the `approxEV` subroutine), it relies on power iteration.[39]
- To approximate lower bounds (`improveLB`), it employs *Gauss-Seidel iteration* [141, Section 5.2]—an optimized variant of Kleene iteration[40]—which provides a good trade-off between ease of implementation and efficiency [141].
- The damping constants are fixed to C = 0.1 and D = 0.5.
- To ensure termination on all instances, the tool aborts the main loop after at most 10 guesses; it reports failure in such cases.
- Pray implements Algorithm 5.1 and its subroutines with 64 bit floating point arithmetic.[41] Using exact rational arithmetic here might blow up the involved fractions and would thus be impractical.

39: See Section 5.3.1.2 on page 136.

[141]: Wojtczak (2009), "Recursive probabilistic models : efficient analysis and implementation".

40: See Section 5.3.1.1 on page 135.

41: We use Java's built-in `double` type.

For additional low-level implementation details (e.g., concrete data structures for polynomials, matrices, etc.), I refer to the source code [133].

▷ **Remark 5.18** (Exact vs floating point arithmetic) As mentioned, our implementation of OVI uses *approximate* floating point arithmetic for efficiency. It is, however, crucial to understand that *approximate arithmetic may—and in practice often does—lead to unsound results*; the reason is that the inductivity condition $f(u) \leq u$[42] may hold when evaluated using floats, but not when evaluated exactly (i.e., the condition is actually violated). As a remedy, Pray verifies the floating point output u of Algorithm 5.1 using *exact* (i.e., arbitrary-precision) rational arithmetic, which can be costly[43] but guarantees soundness (see our conference paper [134] for additional technical details). We treat an unsuccessful exact verification step as a failure of the overall algorithm (indicated with a cross (✗) in Tables 5.1 and 5.2). ◁

42: See Algorithm 5.1 of Algorithm 5.1 on page 134.

43: See, for example, column $Time_{\mathbb{Q}}$ in Table 5.2 on page 145.

5.4.1.2 *Implementation of the Baseline Approaches*

To the best of our knowledge, we were the first to explicitly address the problem of finding ε-close, *inductive* upper bounds for PPSs [134]. Nonetheless, there are (at least) two evident alternative approaches against which we compare our novel OVI variant; we now describe how we implemented these baseline methods.

[134]: Winkler et al. (2023), "Certificates for Probabilistic Pushdown Automata via Optimistic Value Iteration".

SMT Solving The first baseline approach encodes the problem as an SMT query and relies on off-the-shelf solvers. Concretely, given a PPS $\boldsymbol{f}$ over variables $x_1, \ldots, x_n$, we first compute a lower bound $\boldsymbol{\ell} \leq \mu\boldsymbol{f}$ using a standard iterative method—specifically, Gauss-Seidel iteration as implemented in Pray.[44] We then query the SMT solver for a *model* (i.e., a variable assignment) satisfying the following quantifier-free first-order logic formula over the free variables $x_1, \ldots, x_n$, where f_i denotes the x_i-component of $\boldsymbol{f}$:

44: See Section 5.4.1.1 on page 140.

$$\varphi_f(x_1, \ldots, x_n) \overset{\text{def}}{=} \bigwedge_{i=1}^{n} \Big(\underbrace{x_i \geq 0}_{\text{nonneg.}} \wedge \underbrace{f_i(x_1, \ldots, x_n) \leq x_i}_{\text{inductivity}} \wedge \underbrace{\boldsymbol{\ell}_i \leq x_i \leq \boldsymbol{\ell}_i + \varepsilon}_{\varepsilon\text{-closeness}} \Big) .$$

The formula φ_f belongs to the (decidable) theory of *nonlinear real arithmetic*.[45] If a model $\boldsymbol{u} \colon \{x_1, \ldots, x_n\} \to \mathbb{R}_{\geq 0}$ of φ_f exists, then $\mu\boldsymbol{f} \leq \boldsymbol{u}$ and $\|\boldsymbol{\ell} - \boldsymbol{u}\|_\infty \leq \varepsilon$; that is, $\boldsymbol{u}$ is an ε-close inductive upper bound. If no model exists, we improve the lower bound $\boldsymbol{\ell}$ and retry. We repeat this process until either a model is found or a timeout occurs (set to 10 minutes in our experiments). We have implemented this approach using the state-of-the-art solvers cvc5 [10] and z3 [104].

45: This theory is known as *QF_NRA* in the SMT community.

A Simpler OVI Variant Our second baseline is a simpler variant of Algorithm 5.1 that employs a more straightforward heuristic for generating candidate upper bounds. This variant does not approximate the Perron-Frobenius eigenvector to guide its guesses; instead, it computes each candidate upper bound $\boldsymbol{u}$ using the—less informed, but simpler—rule

$$\boldsymbol{u} \;=\; (1 + \varepsilon) \cdot \boldsymbol{\ell} \;,$$

where $\boldsymbol{\ell}$ is the current lower bound. In other words, each candidate is obtained by scaling up the current lower bound by a factor of $1 + \varepsilon$. This approach is inspired by the original OVI algorithm [63];

notice that it yields bounds within *relative* distance ε to the LFP (unlike Algorithm 5.1, which works with absolute precision).[46]

46: All our benchmark PPSs (see Section 5.4.2.1) have LFPs bounded by 1; the relative bounds are thus at least as tight as the absolute bounds.

5.4.2 *Experimental Evaluation*

Our experimental evaluation aims to assess how well our algorithm scales, how its performance compares to the baseline approaches, and whether it is suitable for PPSs with different characteristics (e. g., with dense vs sparse polynomials).

5.4.2.1 *Benchmarks*

We consider two sets of benchmarks: one containing the fundamental PPSs of pPDA models for several probabilistic recursive programs (Table 5.1), and a second one comprising explicitly given PPSs (Table 5.2) encoding stochastic grammars. All the benchmark PPSs are feasible—their LFPs are at most $\mathbf{1}$—and have *rational* coefficients only.[47] The only degenerate instance we are aware of is rw-0.500 (Table 5.1); we strongly conjecture that all other examples are nondegenerate. We now describe the two benchmark sets in more detail.

47: Recall that the LFP of such a PPS might be irrational nonetheless (see e. g., Example 4.4 on page 92).

Probabilistic Recursive Program Benchmarks (Table 5.1) Our tool Pray translates the programs in this benchmark set to *finite* pPDA—finite because the programs operate only on finite data domains (e. g., Booleans, bounded integers)—and constructs the associated fundamental PPSs. In Chapter 7, we shall describe a general translation from probabilistic recursive programs to pPDA in detail.[48] Suffice it to say here that the central idea is standard—we encode the program's call stack via the automaton's pushdown stack—and the translation takes at most a few seconds for the programs in our benchmark set.

48: The version of Pray used in the experiments implements a variant of the translation described in Chapter 7, but the basic principle is the same.

The programs golden, and-or (see Figure 5.7), virus, and gen-fun are based on examples from the literature [22, 105, 124, 131].[49] We designed the instances rw-p, geom-offspring, escape$N$, sequential$N$, and mod$N$ ourselves to cover a variety of additional scenarios. Notice that most of the latter examples depend on a parameter (p or N). The full source code of all nine programs is included in Appendix A.[50] The considered programs all feature unbounded recursion;[51] they thus induce *infinite* Markov chains.

49: Specifically, gen-fun is based on Program 5.6 in van de Meent et al.'s book [124].

50: See Figure A.2 on page 220.

51: We speak of *unbounded recursion* if arbitrarily deep nestings of recursive calls occur with positive probability.

The PPSs in this benchmark set vary in size (38–26,367 variables), are *sparse* (at most 1.7 terms per polynomial on average), and most

Table 5.1: Experimental results on PPSs from pPDA models of recursive probabilistic programs with precision $\varepsilon = 10^{-3}$. Columns: *States*, *Trans.*, *Syms.*: pPDA states, nonzero probability transitions, stack symbols; *Vars.*, *Terms*: dimension and total terms of the pPDA's fundamental PPSs; *SCCs*, $C_{\max}$: number of strongly connected nontrivial sub-PPSs and largest component size; *G.*: total number of guesses made by the two OVI-based algorithms; *Dgts.*: average decimal digits in rational certificate; *Time*: runtime (ms) excluding pPDA/PPS construction; $Time_{\mathbb{Q}}$: percentage of *Time* for exact rational arithmetic. Checkmark (✓): successful inductive upper bound generation; cross (✗): failure due to timeout (TO; 10 min), floating-point to rational conversion failure (see Remark 5.18 on page 141), or exceeding allowed guesses (at most 10 per component).

Benchmark	pPDA Data			PPS Data				Our OVI (PRAY)					z3 Baseline			OVI Baseline			
	States	Trans.	Syms.	Vars.	Terms	SCCs	$C_{\max}$	✓/✗	G.	Dgts.	Time	$Time_{\mathbb{Q}}$	✓/✗	Dgts.	Time	✓/✗	G.	Dgts.	Time
rw-0.499	18	29	5	38	45	1	12	✓	5	**5**	163	17%	✓	**2**	**11**	✓	4	**5**	59
rw-0.500	18	29	5	38	45	1	12	✗	10	–	7,327	–	✓	**2**	**10**	✗	10	–	8,083
rw-0.501	18	29	5	38	45	1	12	✓	5	**4**	36	6%	✓	13	**12**	✓	4	5	23
geom-offsp.	24	40	5	52	80	4	24	✓	8	**6**	15	13%	✓	9	16	✓	8	**6**	**14**
golden	27	49	6	81	94	1	36	✓	1	5	**10**	30%	✓	7	14	✓	2	**4**	12
and-or	50	90	7	149	182	1	48	✓	2	**4**	**19**	26%	✓	12	15,260	✓	2	**4**	**19**
gen-fun	85	219	7	202	327	1	16	✓	2	**3**	22	32%	✓	15	141	✓	2	**3**	**21**
virus	68	149	27	341	551	1	220	✓	1	**5**	**40**	38%	✓	7	139	✓	1	6	59
escape10	109	174	23	220	263	1	122	✓	1	**4**	56	5%	✓	7	**48**	✓	1	8	71
escape25	258	413	53	518	621	1	300	✓	1	**5**	245	17%	✓	7	15,958	✓	1	9	**172**
escape50	508	813	103	1,018	1,221	1	600	✓	1	**7**	653	23%	✓	7	**410**	✗	1	–	400
escape75	760	1,215	153	1,522	1,825	1	904	✓	2	**9**	**3,803**	10%	✗	–	TO	✗	1	–	635
escape100	1,009	1,614	203	2,020	2,423	1	1,202	✗	5	–	29,027	–	✓	**6**	**939**	✗	1	–	901
escape200	2,008	3,213	403	4,018	4,821	1	2,400	✗	6	–	83,781	–	✗	–	TO	✗	1	–	2,206
sequential5	230	490	39	1,017	1,200	10	12	✓	15	**4**	**103**	26%	✓	8	1,074	✓	15	5	204
sequential7	572	1,354	137	3,349	3,856	14	12	✓	21	**5**	1,049	27%	✓	8	12,822	✓	20	**5**	**1,042**
sequential10	3,341	8,666	1,036	26,367	29,616	20	12	✓	30	**5**	**100,613**	2%	✓	8	453,718	✓	30	6	101,554
mod5	44	103	10	296	425	1	86	✓	1	**5**	**28**	39%	✓	9	34,150	✗	2	–	178
mod7	64	159	14	680	1,017	1	222	✓	1	**6**	**172**	69%	✓	7	443	✗	2	–	624
mod10	95	244	20	1,574	2,403	1	557	✗	1	–	675	–	✓	**7**	**1,245**	✗	2	–	882

of them have just one nontrivial strongly connected component (the sequential$N$ instances constitute a notable exception).

Explicit PPS Benchmarks (Table 5.2) Almost all PPSs in the second benchmark set originate from the literature [142] and correspond to *stochastic context-free grammars* (SCFGs), which are equivalent to pPDA with a single state.[52] The instances brown, lemonde, negra, swbd, tiger, tuebadz, and wsj relate to problems from the area of language processing.[53] random is the fundamental system of a randomly generated SCFG with 10,000 nonterminals and approximately two production rules per nonterminal on average.

Most of the benchmarks in this set contain *dense* polynomials (at least 45 terms per polynomial on average); the only exception is the random instance, which is sparse with only approximately 2 terms per polynomial on average.

[142]: Wojtczak et al. (2007), "PReMo: An Analyzer for Probabilistic Recursive Models".

52: One-state pPDA are known in the literature as *probabilistic basic process algebras*; see e. g., Brázdil et al.'s survey [21].

53: The PReMo paper [142] provides more details on these examples.

5.4.2.2 *Experimental Results*

We invoked PRAY[54] and the two baseline implementations[55] on each

54: See Section 5.4.1.1 on page 140.

55: See Section 5.4.1.2 on page 142.

Table 5.2: Experimental results with PPSs corresponding to stochastic grammars (setup and columns as in Table 5.1).

Benchmark	PPS Data				Our OVI (Pray)					z3 Baseline			OVI Baseline			
	Vars.	Terms	SCCs	C_{max}	✓/✗	G.	Dgts.	Time	$Time_{\mathbb{Q}}$	✓/✗	Dgts.	Time	✓/✗	G.	Dgts.	Time
brown	37	22,866	1	22	✓	2	**6**	**3,212**	74%	✗	-	TO	✓	2	8	9,065
lemonde	121	32,885	1	48	✓	2	**5**	40,738	97%	✗	-	TO	✓	2	**5**	**38,107**
negra	256	29,297	1	149	✓	2	7	10,174	89%	✓	**1**	37,248	✓	1	7	**8,873**
swbd	309	47,578	1	243	✓	1	**7**	**18,989**	93%	✗	-	TO	✓	1	8	67,314
tiger	318	52,184	1	214	✓	2	8	94,490	98%	✓	**1**	**17,454**	✓	1	8	90,801
tuebadz	196	8,932	2	168	✓	4	9	**2,666**	85%	✓	**1**	15,323	✓	3	9	2,700
wsj	240	31,170	1	194	✓	2	**9**	30,275	96%	✗	-	TO	✓	2	**9**	**29,038**
random	10,000	20,129	1	8,072	✓	3	**7**	17,585	5%	✗	-	TO	✓	4	8	**16,357**

instance (with different parameters, if applicable) from our two benchmark sets.[56] Tables 5.1 and 5.2 display the results achieved on a laptop with a 2.4 GHz Quad-Core Intel Core i5 processor and 16 GB of RAM. Note that, regarding the SMT baseline approach, we present only the results obtained with z3 in the tables—the results achieved with cvc5 are mostly not competitive on the considered benchmarks; see Section A.2.[57]

56: See Section 5.4.2.1 on page 143.

57: On a few instances, cvc5 was faster than z3, but overall, z3 solved significantly more instances within the time limit. Nonetheless, more examples would be needed to conclude that z3 is strictly better than cvc5 for this specific problem.

Results: Program Benchmarks (Table 5.1) Both Pray and the SMT-based approach with z3 successfully compute inductive upper bounds for the majority of examples within the 10-minute time limit (Pray: 16/20 instances; z3: 18/20 instances). Pray and z3 require at most 101 and 454 seconds, respectively, on the examples where they succeed. The average runtime on the successful instances is under 7 seconds for Pray, and approximately 30 seconds for z3. It is worth noting that, among the PPSs solved by any of the tools, only z3 solves the one with the largest component (escape100 with 1,202 variables).

The OVI baseline method (rightmost column in Table 5.1) solves only a strict subset of both the instances that Pray's OVI can handle, and of those that z3 can solve. On the instances where the OVI baseline succeeds, its runtime is generally comparable to that of Pray.

Results: SCFG Benchmarks (Table 5.2) For the SCFG benchmarks (Table 5.2), the results differ considerably: Pray successfully solves all instances, whereas z3 solves only 3 out of 8 within the time limit (z3 returns the constant inductive upper bound **1** for these three instances). Notably, on most examples, our tool spends a significant amount of time—up to 97%—performing computations with exact rational numbers; this effect occurs because evaluating dense polynomials exactly leads to large numerators and denominators.

The simpler OVI baseline method performs fairly well on this benchmark set: It also succeeds on all 8 instances, with a runtime similar to that of Pray's OVI (notable exceptions are the brown and swbd instances, where Pray is approximately three times faster).

Discussion The experimental results suggest that OVI is a promising approach for computing inductive upper bounds in PPSs: Our prototypical implementation handles sparse PPSs generated by the pPDA benchmarks—with up to hundreds of states and stack symbols—within minutes,[58] and it scales to dense PPSs encoding stochastic grammars with tens of thousands of production rules. One of the most impressive benchmarks is random, which has 10,000 variables (over 8,000 of which belong to a single strongly connected component); both OVI methods terminate on this example in less than 20 seconds. Overall, the performance of Pray's OVI variant is slightly better than that of the simpler OVI baseline: The former solves more instances and usually delivers more succinct certificates (i.e., rational numbers with smaller numerators and denominators). Thus, the more sophisticated guessing heuristic based on eigenvector approximation pays off not only in theory, but also in practice.

58: These pPDA are machine-constructed models of probabilistic recursive programs; see Section 5.4.2.1.

The SMT-based approach performs well on sparser PPSs, but struggles with denser ones with numerous (order of magnitude 10^5 to 10^6) terms. We thus conjecture that, for sparse instances with a moderate number of variables (approximately 10^2 to 10^3), SMT solving is a reasonable choice—it is essentially off-the-shelf and more robust against degeneracy—whereas for larger, especially dense instances, OVI is more suitable.

We also found the performance of OVI to be significantly more predictable than that of z3, which sometimes varies greatly (see, e.g., the escape*N* benchmarks in Table 5.1, where runtime and instance size appear rather unrelated). This unpredictability potentially hinders the practical applicability of z3 on this class of problems.

5.5 *Open Questions*

1. Can we extend OVI to detect—and possibly even certify—infeasibility or degeneracy of the input PPS?
2. How common are degenerate PPSs in practice?[59] (Recall from Proposition 4.25 on page 108 that degenerate PPSs do not necessarily have any rational inductive upper bounds at all.)

59: Among the benchmark PPSs in Section 5.4.2.1, only one instance is degenerate (rw-0.500), and this degeneracy is by design. However, a broader collection of real-world examples would be needed to conclude that degeneracy is rare in practice.

3. Are there any additional practical examples on which we could evaluate OVI? Would the conclusions drawn from our experiments in Section 5.4.2.2 continue to hold on an expanded benchmark set?
4. Obtaining an exact rational inductive upper bound from a floating point approximation is challenging and does not always succeed;[60] in fact, this conversion step is still a major bottleneck of the current implementation. Could we adopt more specialized strategies to improve the success rate of the conversion (e.g., the *smoothing* and rounding techniques described in [26, Section 6])?

60: See Remark 5.18 on page 141.

[26]: Chatterjee et al. (2025), "Fixed Point Certificates for Reachability and Expected Rewards in MDPs".

5.6 Additional Bibliographic Remarks

The original OVI for Markov decision processes [63] was well received in the probabilistic verification community; today, the algorithm ranks among the state-of-the-art techniques for reliably solving such models [65]. Since its introduction in 2020, researchers have developed variants of OVI for other models, including *simple stochastic games* [7] and *expected visiting times* in DTMCs [101].

[63]: Hartmanns et al. (2020), "Optimistic Value Iteration".

[65]: Hartmanns et al. (2025), "The revised practitioner's guide to MDP model checking algorithms".

[7]: Azeem et al. (2022), "Optimistic and Topological Value Iteration for Simple Stochastic Games".

[101]: Mertens et al. (2025), "Computing Expected Visiting Times and Stationary Distributions in Markov Chains: Fast and Accurate".

The idea of analyzing PPSs in a modular fashion (i.e., processing each strongly connected component separately; see Section 5.1) rather than monolithically is due to Etessami and Yannakakis [49]. Analogous techniques are also common in verification algorithms for Markov decision processes; in this context, researchers often speak of *topological analyses*, a term referring to the topological order of the DAG of strongly connected components that such algorithms traverse in a bottom-up manner [33].

[49]: Etessami et al. (2005), "Recursive Markov Chains, Stochastic Grammars, and Monotone Systems of Nonlinear Equations".

[33]: Ciesinski et al. (2008), "Reduction Techniques for Model Checking Markov Decision Processes".

Cone vectors, which include Perron-Frobenius eigenvectors[61] as a special case, play a central role in Esparza, Kiefer, and Luttenberger's analysis of Newton's method for PPSs [43]. The same authors also provide a method for deducing *guaranteed upper bounds* from the lower bounds computed by Newton's method [43, Section 5.4]; investigating whether this technique (or a variant thereof) yields inductive upper bounds appears promising.

61: See Lemma 5.11 on page 130.

[43]: Esparza et al. (2010), "Computing the Least Fixed Point of Positive Polynomial Systems".

6 Model Checking Temporal Pushdown Specifications

6.1 Preliminaries . 153
6.1.1 VPLs and VPA 153
6.1.2 Stair Parity DVPA 157
6.1.3 CaRet 161
6.2 pVPA 163
6.3 Checking pVPA vs Stair Parity DVPA 167
6.3.1 Products of VPA 167
6.3.2 Stair Parity Acceptance Probabilities . . 168
6.3.3 Reduction to ETR 177
6.4 Checking pVPA vs Büchi VPA and CaRet . . . 182
6.5 Open Questions 184
6.6 Additional Bibliographic Remarks 184

In Chapters 3 and 4, we have studied rather elementary properties of pPDA (probabilistic pushdown automata), such as termination probabilities and expected runtimes. In this chapter, we consider a more expressive class of *temporal specifications* tailored to the structured computations arising from pushdown systems.

We begin by illustrating and motivating temporal properties of recursive probabilistic processes by means of an example.

Motivating Example: Modeling an Epidemic We consider a toy epidemiological model for the outbreak of an infectious disease in a large population where the number of susceptible individuals can be assumed to be infinite. Our model distinguishes between young and elderly persons. Each infected individual transmits the disease to a uniformly distributed number of others, with varying rates (expected values) depending on the age groups (Table 6.1). The fatality rate for infected elderly persons is 1%, while for young persons it is 0%. This model can be readily specified as a probabilistic program with mutually recursive procedures (Figure 6.1). Procedure `elder()` returns a Boolean to signal to its callers whether the infection has resulted in a fatality. Initially, we assume there is a single infected young person, that is, the program is started by calling `young()`.

	Y	E
Y	1.5	1.0
E	0.5	2.0

Table 6.1: Example infection rates (i. e., expected number of infections caused) by age groups. Y: young, E: elderly.

As already mentioned in Section 5.4.2.1, recursive probabilistic programs such as the one from Figure 6.1 can be modeled naturally as finite-state pPDA.[1] We detail a concrete translation from programs to pPDA in Chapter 7.

In this chapter, we study *temporal* properties, which, in our example, are useful for specifying the chronological order of certain events during the epidemic. An example are chains of infection—for instance, we might ask:

> *What is the probability that eventually a young person—whose entire chain of infection consists only of young persons—passes the virus to an elderly person who then dies?*

At the level of the program in Figure 6.1, this question asks for the probability of reaching a configuration where (1) the call stack contains only invocations of `young`, and (2) during the execution of the current `young`, the local variable `f` becomes true (line 8).

1: In fact, the example program from Figure 6.1 was one of the benchmarks in our experimental evaluation of the OVI algorithm in Chapter 5.

```
1  void young() {
2    int y = uniform(4);
3    while(y > 0) {
4      young();
5      y = y-1; }
6    int e = uniform(3);
7    while(e > 0) {
8      bool f = elder();
9      e = e-1; }
10 }
```

```
11 bool elder() {
12   int y = uniform(2);
13   while(y > 0) {
14     young();
15     y = y-1; }
16   int e = uniform(5);
17   while(e > 0) {
18     elder();
19     e = e-1; }
20   /* fatality rate for
21   elderly is 1% */
22   return flip(0.01);
23 }
```

Figure 6.1: Recursive probabilistic program modeling the outbreak of an infectious disease. `uniform`(n) denotes a uniform sample from the integer interval $[0..n-1]$.

Crucially, this property requires reasoning about the nesting of procedure calls and returns: To determine whether `f` will become `true` within a given invocation of `young`, we must consider only the return values of `elder` calls made directly from that invocation; return values from deeper nested calls to `elder` are irrelevant.

CaRet: A Temporal Logic of Nested Calls and Returns The example specification from the previous paragraph can be naturally expressed in *CaRet* [2], an extension of *Linear Temporal Logic* (LTL) [108]. Concretely, the property is captured by the following CaRet formula:

$$\varphi \;=\; \Diamond^{\mathrm{g}}\big(\,\Box^{-} p_{\texttt{young}} \;\wedge\; p_{\texttt{young}} \;\wedge\; \Diamond^{\mathrm{a}} f\,\big)\,.$$

The idea is that the *atomic proposition* $p_{\texttt{young}}$ holds at states[2] that correspond to being inside procedure `young`, and f indicates that `f = true`. Intuitively, φ encodes that eventually (outer $\Diamond^{\mathrm{g}}$) the computation reaches a configuration where only `young` is on the call stack and the current procedure is `young` ($\Box^{-} p_{\texttt{young}} \wedge p_{\texttt{young}}$), and moreover the local—also called *abstract*—path within the current procedure invocation reaches a state where `f` is true ($\Diamond^{\mathrm{a}} f$).

Due to its ability to reason about abstract paths jumping from calls to their matching returns, CaRet is strictly more expressive than LTL. More precisely, CaRet-definable properties are *ω-visibly pushdown languages* (VPLs) [3], a class strictly between ω-regular and ω-context-free languages (Figure 6.3).

[2]: Alur et al. (2004), "A Temporal Logic of Nested Calls and Returns".

[108]: Pnueli (1977), "The Temporal Logic of Programs".

2: That is, the states of an appropriate pPDA model of our program are labeled with these atomic propositions; we do not explicitly construct this model here.

[3]: Alur et al. (2004), "Visibly pushdown languages".

Problem Statement and Results In this chapter, we develop a solution to the following probabilistic model checking problem:

Table 6.2: Complexity results established in this chapter. The *qualitative* variants ask whether the probability of satisfying the specification equals 1, whereas the *quantitative* problems ask—more generally—whether it is at least a given threshold. See Definitions 6.18 and 6.19 on page 166 for the precise problem statements.

Type of specification	Qualitative	Quantitative	Reference
Deterministic stair parity VPA	in PSPACE	in PSPACE	Theorem 6.32
Nondeterministic Büchi VPA	EXPTIME-compl.	in EXPSPACE	Theorem 6.36
CaRet formula	in 2EXPTIME	in 2EXPSPACE	Theorem 6.37

Given a finite pPDA[3] Δ with state labels and an ω-VPL $\mathcal{L}$ specified either as a CaRet formula or a suitable automaton, is the probability that the trace (sequence of labels) of a random trajectory of Δ falls within $\mathcal{L}$ at least a given threshold?

3: With a mild restriction necessary to ensure decidability; see paragraph *Outline of our Approach* and Section 6.2 for details.

The complexity results for the associated decision problems are summarized in Table 6.2.

Outline of our Approach To obtain the results in Table 6.2, we combine several existing constructions from the literature; see Figure 6.2 for an overview.

First, for decidability, we impose a mild *visibility* restriction on the given state-labeled pPDA Δ, namely that the the type of the label—either, *call*, *internal*, or *return*—prescribes the stack action of the outgoing transitions at every state [39].[4] We call pPDA restricted in this way *probabilistic visibly pushdown automata* (pVPA).

[39]: Dubslaff et al. (2012), "Model checking probabilistic systems against pushdown specifications".

4: Dubslaff, Baier, and Berg [39] actually use a slightly different criterion; see Remark 6.20.

We then proceed similarly to the usual automata-based approach.[5] We translate CaRet specifications φ into nondeterministic *visibly pushdown automata* (VPA) [3] $\mathcal{A}$ with Büchi acceptance, which we determinize using a procedure by Löding, Madhusudan, and Serre [90]. The resulting deterministic VPA (DVPA) $\mathcal{D}$ use *stair parity* [90] acceptance conditions, which are strictly more expressive than standard parity on DVPA [3]. Stair parity differs from usual parity by considering only certain positions—called *steps* [90]—from which onward the stack height never decreases again.

5: See, e. g., [8, Section 10.3].

[3]: Alur et al. (2004), "Visibly pushdown languages".

[90]: Löding et al. (2004), "Visibly Pushdown Games".

Once we have converted the specification to a DVPA $\mathcal{D}$, we construct an automata-theoretic product $\Delta \times \mathcal{D}$, where visibility ensures synchronization of stack actions, yielding a single-stack product automaton (two independent stacks would lead to undecidability [39]). Since $\mathcal{D}$ is deterministic, the product automaton $\Delta \times \mathcal{D}$ is again a pPDA. To compute the stair parity acceptance probability, we construct a finite DTMC $\mathcal{M}_{\Delta \times \mathcal{D}}$, which we call the *step chain* in this chapter [44].[6] Intuitively, the step chain *jumps* from one step to the

[44]: Esparza et al. (2004), "Model Checking Probabilistic Pushdown Automata".

6: The idea of step chains was introduced by Esparza, Kucera, and Mayr [44] for model checking pPDA against deterministic Büchi automata.

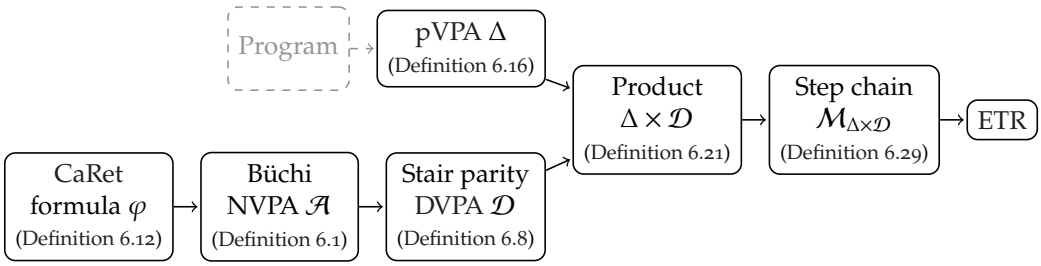


Figure 6.2: Chain of reductions used in this chapter. ETR is short for *existential theory of the reals*, the existentially quantified fragment of the first-order theory over $(\mathbb{R}, +, \cdot, \leq)$.

next, thereby reducing the overall problem to computing *standard parity* acceptance probabilities in a DTMC.

The transition probabilities of the step chain depend on the *return probability triples* $[qZr]$ of the product pPDA $\Delta \times \mathcal{D}$. Recall from Chapter 3 that the triples $[qZr]$ are the least solution of a PPS (positive polynomial system). The latter can be encoded in the *existential theory of the reals* (ETR) [86]. Our complexity results ultimately rely on the fact that ETR is decidable in PSPACE [23].

[86]: Kucera et al. (2006), "Model Checking Probabilistic Pushdown Automata".

[23]: Canny (1988), "Some Algebraic and Geometric Computations in PSPACE".

Notably, two crucial constructions in our approach—the determinization of Büchi VPA [90] via stair parity and the step chain construction [44]—were developed independently and for different purposes. One of our key insights is that these two constructions can be combined to solve our model checking problems.

Chapter Outline, Contributions, and Origins This chapter covers our FoSSaCS 2022 conference paper [131] and its subsequent journal version [132], extending the two papers with additional examples and improved explanations and proofs. The structure is as follows:

[131]: Winkler et al. (2022), "Model Checking Temporal Properties of Recursive Probabilistic Programs".

Section 6.1 (Preliminary Definitions and Results) We present the definitions of (D)VPA and CaRet from the literature and state the known translations between these formalisms.

Section 6.2 (Probabilistic Visibly Pushdown Automata) We define pVPA and formally state the model checking problems from Table 6.2.

Section 6.3 (Checking pVPA vs Stair Parity DVPA) We present our model checking procedure for pVPA against stair parity DVPA specifications, which constitutes the main technical content of this chapter.

Section 6.4 (Checking pVPA vs Büchi VPA and CaRet) We derive complexity results for checking pVPA against Büchi VPA and CaRet specifications as consequences of the result from the previous section.

See Section 6.5 for open questions and Section 6.6 for additional bibliographic remarks and related works.

6.1 Preliminary Definitions and Results

In this section, we summarize the relevant existing results on visibly pushdown languages and their corresponding automata models. We also present the syntax and semantics of CaRet [2].

We recall the following standard notations for words and languages: Given an alphabet Σ, we denote by Σ^* the set of all finite words (including the empty word ϵ), and by Σ^ω the set of all infinite words over Σ. We write $|w| \in \overline{\mathbb{N}} = \mathbb{N} \cup \{\infty\}$ for the length of a word $w \in \Sigma^* \cup \Sigma^\omega$, and $w(i)$ for the ith symbol of w (for $0 \leq i < |w|$). We define $\Sigma^n = \{w \in \Sigma^* \mid |w| = n\}$ for $n \in \mathbb{N}_0$. For sets $A \subseteq \Sigma^*$ and $B \subseteq \Sigma^* \cup \Sigma^\omega$, we write $A \cdot B$ for their concatenation, using $a \cdot B$ and $A \cdot b$ as shorthands for $\{a\} \cdot B$ and $A \cdot \{b\}$, respectively.

6.1.1 Visibly Pushdown Languages and Automata

A *pushdown alphabet*[7] is a finite alphabet Σ equipped with a partition $\Sigma = \Sigma_{\text{call}} \uplus \Sigma_{\text{int}} \uplus \Sigma_{\text{ret}}$ into three—possibly empty—subsets of *call*, *internal*, and *return* symbols. In the examples in this chapter, we often use the singleton alphabets $\Sigma_{\text{call}} = \{c\}$, $\Sigma_{\text{int}} = \{\tau\}$, and $\Sigma_{\text{ret}} = \{r\}$.

A *visibly pushdown automaton* (VPA)[8] is similar to a traditional (nondeterministic) pushdown automaton but with additional syntactic restrictions—the input alphabet must be a pushdown alphabet and, crucially:

> *Reading a call symbol must trigger a push transition, reading an internal symbol must trigger an internal transition, and reading a return symbol must trigger a pop transition.*

The stack action is thus *visible* in the input symbol—hence the name. We define VPA formally as follows:[9]

7: Pushdown alphabets are not to be confused with *stack alphabets*—the former specify *input* symbols whereas the latter provide the symbols that can be pushed onto or popped from the stack.

8: VPA are also called *input-driven* pushdown automata in the literature [99].

9: We deliberately do not introduce VPA as an instance of the abstract framework from Chapter 3; see Remark 6.4 for a discussion.

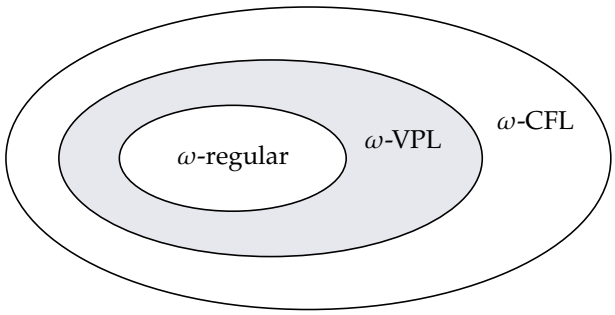


Figure 6.3: Inclusion hierarchy of ω-languages. ω-VPLs lie strictly in between ω-regular languages and ω-context-free languages (ω-CFLs).

Definition 6.1 ***(VPA: Visibly pushdown automaton*** [3]**)** Let $\Sigma = \Sigma_{\text{call}} \uplus \Sigma_{\text{int}} \uplus \Sigma_{\text{ret}}$ be a pushdown alphabet. A VPA over Σ is a tuple $\mathcal{A} = (S, s_0, \Gamma, \bot, \delta, \Sigma)$, where S is a finite set of states, $s_0 \in S$ is an initial state, Γ is a finite stack alphabet, $\bot \in \Gamma$ is a special bottom-of-stack symbol, and $\delta = (\delta_{\text{call}}, \delta_{\text{int}}, \delta_{\text{ret}})$ is a triple of relations

$$\begin{aligned} \delta_{\text{call}} &\subseteq (S \times \Sigma_{\text{call}}) \times (S \times \Gamma_{\text{-}\bot}) \,, \\ \delta_{\text{int}} &\subseteq (S \times \Sigma_{\text{int}}) \times S \,, \\ \delta_{\text{ret}} &\subseteq (S \times \Sigma_{\text{ret}} \times \Gamma) \times S \,, \end{aligned}$$

where $\Gamma_{\text{-}\bot} \stackrel{\text{def}}{=} \Gamma \setminus \{\bot\}$. △

See Figures 6.4, 6.5 and 6.7 on page 156, on page 158 and on page 160, respectively, for examples of VPA.

Let us define some additional basic notation and terminology. Throughout the following paragraphs, let $\mathcal{A} = (S, s_0, \Gamma, \bot, \delta, \Sigma)$ be a VPA as in Definition 6.1. For states $s, t \in S$, a stack symbol $Z \in \Gamma$, and an input symbol $a \in \Sigma$, we write

$$s \xrightarrow{a} tZ \,, \quad s \xrightarrow{a} t \,, \quad \text{and} \quad sZ \xrightarrow{a} t$$

to indicate that there exist transitions $(s, a, t, Z) \in \delta_{\text{call}}$, $(s, a, t) \in \delta_{\text{int}}$, and $(s, a, Z, t) \in \delta_{\text{ret}}$, respectively.[10] We define the semantics of VPA in the usual way: A *configuration* of $\mathcal{A}$ is a tuple

$$(s, \gamma) \in S \times (\Gamma_{\text{-}\bot}^* \cdot \bot) \,,$$

written more succinctly as $s\gamma$ in the sequel. Intuitively, being in configuration $s\gamma$ means that the automaton is currently in state s and has the word γ on its stack. Notably, the special bottom-of-stack symbol $\bot$ appears exactly once in every configuration, namely at the bottom of the stack. As in Chapter 3, if $s\gamma = sZ\gamma'$ for some $Z \in \Gamma$, then we call sZ the *head* of $s\gamma$. A *bottom configuration* is a configuration with head $s\bot$ for some $s \in S$.

10: This notation is analogous to the one introduced in Chapter 3 (Section 3.1.1) and reflects that internal and call transtions cannot access the current topmost stack symbol in VPA.

Let $w \in \Sigma^\omega$ be an infinite[11] input word. An infinite sequence

$$\rho \;=\; s_0\gamma_0\,,\; s_1\gamma_1\,,\; s_2\gamma_2\,,\; \ldots$$

of configurations is a *run* of $\mathcal{A}$ on w if $\gamma_0 = \bot$ and if exactly one of the following four cases applies to all positions $i \geq 0$:

1. *Push/call:* There exists $Z \in \Gamma_{\text{-}\bot}$ such that $\gamma_{i+1} = Z\gamma_i$ and $s_i \xrightarrow{w(i)} s_{i+1}Z$.[12]
2. *Internal:* $\gamma_{i+1} = \gamma_i$ and $s_i \xrightarrow{w(i)} s_{i+1}$.
3. *Pop/return:* There exists $Z \in \Gamma_{\text{-}\bot}$ such that $Z\gamma_{i+1} = \gamma_i$ and $s_iZ \xrightarrow{w(i)} s_{i+1}$.
4. *Pop/return at* $\bot$: $\gamma_i = \gamma_{i+1} = \bot$ and $s_i\bot \xrightarrow{w(i)} s_{i+1}$.

11: We could define runs for finite input words similarly; however, we only consider automata on infinite words (ω-automata) in this chapter.

12: By the visibility restriction, this condition implies that $w(i)$ is a call symbol, that is, an element of Σ_{call}. This remark applies similarly to the subsequent cases.

Intuitively, if $\mathcal{A}$ reads a call symbol $a \in \Sigma_{\text{call}}$ while in configuration $s\gamma$ and there exists a transition $s \xrightarrow{a} tZ$, then a run of $\mathcal{A}$ may evolve—nondeterministically in general—from $s\gamma$ to $tZ\gamma$. Similarly, if $\mathcal{A}$ reads an internal symbol $a \in \Sigma_{\text{int}}$ while in configuration $s\gamma$ and there is a transition $s \xrightarrow{a} t$, then a run may move from $s\gamma$ to $t\gamma$. For return symbols, if $\mathcal{A}$ reads $a \in \Sigma_{\text{ret}}$ in configuration $sZ\gamma$ where $Z \neq \bot$ and there is a transition $sZ \xrightarrow{a} t$, then a run can evolve from $sZ\gamma$ to $t\gamma$. Importantly, executing a return transition in a bottom configuration $s\bot$ does *not* remove the bottom-of-stack symbol $\bot$. Such transitions consume return symbols nevertheless.

A *Büchi acceptance condition* for $\mathcal{A}$ is a subset $F \subseteq S$ of states. A VPA equipped with a Büchi condition is called a *Büchi VPA*. An infinite word $w \in \Sigma^\omega$ is accepted by a Büchi VPA if there exists a run $s_0\gamma_0, s_1\gamma_1, \ldots$ of $\mathcal{A}$ on w such that $s_i \in F$ for infinitely many $i \geq 0$. The ω-language of words accepted by a Büchi VPA $\mathcal{A}$ is denoted $\mathcal{L}(\mathcal{A}) \subseteq \Sigma^\omega$.

We can now formally define the class of temporal properties whose verification we address in this chapter:

Definition 6.2 ***(ω-VPL: ω-Visibly pushdown language*** [3]***)*** Let Σ be a pushdown alphabet. A set $L \subseteq \Sigma^\omega$ of infinite words is an ω-VPL if $L = \mathcal{L}(\mathcal{A})$ for a Büchi VPA $\mathcal{A}$ over Σ. △

[3]: Alur et al. (2004), "Visibly pushdown languages".

► **Example 6.3** (Infinite words of well-matched parentheses) An infinite string of parentheses is *well-matched* if every opening parenthesis has a corresponding closing parenthesis, and vice versa. For instance, the strings

$$()()()\ldots \quad \text{and} \quad ()(())((()))\ldots$$

are well-matched, whereas $(((((\ldots$ and $)\ldots$ are not.[13]

13: Languages of well-matched (or *balanced*) parentheses words are often called *Dyck languages* in the literature.

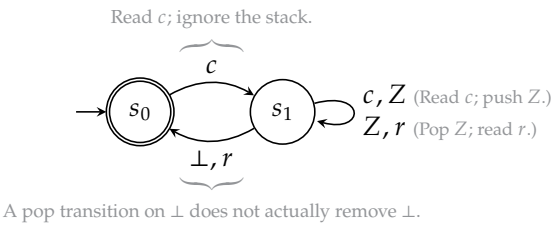


Figure 6.4: Example Büchi VPA (see Definition 6.1) accepting the language of well-matched words (c and r represent opening and closing parentheses, respectively; see Example 6.3). The VPA accepts an infinite word if and only if it has a run visiting the accepting state s_0 (doubly circled) infinitely often.

The language of all infinite well-matched words is an ω-VPL, as witnessed by the Büchi VPA in Figure 6.4. We use the pushdown alphabet with $\Sigma_{\text{call}} = \{c\}$ and $\Sigma_{\text{ret}} = \{r\}$, where c and r represent opening and closing parentheses, respectively. The automaton uses the stack to track the number of unmatched opening parentheses encountered so far. To ensure that every opening parenthesis is eventually matched, the automaton must visit the accepting state s_0 infinitely often; this state is only reachable when the stack is empty (i.e., contains only the bottom marker $\bot$). ..... ◀

A VPA is *deterministic* (DVPA) if its transition relations δ_{call}, δ_{int}, and δ_{ret} are functional;[14] in this case, we identify the relations with the corresponding functions

$$\begin{aligned}\delta_{\text{call}} &: (S \times \Sigma_{\text{call}}) \to (S \times \Gamma_{\text{-}\bot}) ,\\ \delta_{\text{int}} &: (S \times \Sigma_{\text{int}}) \to S ,\\ \delta_{\text{ret}} &: (S \times \Sigma_{\text{ret}} \times \Gamma) \to S .\end{aligned}$$

14: That is, the relations are the graphs of total functions

Crucially, a DVPA thus has *exactly one* run on each word of its input alphabet.

It is well-known that standard (non-pushdown) deterministic Büchi automata are strictly less expressive than their nondeterministic counterparts (e.g., [8, Theorem 4.50]). As a consequence, Büchi DVPA are strictly less expressive than nondeterministic Büchi VPA; that is, Büchi DVPA characterize only a proper subset of ω-VPLs.

[8]: Baier et al. (2008), *Principles of Model Checking*.

There is, however, a crucial difference between the pushdown and the non-pushdown case: Whereas nondeterministic Büchi automata are as expressive as deterministic automata with the more general *parity*[15] acceptance condition, this equivalence does not hold for VPA: Büchi VPA cannot be determinized by switching to parity acceptance; parity DVPA remain strictly less expressive than nondeterministic Büchi VPA [3, Theorem 7].

15: The parity condition assigns nonnegative integer *priorities* to the automaton's states and requires the minimum priority seen infinitely often to be even; cf. Definition 6.8 on page 159.

[3]: Alur et al. (2004), "Visibly pushdown languages".

Despite the preceding observations, Löding, Madhusudan, and Serre [90] introduced a deterministic automaton model for ω-VPL:

[90]: Löding et al. (2004), "Visibly Pushdown Games".

It relies on a *stair parity* acceptance condition, as detailed in the upcoming Section 6.1.2.

▷ **Remark 6.4** (VPA in the weighted automata framework) We could also introduce general *weighted* visibly pushdown automata using ingredients from the framework presented in Chapters 2 and 3. The natural approach would be to employ a formal power series semiring[16] $\mathcal{S}\langle\!\langle \Sigma^* \rangle\!\rangle$, where Σ is a pushdown alphabet, and impose additional syntactic restrictions on the transition weights, similar to Definition 6.1 on page 154.

16: See Definition 2.11 on page 21.

However, the resulting additional layer of abstraction would add relatively little benefit to the discussion. Therefore, in this chapter, we work outside the weighted automata framework. This also enables us to use the original definition of VPA from [3] (reproduced as Definition 6.1), which is standard in the literature but—besides the visibility restriction—differs from the pushdown automata in Chapter 3 in several technical and formal aspects:

- In VPA, call (push) and internal[17] transitions can neither read nor exchange the topmost stack symbol.
- VPA always have a fixed initial state s_0.
- VPA have a dedicated bottom-of-stack symbol $\perp$, which always constitutes the initial stack content and which can neither be popped nor pushed.

17: Internal transitions correspond to *swap* transitions in Chapter 3; in the present chapter, we use the term *internal*.

Another important semantic discrepancy with respect to the framework from Chapters 2 and 3 is that we consider only *infinite* computation paths (runs) in this chapter—the bottom symbol $\perp$ is convenient in this context as it ensures that runs never get stuck due to an empty stack. General weighted automata with infinite behaviors have been studied in the literature (e. g., [42, Section 3]), but are beyond the scope of this thesis. ..... ◁

[42]: Ésik et al. (2009), "Finite Automata".

6.1.2 A Deterministic Automaton Model for ω-VPLs

Consider an arbitrary (infinite) run $\rho = s_0\gamma_0, s_1\gamma_1, \ldots$ of a VPA[18] on some input word. We define the *stack height* of the ith configuration as $\text{height}(\rho(i)) = |\gamma_i| - 1$ (i.e., the bottom symbol $\perp$ does not count to the stack height).

18: See Definition 6.1 on page 154.

The stair parity condition (to be introduced in Definition 6.8 on page 159) relies on the notion of *steps*:

Definition 6.5 ***(Step)*** Let ρ be a run of a VPA $\mathcal{A}$. An index (or position) $i \geq 0$ is a *step* of ρ if

$$\forall n \geq i\colon \quad \text{height}(\rho(n)) \;\geq\; \text{height}(\rho(i))\,. \qquad \triangle$$

A step is thus a position of a VPA run from which the stack never drops below its current height. Slightly abusing terminology, we also say that a *configuration* $s_i\gamma_i$ occurring in a given run $\rho =$

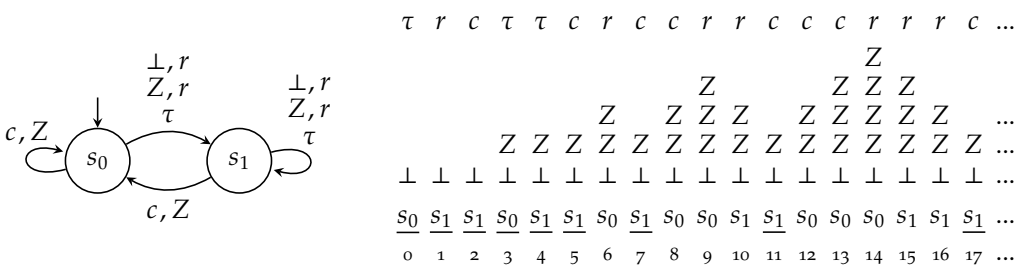


Figure 6.5: Left: An example VPA (in fact, a DVPA) with $\Gamma = \{Z, \bot\}$ over input alphabet $\Sigma = \{c\} \uplus \{\tau\} \uplus \{r\}$. Transitions labeled c, Z are *call transitions* that push Z on the stack. The transitions labeled τ are *internal transitions*; they ignore the stack completely. Transitions labeled Z, r and $\bot, r$ are *return transitions* that are only enabled if Z ($\bot$, respectively) is on top of the stack. When executing Z, r, the symbol Z is popped from the stack. However, the special bottom-of-stack symbol $\bot$ can never be popped (e.g., position 1 in the run). Right: The *unique* run of the DVPA on the infinite example input word $\tau\, r\, c\, \tau\, \tau\, c\, r\, c^2\, r^2\, c^3\, r^3\, c^4\, r^4\, c^5\, r^5 \ldots$. Steps are underlined.

$s_0\gamma_0, s_1\gamma_1, \ldots$ (which will often be clear from the context) is a step if position i is a step.[19] Steps are also called *minima* in the literature [44].

19: Notice that it would not make sense to say that a configuration is a step without reference to a particular run.

► **Example 6.6** (Steps) Figure 6.5 depicts a DVPA and the initial fragment of its unique run ρ on the input word $\tau\, r\, c\, \tau\, \tau\, c\, r\, c^2\, r^2\, c^3\, r^3 \ldots$. The step positions are underlined, that is, positions 0–5, 7, 11, and 17 are steps. Note that bottom configurations are always steps. .....◄

Steps play a central role in this chapter. Let us summarize some of their fundamental properties:

1. *Bounded increase of stack height between two steps:* The stack height from one step to the next either remains the same or increases by exactly one. Formally, if positions $i < j$ are adjacent steps in a run ρ (i.e., there is no step strictly between i and j in ρ), then $\text{height}(\rho(j)) - \text{height}(\rho(i)) \in \{0, 1\}$.
2. *Existence of a next step:* For each step $i \geq 0$ in a run on an infinite word w, there exists a *next step* $j > i$: If $w(i)$ is an internal or return symbol (e.g., $i = 0, 1, 3, 4$ in Figure 6.5), then $j = i + 1$. If $w(i)$ is a call, either it has no matching return (e.g., $i = 2$), in which case $j = i + 1$, or the call is eventually matched (e.g., $i = 5, 7, 11, 17$), and $j > i + 1$ occurs after reading the matching return.
3. *Infinitely many steps:* As a consequence of the previous observation, *every (infinite) run has infinitely many steps*.[20]
4. *Monotonicity of stack height at steps:* The stack height at the step positions forms a monotonic sequence and thus either grows unboundedly or stabilizes eventually; both cases are possible.

20: However, notice that the distance between two adjacent steps may grow unboundedly, as is the case in Figure 6.5.

▷ **Remark 6.7** (Steps of words vs steps of runs) We can also define the steps of a *word* $w \in \Sigma^\omega$, where Σ is a pushdown alphabet, as the positions where

a run of *any arbitrary* VPA on w has a step. Due to the visibility restriction, the actual behavior of the VPA does not influence the step positions [90]. In other words, the step positions are predetermined by the input word. Thus, we may also speak of the stack height of word w at position $i \geq 0$, which we denote with height(w, i). ◁

We need one more notion before defining stair parity: The *footprint* of an infinite run $\rho = s_0\gamma_0, s_1\gamma_1, \ldots$ of a VPA is the infinite sequence of states

$$\rho{\downarrow_{\text{Steps}}} \;=\; s_{n_0} s_{n_1} \ldots \,,$$

where n_i is the ith step of ρ. In other words, $\rho{\downarrow_{\text{Steps}}}$ is the projection of the run ρ onto the states occurring at its steps. For the example run in Figure 6.5, we have $\rho{\downarrow_{\text{Steps}}} = s_0 s_1 s_1 s_0 s_1^\omega$.

[90]: Löding et al. (2004), "Visibly Pushdown Games".

Definition 6.8 ***(Stair parity acceptance*** [90]***)*** Let $\mathcal{A}$ be a VPA[21] over a pushdown alphabet Σ. A stair parity acceptance condition for $\mathcal{A}$ is defined in terms of a *priority function* $\Omega\colon S \to \mathbb{N}_0$. A word $w \in \Sigma^\omega$ is accepted if $\mathcal{A}$ has a run ρ on w such that[22]

$$\min\left\{k \mid \overset{\infty}{\exists} i \geq 0\colon\ \Omega(\,\rho{\downarrow_{\text{Steps}}}(i)\,) \;=\; k\right\} \quad \text{is even}\,.$$

The language accepted by $\mathcal{A}$ is denoted $\mathcal{L}(\mathcal{A})$. △

21: See Definition 6.1 on page 154.

22: The symbol $\overset{\infty}{\exists}$ is short for "there exist infinitely many different."

Intuitively, a stair parity VPA thus accepts exactly the words that have a run such that the minimum priority occurring infinitely often at the states in the run's footprint is even. Stair parity thus differs from standard parity in that the former ignores the priorities of states at non-step positions, whereas the latter considers the priorities of all states in a run.

► **Example 6.9** (Recognizing repeatedly bounded words with stair parity) Suppose that we equip the DVPA from Figure 6.5 (restated in the margin) with the priority function Ω defined by $\Omega(s_0) = 1$ and $\Omega(s_1) = 2$. Assuming stair parity acceptance, the DVPA recognizes the language

$$\mathcal{L}_{\text{repbdd}} \;=\; \{w \in \Sigma^\omega \mid \exists B \geq 0\,,\ \overset{\infty}{\exists} i \geq 0\colon\ \text{height}(w, i) \;=\; B\}$$

of *repeatedly bounded words* [90], that is, the words whose stack height[23] equals some constant $B \geq 0$ infinitely often. For instance, the word

$$\tau\, r\, c\, \tau\, \tau\, c\, r\, c^2\, r^2\, c^3\, r^3\, c^4\, r^4\, c^5\, r^5 \ldots$$

from Figure 6.5 belongs to $\mathcal{L}_{\text{repbdd}}$ with $B = 1$.

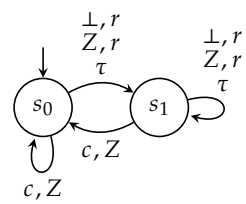


Figure 6.6: The DVPA from Figure 6.5 restated.

23: See Remark 6.7 on the preceding page.

To understand why the DVPA accepts precisely $\mathcal{L}_{\text{repbdd}}$, observe that a word is repeatedly bounded if and only if the stack height at the steps eventually stabilizes. The latter occurs if and only if at most finitely many

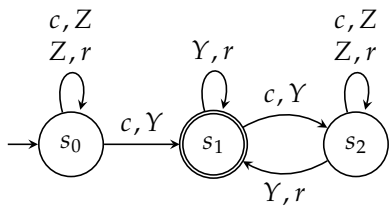


Figure 6.7: The nondeterministic Büchi VPA for repeatedly bounded words from Example 6.11.

steps are preceded by a call transition. The DVPA detects this behavior as follows: When reading a call symbol, it always moves to state s_0, which has odd priority; it thus accepts if and only if s_0 is visited finitely often at step positions. Notably, $\mathcal{L}_{\text{repbdd}}$ is not recognizable by any DVPA with a standard parity condition [3]. ◀

[3]: Alur et al. (2004), "Visibly pushdown languages".

Löding, Madhusudan, and Serre [90] have shown that stair parity DVPA can simulate—possibly nondeterministic—Büchi VPA:

[90]: Löding et al. (2004), "Visibly Pushdown Games".

Theorem 6.10 (Determinizing Büchi VPA via stair parity [90]) *For every Büchi VPA $\mathcal{A}$ with n states there exists a deterministic stair parity DVPA $\mathcal{D}$ with $2^{O(n^2)}$ states such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{D})$. Moreover, $\mathcal{D}$ can be constructed effectively in exponential time in the size of $\mathcal{A}$.*

In fact, stair parity DVPA are *exactly as expressive* as nondeterministic Büchi VPA: Both automaton models characterize the class of ω-VPLs [90].

► **Example 6.11** (A nondeterministic Büchi VPA for repeatedly bounded words) To provide a basic intuition for the equivalence of stair-parity DVPA and nondeterministic Büchi VPA, we revisit the language $\mathcal{L}_{\text{repbdd}}$ of repeatedly bounded words from Example 6.9.

A Büchi VPA can accept repeatedly bounded words as follows:[24] It nondeterministically guesses a position $i \geq 0$ in the input word w such that there are infinitely many positions $j > i$ with $\text{height}(w, i) = \text{height}(w, j)$, implying that w is repeatedly bounded by $B = \text{height}(w, i)$. The automaton may furthermore guess i such that B is minimal (this implies that i is a step). To verify that it guessed correctly, the automaton pushes a dedicated marker symbol every time the stack height exceeds B; this marker must always eventually be popped again, which can be signaled by means of an appropriate Büchi condition.

24: Our approach is inspired by [3, Proof of Theorem 7].

A concrete Büchi VPA for repeatedly bounded words over $\Sigma = \{c, r\}$ is depicted in Figure 6.7 (Y is the aforementioned marker symbol; for simplicity, we exclude words with *unmatched returns*, i. e., return symbols occurring on the empty stack). It is possible to further adapt the VPA to account for internal symbols and unmatched returns as well, but we omit the details for brevity. ◀

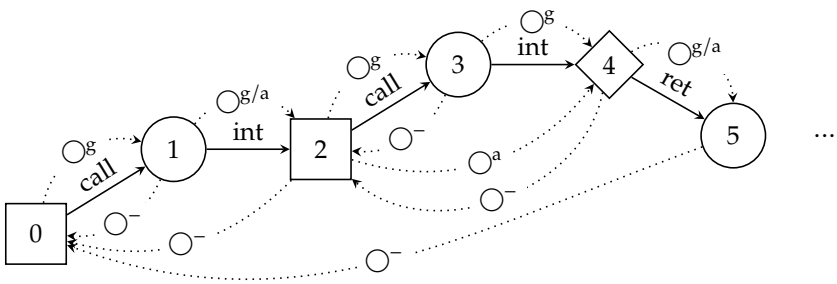


Figure 6.8: CaRet's three successor modalities $\bigcirc^{\mathrm{g}}$, $\bigcirc^{\mathrm{a}}$, and $\bigcirc^{-}$ applied to a prefix of an example word of the form

$$(X, \text{call})\,,\ (X, \text{int})\,,\ (X, \text{call})\,,\ (X, \text{int})\,,\ (X, \text{ret})\,,\ (X, \text{int})^{\omega}\,,$$

where X is a set of atomic propositions (its exact content is irrelevant for this illustration). Boxes, circles, and diamonds indicate call, internal, and return positions, respectively. The abstract successor $\bigcirc^{\mathrm{a}}$ is not always defined: at position 0, it is undefined because the call has no matching return; at position 3, it is undefined because the immediate successor (position 4) is a return.

6.1.3 CaRet, a Temporal Logic of Calls and Returns

The *Temporal Logic of Nested Calls and Returns* (CaRet) [2], an extension of LTL, allows specifying a class of ω-VPLs in a convenient, syntax-based manner:

[2]: Alur et al. (2004), "A Temporal Logic of Nested Calls and Returns".

Definition 6.12 ***(Syntax of CaRet* [2]*)*** Let AP be a finite set of *atomic propositions.* CaRet formulas adhere to the grammar

$$\begin{aligned}\varphi \quad ::= \quad & p \mid \varphi \vee \varphi \mid \neg\varphi && \text{(Boolean fragment)}\\ & \mid \bigcirc^{\mathrm{g}}\varphi \mid \bigcirc^{\mathrm{a}}\varphi \mid \bigcirc^{-}\varphi && \text{(successor modalities)}\\ & \mid \varphi\,\mathcal{U}^{\mathrm{g}}\varphi \mid \varphi\,\mathcal{U}^{\mathrm{a}}\varphi \mid \varphi\,\mathcal{U}^{-}\varphi\,, && \text{(until modalities)}\end{aligned}$$

where $p \in AP \cup \{\text{call}, \text{int}, \text{ret}\}$. △

Other common modalities such as $\Diamond^b$ and $\Box^b$ for $b \in \{\mathrm{g}, \mathrm{a}, -\}$ are defined as usual via $\Diamond^b\varphi = \mathit{true}\,\mathcal{U}^b\,\varphi$, and $\Box^b\varphi = \neg\Diamond^b\neg\varphi$.

We now explain the intuitive meaning of CaRet formulas.[25] We assume familiarity with LTL (see, e.g., [8, Chapter 5] for an introduction).

25: See Definition 6.14 on page 163 for the formal semantics.

[8]: Baier et al. (2008), *Principles of Model Checking*.

CaRet formulas are interpreted over infinite words from the pushdown alphabet[26] $\Sigma = 2^{AP} \times \{\text{call}, \text{int}, \text{ret}\}$, where $\Sigma_{\text{call}} = 2^{AP} \times \{\text{call}\}$, and Σ_{int} and Σ_{ret} are defined analogously. The modalities $\bigcirc^{\mathrm{g}}$ and $\mathcal{U}^{\mathrm{g}}$ are the standard *successor* and *until* modalities from LTL; in the context of CaRet, they are often referred to as *global* modalities.

26: See page 153.

CaRet extends LTL with two additional successor modalities: the *caller* $\bigcirc^{-}$ and the *abstract successor* $\bigcirc^{a}$. Figure 6.8 on the preceding page illustrates the semantics of these modalities. The caller $\bigcirc^{-}$ is a *past* modality that points to the position of the last (innermost) pending call symbol, if such a call exists. The abstract successor $\bigcirc^{a}$ of a call symbol is its *matching return*, if one exists, and is undefined otherwise. For internal or return symbols, the abstract successor typically coincides with the global successor $\bigcirc^{g}$, except when the latter is a return symbol, in which case $\bigcirc^{a}$ is undefined (e. g., position 3 in Figure 6.8).

The caller and abstract successor modalities induce sequences of positions, which we call the *caller path* and *abstract path*, respectively. The caller path is always finite, whereas the abstract path may be either finite or infinite. The until modalities $\mathcal{U}^{-}$ and $\mathcal{U}^{a}$ are defined analogously to LTL's standard until $\mathcal{U}^{g}$, with the difference that they are interpreted over the caller and abstract paths, respectively.

► **Example 6.13** (CaRet specifications) A prime application of CaRet is expressing *total correctness* of a procedure F [2]:

$$\varphi_{\text{total}} \;=\; \Box^{g}(\text{call} \wedge p \wedge p_F \rightarrow \bigcirc^{a} q)\,,$$

where p and q are atomic propositions that hold at the states where the pre- and postcondition of F are satisfied, respectively, and p_F is an atomic proposition marking calls to F. Intuitively, the formula φ_{total} asserts that whenever F is called in a state satisfying the precondition p, the corresponding return (if it exists) satisfies the postcondition q. Another example is the language of repeatedly bounded words from Example 6.9, which is captured by the formula

$$\varphi_{\text{repbdd}} \;=\; \Diamond^{g}\Box^{g}(\text{call} \rightarrow \bigcirc^{a}\text{ret})\,.$$

This formula states that, from some point onward, every call has a matching return, which implies that the stack height is eventually bounded. See [2] for further examples of CaRet specifications. ◄

[2]: Alur et al. (2004), "A Temporal Logic of Nested Calls and Returns".

We now present the formal semantics of CaRet. Let $\Sigma = 2^{AP} \times \{\text{call}, \text{int}, \text{ret}\}$ be the pushdown alphabet and $w \in \Sigma^{\omega}$. For $w(i) \in \Sigma_{\text{call}}$, let $\text{mr}_w(i)$ be the position of the matching return of $w(i)$, or undefined if no matching return exists. The *abstract successor* $\text{succ}^{a}_w(i)$ of position i in word w is defined as:

$$\text{succ}^{a}_w(i) \;=\; \begin{cases} \text{mr}_w(i) & \text{if } w(i) \in \Sigma_{\text{call}} \\ i+1 & \text{if } w(i) \notin \Sigma_{\text{call}} \;\wedge\; w(i{+}1) \notin \Sigma_{\text{ret}} \\ \text{undefined} & \text{if } w(i) \notin \Sigma_{\text{call}} \;\wedge\; w(i{+}1) \in \Sigma_{\text{ret}} \end{cases}$$

The *caller* $\text{succ}^{-}_w(i)$ of position i in word w is defined as the innermost

(i.e., nearest) unmatched call position $j < i$, or undefined if no such position exists.

Definition 6.14 ***(Semantics of CaRet)*** Let $\Sigma = 2^{AP} \times \{\text{call}, \text{int}, \text{ret}\}$ and $w \in \Sigma^\omega$. For all CaRet formulas φ and $i \in \mathbb{N}_0$, we define the relation $w(i) \models \varphi$ by induction over the structure of φ:

- $w(i) \models p$ iff $w(i) = (X, \mathit{type})$ such that either $p \in X$ or $p = \mathit{type}$
- $w(i) \models \varphi_1 \vee \varphi_2$ iff $w(i) \models \varphi_1$ or $w(i) \models \varphi_2$
- $w(i) \models \neg\varphi$ iff $w(i) \not\models \varphi$
- $w(i) \models \bigcirc^{\mathrm{g}} \varphi$ iff $w(i+1) \models \varphi$
- $w(i) \models \bigcirc^{\mathrm{a}} \varphi$ iff $\mathrm{succ}^{\mathrm{a}}_w(i) = j \in \mathbb{N}_0$ is defined and $w(j) \models \varphi$
- $w(i) \models \bigcirc^{-} \varphi$ iff $\mathrm{succ}^{-}_w(i) = j \in \mathbb{N}_0$ is defined and $w(j) \models \varphi$
- $w(i) \models \varphi_1 \, \mathcal{U}^b \varphi_2$ for $b \in \{a, g, -\}$ if there exist positions $i = i_0, i_1, \ldots, i_k$, $k \in \mathbb{N}_0$, such that
  1. $w(i_k) \models \varphi_2$, and
  2. for all $0 \leq j < k$, $w(i_j) \models \varphi_1$ and $i_{j+1} = \mathrm{succ}^{b}_w(i_j)$.

We write $w \models \varphi$ if $w(0) \models \varphi$. The language of all words satisfying a CaRet formula φ is denoted $\mathcal{L}(\varphi) = \{w \in \Sigma^\omega \mid w \models \varphi\}$. △

As mentioned, it is known that every CaRet specification encodes an ω-VPL; CaRet formulas can thus be translated to VPA [1, 4]:

[1]: Alur et al. (2018), "Model Checking Procedural Programs".

[4]: Alur et al. (2007), "First-Order and Temporal Logics for Nested Words".

Theorem 6.15 (From CaRet to VPA; e.g., [4, Theorem 5.1]) *CaRet-definable languages are ω-VPLs.*[27] *More specifically, for each CaRet formula φ there exists a (generally nondeterministic) Büchi VPA $\mathcal{A}$ such that $\mathcal{L}(\varphi) = \mathcal{L}(\mathcal{A})$, and $\mathcal{A}$ can be constructed in time $2^{O(|\varphi|)}$.*

27: See Definition 6.2 on page 155.

Gehnen provides a detailed description of the construction underlying Theorem 6.15 [55]. It generalizes the standard translation from LTL to Büchi automata (see, e.g., [8, Section 5.2]).

[55]: Gehnen (2021), "Automata-based model checking of recursive systems".

[8]: Baier et al. (2008), *Principles of Model Checking*.

Theorems 6.10 and 6.15 together imply that each CaRet formula can be translated to a deterministic stair parity VPA with at most *doubly-exponential* size.

6.2 Probabilistic Visibly Pushdown Automata

As explained in the introduction of this chapter,[28] we consider a labeled variant of pPDA with an additional visibility restriction. This section formally introduces this model and the resulting problem statements.

28: See paragraph *Outline of our Approach* on page 151.

Given a finite set X, we let

$$\mathrm{Distr}(X) \;=\; \Big\{ f : X \to [0,1] \mid \sum_{a \in X} f(a) = 1 \Big\}$$

be the set of probability distributions over X.[29] Our modified pPDA model is as follows:[30]

Definition 6.16 (pVPA: *Probabilistic visibly pushdown automaton*) A pVPA is a tuple $\Delta = (Q, q_0, \Gamma, \bot, P, \Sigma, \lambda)$ where Q is a finite set of states partitioned into $Q = Q_{\mathrm{call}} \uplus Q_{\mathrm{int}} \uplus Q_{\mathrm{ret}}$, $q_0 \in Q$ is an initial state, Γ is a finite stack alphabet, $\bot \in \Gamma$ is a special bottom-of-stack symbol, $P = (P_{\mathrm{call}}, P_{\mathrm{int}}, P_{\mathrm{ret}})$ is a triple of transition functions[31]

$$\begin{aligned} P_{\mathrm{call}} &: Q_{\mathrm{call}} \to \mathrm{Distr}(Q \times \Gamma_{\text{-}\bot}) \,, \\ P_{\mathrm{int}} &: Q_{\mathrm{int}} \to \mathrm{Distr}(Q) \,, \\ P_{\mathrm{ret}} &: Q_{\mathrm{ret}} \times \Gamma \to \mathrm{Distr}(Q) \,, \end{aligned}$$

$\Sigma = \Sigma_{\mathrm{call}} \uplus \Sigma_{\mathrm{int}} \uplus \Sigma_{\mathrm{ret}}$ is a pushdown alphabet, and $\lambda : Q \to \Sigma$ is a state labeling function which is required to be consistent with the partition on Q in the following sense: For all $\mathit{type} \in \{\mathrm{call}, \mathrm{int}, \mathrm{ret}\}$ and all $q \in Q$, we have $q \in Q_{\mathit{type}}$ if and only if $\lambda(q) \in \Sigma_{\mathit{type}}$. △

Note that in pVPA, which are in instance of *generative* automata, the stack action is visible in the states' labels, whereas in VPA—an instance of *reactive* automata—the stack action is visible in the *input symbols*.[32]

Similar to VPA, we use the notation $q \xrightarrow{p} rZ$, $q \xrightarrow{p} r$, and $qZ \xrightarrow{p} r$ to indicate that $P_{\mathrm{call}}(q)(r, Z) = p$, $P_{\mathrm{int}}(q)(r) = p$, and $P_{\mathrm{ret}}(q, Z)(r) = p$, respectively.

The intuitive behavior of a pVPA is almost identical to that of a standard pPDA,[33] except that—as in the VPA from Definition 6.1—the special symbol $\bot$ is *not* popped from the stack if a pop transition occurs in a bottom configuration.

The formal semantics of a pVPA is defined in terms of a countably infinite DTMC (discrete-time Markov chain)[34] with a state labeling function and a fixed initial state. We specify such DTMCs as 5-tuples $(S, s_0, P, \Sigma, \lambda)$, where S is the state space, $s_0 \in S$ is the initial state, $P : S \times S \to [0,1]$ is the (row-stochastic) transition probability matrix, Σ is an alphabet of labels, and $\lambda : S \to \Sigma$ is a state labeling function. A *run* of a DTMC is an element from S^ω, the set of infinite sequences of states. We extend the labeling function λ from states to runs in the natural way.

29: In this chapter, we disallow subdistributions in probabilistic transition functions; i. e., we consider only *proper* automata in the sense of Definition 3.22.

30: For reasons similar to those in Remark 6.4 on page 157, we do not define pVPA as an instance of the the weighted automata from Chapter 3.

31: Recall from Definition 6.1 on page 154 that $\Gamma_{\text{-}\bot} = \Gamma \setminus \{\bot\}$.

32: The introduction of Chapter 2 discusses the difference between generative and reactive automata in more detail.

33: See Definition 3.22 on page 71.

34: See Definition 2.38 on page 35.

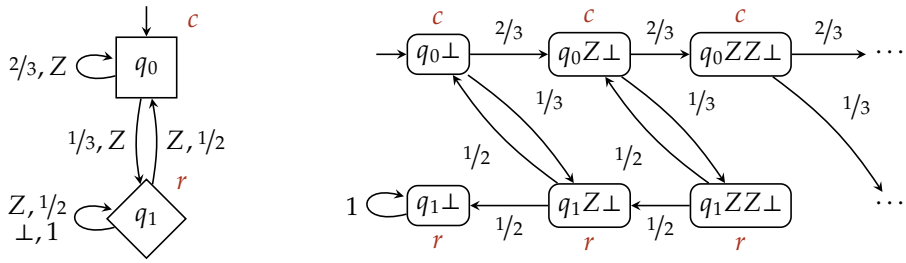


Figure 6.9: *Left:* An example pVPA Δ (Definition 6.16). Call and return states are rendered as squares and diamonds, respectively. State labels are colored. Transitions labeled p, Z (where $p \in [0,1]$) push Z with probability p. Transitions labeled X, p, for $X \in \Gamma$ and $p \in [0,1]$, mean that *if* X is on top of the stack, then the transition is taken with probability p and X is popped (if $X \neq \bot$). *Right:* The labeled DTMC $\mathfrak{M}_\Delta$ generated by Δ.

Consider a pVPA $\Delta = (Q, q_0, \Gamma, \bot, P, \Sigma, \lambda)$. We formally define *induced* labeled DTMC of Δ as

$$\mathfrak{M}_\Delta = \left(Q \times (\Gamma_{\text{-}\bot}^* \cdot \bot),\ q_0\bot,\ P_\Delta,\ \Sigma,\ \lambda_\Delta\right);$$

that is, the state space of $\mathfrak{M}_\Delta$ is the set of configurations of Δ, and the transition probability matrix P_Δ is defined as follows: We have $P_\Delta(q\gamma, r\gamma') = p > 0$ if and only if exactly one of the following cases applies:

1. *Push/call:* There exists $Z \in \Gamma_{\text{-}\bot}$ such that $q \xrightarrow{p} rZ$ and $\gamma' = Z\gamma$.
2. *Internal:* $q \xrightarrow{p} r$ and $\gamma' = \gamma$.
3. *Pop/return:* There exists $Z \in \Gamma_{\text{-}\bot}$ such that $qZ \xrightarrow{p} r$ and $Z\gamma' = \gamma$.
4. *Pop/return at* $\bot$: $q\bot \xrightarrow{p} r$ and $\gamma' = \gamma = \bot$.

Moreover, the labeling function of $\mathfrak{M}_\Delta$ is defined as $\lambda_\Delta(q\gamma) = \lambda(q)$ for all configurations $q\gamma$ of Δ.

► **Example 6.17** (The induced DTMC of a pVPA) Figure 6.9 depicts a pVPA Δ and a fragment of its induced labeled DTMC $\mathfrak{M}_\Delta$. ◄

We define the set Runs_Δ of a pVPA Δ as the runs of the Markov chain $\mathfrak{M}_\Delta$, that is, $\text{Runs}_\Delta = (Q \times (\Gamma_{\text{-}\bot}^* \cdot \bot))^\omega$. *Steps* of pVPA runs are defined as in Definition 6.5. A further example pVPA and its possible runs are depicted in Figure 6.10 on page 171.

We associate a probability space with Runs_Δ in the usual way (see, e.g., [8, Chapter 10]). To this end, we define the *σ-algebra* $\mathcal{F} \subseteq 2^{\text{Runs}_\Delta}$ as the smallest set that contains all *cylinder sets* $\rho \cdot \text{Runs}_\Delta$, where ρ is an arbitrary *finite* prefix of a run, and that is closed under complement and countable union. The sets in $\mathcal{F}$ are called *measurable*. We obtain a *unique*[35] probability measure $\mathbb{P}_\Delta : \mathcal{F} \to [0,1]$ by stipulating that,

[8]: Baier et al. (2008), *Principles of Model Checking*.

35: Uniqueness follows from standard results in measure theory.

for all cylinder sets $\rho \cdot \mathrm{Runs}_\Delta$ with $|\rho| > 0$,

$$\mathbb{P}_\Delta(\rho \cdot \mathrm{Runs}_\Delta) \;=\; [\rho(0) = q_0\bot] \cdot \prod_{i=0}^{|\rho|-2} P_\Delta\big(\rho(i), \rho(i{+}1)\big) \;,$$

where an empty product equals 1 and the *Iverson bracket* $[\ldots]$ evaluates to 1 if the condition inside is true and to 0 otherwise; this ensures that runs not starting in the initial configuration $q_0\bot$ have probability 0. We write $\mathbb{P}$ instead of $\mathbb{P}_\Delta$ whenever the pVPA is clear from the context.

We can now state the model checking problems studied in this chapter formally:

Definition 6.18 ***(Probabilistic VPA & DVPA model checking)*** Let $\mathcal{A}$ be a (possibly nondeterministic) Büchi VPA[36] over a pushdown alphabet Σ, Δ a pVPA with rational[37] transition probabilities and labels from Σ, and θ a rational probability threshold. The *quantitative* Büchi VPA model checking problem is to decide whether

$$\mathbb{P}\left(\{\rho \in \mathrm{Runs}_\Delta \mid \lambda(\rho) \in \mathcal{L}(\mathcal{A})\}\right) \overset{?}{\geq} \theta \;.$$

The corresponding *qualitative* model checking problem is the special case where $\theta = 1$.

The (quantitative and qualitative) probabilistic stair parity DVPA[38] model checking problems are defined similarly. △

Definition 6.19 ***(Probabilistic CaRet model checking)*** Let AP be a finite set of atomic propositions, φ a CaRet formula[39] over AP, Δ a pVPA with rational transition probabilities and labels from the pushdown alphabet $\Sigma = 2^{AP} \times \{\mathrm{call}, \mathrm{int}, \mathrm{ret}\}$, and θ a rational probability threshold. The *quantitative* CaRet model checking problem is to decide if

$$\mathbb{P}\left(\{\rho \in \mathrm{Runs}_\Delta \mid \lambda(\rho) \in \mathcal{L}(\varphi)\}\right) \overset{?}{\geq} \theta \;.$$

The *qualitative* CaRet model checking problem is the special case where $\theta = 1$. △

The probabilities occurring in Definitions 6.18 and 6.19 are well-defined because ω-VPLs are measurable sets [90].

▷ **Remark 6.20** (On alternative visibility restrictions) Our visibility restriction for pVPA differs slightly from an existing definition by Dubslaff, Baier, and Berg [39], who require all *incoming* transitions to a state to be of the same type (call, internal, or return). In contrast, our definition

36: See Definition 6.1 on page 154.

37: We will study the complexity of this problem assuming a *binary* encoding of the rational probabilities.

38: See Section 6.1.2 on page 157.

39: See Section 6.1.3 on page 161.

[39]: Dubslaff et al. (2012), "Model checking probabilistic systems against pushdown specifications".

imposes this requirement on the states' *outgoing* transitions. Our condition is natural for pVPA derived from procedural programs, such as the one in Figure 6.1. In particular, programs where randomness is restricted to *internal* statements like $x := \texttt{bernoulli}(0.5)$ or $x := \texttt{uniform}(0, 3)$ naturally satisfy our visibility condition because all call and return states in such programs are deterministic. ◁

6.3 Checking pVPA vs Stair Parity DVPA

In this section, we show that model checking pVPA[40] against ω-VPLs given in terms of a stair parity DVPA[41] is decidable. This is achieved by first computing an automata-theoretic product of the pVPA and the DVPA and then evaluating the acceptance condition in the product automaton.

40: See Definition 6.16 on page 164.

41: See Section 6.1.2 on page 157.

6.3.1 Products of Visibly Pushdown Automata

In general, pushdown automata are not closed under products as this would require *two* independent stacks. However, the visibility conditions on VPA and pVPA ensure that their product is again an automaton with just a single stack because the stack operations (push, internal, or pop) are forced to synchronize.

We now define the product formally. An *unlabeled* pVPA is a pVPA where the labeling function λ and alphabet Σ are omitted.

Definition 6.21 ***(Product of a pVPA and a DVPA)*** Let $\Delta = (Q, q_0, \Gamma, \bot, P, \Sigma, \lambda)$ be a pVPA, and $\mathcal{D} = (S, s_0, \Gamma', \bot, \delta, \Sigma)$ be a DVPA over the same pushdown alphabet Σ. The product of Δ and $\mathcal{D}$ is the unlabeled pVPA

$$\Delta \times \mathcal{D} \;=\; \big(Q \times S,\ (q_0, s_0),\ \Gamma \times \Gamma',\ \langle \bot, \bot \rangle,\ P_{\Delta\times\mathcal{D}}\big)\,,$$

where $P_{\Delta\times\mathcal{D}}$ is the smallest set of transitions satisfying the following rules:[42]

42: A rule $\frac{A \quad B}{C}$ means: If A and B hold, then C must also hold. (Any free symbol occurring in A, B, or C is universally quantified.)

$$\text{(Push)}\ \frac{q \xrightarrow{p}_{\Delta} rZ \qquad s \xrightarrow{\lambda(q)}_{\mathcal{D}} tY}{(q,s) \xrightarrow{p}_{\Delta\times\mathcal{D}} (r,t)\langle Z, Y\rangle} \qquad \text{(Pop)}\ \frac{qZ \xrightarrow{p}_{\Delta} r \qquad sY \xrightarrow{\lambda(q)}_{\mathcal{D}} t}{(q,s)\langle Z, Y\rangle \xrightarrow{p}_{\Delta\times\mathcal{D}} (r,t)}$$

$$\text{(Internal)}\ \frac{q \xrightarrow{p}_{\Delta} r \qquad s \xrightarrow{\lambda(q)}_{\mathcal{D}} t}{(q,s) \xrightarrow{p}_{\Delta\times\mathcal{D}} (r,t)}\,.$$

If the DVPA $\mathcal{D}$ is equipped with a priority function $\Omega\colon S \to \mathbb{N}_0$, then we extend Ω to $\Omega'\colon Q \times S \to \mathbb{N}_0$ via $\Omega'(q,s) = \Omega(s)$. △

It is not difficult to show that $\Delta \times \mathcal{D}$ is indeed a well-defined pVPA and moreover satisfies the following property:

Lemma 6.22 (Soundness of $\Delta \times \mathcal{D}$) *Let Δ be a pVPA and $\mathcal{D}$ be a stair parity DVPA with priority function Ω, both over the same pushdown alphabet Σ. Then the product pVPA $\Delta \times \mathcal{D}$ with priority function Ω' as in Definition 6.21 satisfies*

$$\begin{aligned} &\mathbb{P}\big(\{\rho \in \mathrm{Runs}_\Delta \mid \lambda(\rho) \in \mathcal{L}(\mathcal{D})\}\big) \\ = \ &\mathbb{P}\big(\{\rho \in \mathrm{Runs}_{\Delta\times\mathcal{D}} \mid \rho{\downarrow_{\mathrm{Steps}}} \in \mathrm{Parity}_{\Omega'}\}\big)\ , \end{aligned}$$

where $\mathrm{Parity}_{\Omega'}$ *denotes the set of words from* $(Q \times S)^\omega$ *that satisfy the* standard *parity condition induced by* Ω'*. Moreover, we can construct* $\Delta \times \mathcal{D}$ *in polynomial time.*

The proof of Lemma 6.22 is standard and given in Section B.2.

▷ **Remark 6.23** (On the visibility condition in the product) The product from Definition 6.21 satisfies the visibility condition of a pVPA (namely that the outgoing transitions of any given state perform the same stack action). However, this property is not essential for our approach—the techniques we apply to the product also work without this assumption. ◁

6.3.2 Stair Parity Acceptance Probabilities

Lemma 6.22 reduces model checking pVPA against stair parity DVPA to computing stair parity acceptance in the product, which is again an (unlabeled) pVPA. We therefore focus on pVPA in this section and do not consider DVPA.

Throughout this section, we fix an unlabeled pVPA $\Delta = (Q, q_0, \Gamma, \bot, P)$ with associated probability measure $\mathbb{P}_\Delta = \mathbb{P}$ on Runs_Δ. On the next pages we, describe the construction of a *finite* Markov chain $\mathcal{M}_\Delta$ that we call the *step chain* of Δ. Loosely speaking, $\mathcal{M}_\Delta$ simulates jumping from one *step*[43] of a run of Δ to the next.

43: See Definition 6.5 on page 157.

▷ **Remark 6.24** (On our variant of the step chain) To the best of my knowledge, the idea of the step Markov chain is due to Esparza, Kucera, and Mayr [44]. The variant presented here differs from the original definition [44] in at least two aspects. First, we must account for the semantics of the special bottom symbol $\bot$. This is why our chain uses a subset of $Q \cup Q\bot$ as states—it must distinguish whether a step occurs at a bottom configuration. The pPDA in [44], on the other hand, may have both finite and infinite runs, which requires a different treatment in the step chain. Second, we use step chains for a different purpose than [44], namely to show that general measurable properties defined on the sequence of steps of a run—such as stair parity—can be evaluated on pVPA.[44] ◁

[44]: Esparza et al. (2004), "Model Checking Probabilistic Pushdown Automata".

44: See Lemma 6.31 on page 175.

6.3.2.1 States at Steps: A Markov Process

For all $n \in \mathbb{N}_0$, we define a *random variable* $V^{(n)}$ on Runs_Δ whose value is either the state q of Δ at the nth step, or the *extended state* $q\bot$ in the special case where the nth step occurs at a bottom configuration $q\bot$, for some $q \in Q$. We denote the set of all such extended states with $Q\bot = \{q\bot \mid q \in Q\}$. Formally, $V^{(n)} \colon \mathrm{Runs}_\Delta \to Q \cup Q\bot$ is defined as

$$V^{(n)}(\rho) \;=\; \begin{cases} q & \text{if } \mathrm{step}_n(\rho) = q\gamma \text{ and } \gamma \neq \bot \\ q\bot & \text{if } \mathrm{step}_n(\rho) = q\bot\,, \end{cases}$$

where $\mathrm{step}_n(\rho)$ denotes the configuration at the nth step of ρ.

Lemma 6.25 (The state at the nth step is a random variable) *For all $n \in \mathbb{N}_0$ and $v \in Q \cup Q\bot$, the event $V^{(n)} = v$ is measurable. Thus, $V^{(n)}$ is a well-defined random variable.*

Proof. A more general version of this statement was proved in the literature [44]. Here, we give an alternative proof using that ω-VPLs are measurable sets [90]. We view $Q \cup Q\bot$ as a pushdown alphabet (the partition is induced by the partition on Q). We can construct a nondeterministic Büchi VPA that accepts a word $w \in (Q \cup Q\bot)^\omega$ if and only if the nth step of w is v (the size of this automaton depends on n). To this end, the VPA guesses the first n positions that are steps and goes to an accepting state s if the nth step was v. The automaton can verify that it guessed correctly as follows. If it believes a call symbol is a step, it pushes a special marker on the stack; if this marker is ever popped, then the call was no step and the guess was wrong. If it detects a wrong guess in this way, then it leaves the accepting state s, otherwise it loops there forever. The claim follows because the function $f \colon \mathrm{Runs}_\Delta \to (Q \cup Q\bot)^\omega$ that maps runs to the sequence of their (extended) states is measurable. Indeed, the preimage $f^{-1}(w)$ of every $w \in (Q \cup Q\bot)^\omega$ is

$$f^{-1}(w) \;=\; \bigcap_{i \geq 0} \bigcup_{\gamma_0 \ldots \gamma_i \in \Gamma_{\bot}^* \cdot \bot} w(0)\gamma_0 \ldots w(i)\gamma_i \cdot \mathrm{Runs}_\Delta\,,$$

which is a countable intersection of countable unions of basic cylinder sets. □

[44]: Esparza et al. (2004), "Model Checking Probabilistic Pushdown Automata".
[90]: Löding et al. (2004), "Visibly Pushdown Games".

In the parlance of probability theory, the random sequence

$$V^{(0)}\,,\; V^{(1)}\,,\; V^{(2)}\,,\; \ldots$$

of steps is a *stochastic process*. A crucial observation is that, for all $n \in \mathbb{N}_0$, $V^{(n+1)}$ depends only on $V^{(n)}$, but not on $V^{(i)}$ for $i < n$. This is because of the more general observation that the *state* q at any step configuration $q\gamma$ (with $\gamma \neq \bot$) *fully determines the future of the run* because being a step already implies that no symbol in γ can ever be read, as reading it implies popping it from the stack. In particular, q determines the probability distribution over possible next steps. A similar observation applies to bottom configurations of the form $q\bot$. The process $V^{(0)}, V^{(1)} \ldots$ thus has the *Markov property*, that is,

$$\begin{aligned} &\mathbb{P}(V^{(n)} = v_n \mid V^{(n-1)} = v_{n-1} \wedge \ldots \wedge V^{(0)} = v_0) \\ =\ &\mathbb{P}(V^{(n)} = v_n \mid V^{(n-1)} = v_{n-1}) \end{aligned} \tag{6.1}$$

holds for all $n > 0$ and $v_0, \ldots, v_n \in Q \cup Q\bot$ such that the conditional probabilities in Equation (6.1) are well-defined.[45] This was proved in detail in [44]. It is also clear that the Markov process is time-homogeneous in the sense that

$$\mathbb{P}(V^{(n+1)} = v' \mid V^{(n)} = v) \;=\; \mathbb{P}(V^{(m+1)} = v' \mid V^{(m)} = v) \tag{6.2}$$

for all $n, m \in \mathbb{N}_0$ for which the two conditional probabilities are well-defined. The following example provides some intuition for these facts.

45: A conditional probability is well-defined if the *condition*, i. e., the event on the right hand side of the vertical bar, has positive probability. The expressions in Equation (6.1) are not necessarily well-defined because the probability that $V^{(n-1)} = v_{n-1}$ might be zero for certain values of n and v_{n-1}.

► **Example 6.26** (The Markov process of states at steps) Consider the pVPA in Figure 6.10 (left). The initial fragments of its two equally likely runs are depicted in Figure 6.10 (middle). In this example, we can determine the next-step probabilities

$$\mathbb{P}(V^{(n)} \;=\; v_n \mid V^{(n-1)} = v_{n-1})$$

for all $n \in \mathbb{N}_0$ and $v_n, v_{n-1} \in Q \cup Q\bot$ by hand (these probabilities constitute the transition probabilities of the DTMC in Figure 6.10 (right), a simple instance of the general *step chain* construction that we will formally introduce in Definition 6.29 on page 175):

- For example, $V^{(0)} = q_0\bot$ holds with probability 1, and $V^{(1)} = q_1$ as well as $V^{(1)} = q_3\bot$ each hold with probability $1/2$ because the second step occurs either at position 1 with configuration $q_1 Z\bot$ or at position 3 with configuration $q_3\bot$ (and the two possibilities are equally likely). Thus, we have

$$\mathbb{P}(V^{(1)} = q_1 \mid V^{(0)} = q_0\bot) \;=\; \mathbb{P}(V^{(1)} = q_3\bot \mid V^{(0)} = q_0\bot) \;=\; \tfrac{1}{2}\,.$$

- Another simple consequence of the definitions is that $\mathbb{P}(V^{(n+1)} = q_2 \mid V^{(n)} = q_2) = 1$ for all $n \in \mathbb{N}_0$; this follows because q_2 only has a self-loop that pushes an additional symbol on the stack. Thus, every configuration at q_2 is a step whose successor configuration is the next step.

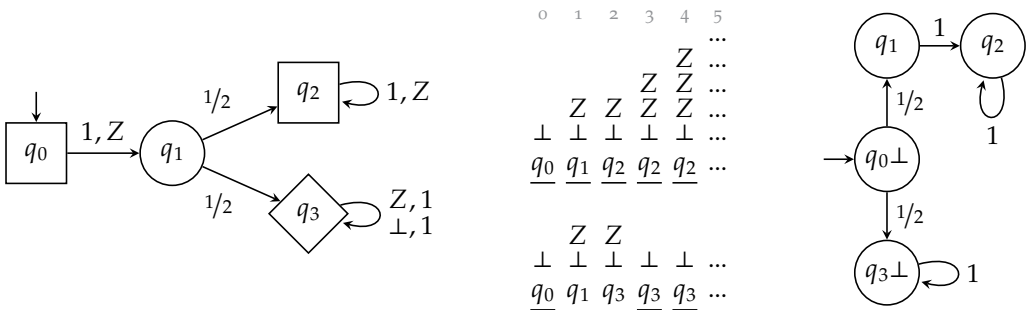


Figure 6.10: *Left:* An example (unlabeled) pVPA Δ. Recall that call, internal, and return states are drawn as squares, circles, and diamonds, respectively. *Middle:* Initial fragments of the two possible runs of Δ (steps underlined). *Right:* The step Markov chain $\mathcal{M}_\Delta$ (formally introduced in Definition 6.29 on page 175).

– Let us consider a more interesting case: the conditional probability $\mathbb{P}(V^{(2)} = q_2 \mid V^{(1)} = q_1)$. Here, we condition on a step at a configuration $q_1\gamma$ with $\gamma \neq \bot$, which implies that the topmost symbol of γ will never be popped. As a consequence, our assumption excludes $q_3\gamma$ as a successor configuration of $q_1\gamma$ because the automaton would pop the topmost symbol of γ after entering q_3. Thus, the only possible successor configuration of $q_1\gamma$ is $q_2Z\gamma$, and since every configuration at q_2 is a step,

$$\mathbb{P}(V^{(2)} = q_2 \mid V^{(1)} = q_1) \;=\; 1\,.$$

Crucially, the preceding example shows that, intuitively speaking, conditioning a pVPA on being in a certain state at a step position has a *global* effect—it influences the entire future of the run after that step; in particular, it determines which states may occur at the next step and with which probabilities. ◀

6.3.2.2 Characterizing the Next-Step Probabilities

In the following, we provide general expressions for the next-step probabilities $\mathbb{P}(V^{(n+1)} = v' \mid V^{(n)} = v)$.[46] We will see that these conditional probabilities can be expressed in terms of the *return* and *diverge* probabilities of Δ:

46: We determined several such probabilities in an ad hoc manner in Example 6.26.

Definition 6.27 ***(Return triples and diverge probabilities)*** Let $q, r \in Q$ be states and let $Z \in \Gamma_{\text{-}\bot}$ be a stack symbol of the pVPA Δ. We define the following probabilities:

A. The *return probability* $[qZ\,r]$ is the probability of reaching configuration $r\bot$ from $qZ\bot$ *without visiting any other bottom configuration*

Table 6.3: Transition probabilities of the step Markov chain. P_{type} for $\mathsf{type} \in \{\mathsf{call}, \mathsf{int}, \mathsf{ret}\}$ are the probabilities of the pVPA's call, internal, and return transitions, respectively. The values $[r'Z\,r]$ and $[q\uparrow]$ are the return and diverge probabilities from Definition 6.27.

	$\mathbf{q} \to \mathbf{r}$	$\mathbf{q}\bot \to \mathbf{r}\bot$	$\mathbf{q}\bot \to \mathbf{r}$	$\mathbf{q} \to \mathbf{r}\bot$
$q \in Q_{\text{call}}$	$\frac{[r\uparrow]}{[q\uparrow]}\left(\sum_{r',Z} P_{\text{call}}(q,r'Z)[r'Z\,r] + \sum_{Z} P_{\text{call}}(q,rZ)\right)$	$\sum_{r',Z} P_{\text{call}}(q,r'Z)[r'Z\,r]$	$\sum_{Z} P_{\text{call}}(q,rZ)[r\uparrow]$	0
$q \in Q_{\text{int}}$	$\frac{[r\uparrow]}{[q\uparrow]} P_{\text{int}}(q,r)$	$P_{\text{int}}(q,r)$	0	0
$q \in Q_{\text{ret}}$	undef.	$P_{\text{ret}}(q\bot, r)$	0	undef.

before $r\bot$. Formally,

$$[qZ\,r] \stackrel{\text{def}}{=} \mathbb{P}_{qZ\bot}\big(\{s\gamma\bot \mid s \in Q, \gamma \in \Gamma^{+}_{\text{-}\bot}\}\,\mathcal{U}\,\{r\bot\}\big)\,,$$

where $\mathbb{P}_{qZ\bot}$ is the probability measure associated with the infinite Markov chain $\mathfrak{M}_\Delta$ assuming initial state $qZ\bot$,[47] and $\mathcal{U}$ is the standard until operator from LTL.[48]

47: See page 165.

48: See Definition 6.14 on page 163.

B. The *diverge* probability

$$[q\uparrow] \stackrel{\text{def}}{=} 1 - \sum_{t \in Q} [qZ\,t]$$

is the probability to never reach a bottom configuration when starting in $qZ\bot$. △

A few remarks about the preceding definition are in order. The return probabilities as defined in Definition 6.27A essentially coincide with the triples considered in Chapters 3 and 4 for standard pPDA.[49] More precisely, if we treat a pVPA Δ as a standard pPDA Δ' by ignoring the semantics of the special bottom symbol $\bot$, then the two definitions coincide; the only exception is that triples of the form $[q\bot\,r]$ remain undefined for Δ. Thus, the return probabilities of Δ can be characterized via the fundamental equation system of Δ',[50] a fact we will exploit later in this chapter (Section 6.3.3).

49: These triples were first defined in Definition 3.8 on page 56.

50: See Theorem 3.11 on page 58.

In Definition 6.27B, the notation $[q\uparrow]$ is justified because the probability is actually independent of the current topmost symbol: In pVPA, the only way to access that symbol is by *popping* it from the stack.

Diverge probabilities are closely related to steps. In fact, the probability that any given non-bottom configuration with head qZ is a step is equal to $[q\uparrow]$. For example, in the pVPA in Figure 6.10, the configuration $q_1Z\bot$ is a step with probability $[q_1\uparrow] = 1/2$.

We can now state the technical key lemma of this section:

Lemma 6.28 *The conditional next-step probabilities in Table 6.3 are correct in the sense that whenever* $\mathbb{P}(V^{(n+1)} = v' \mid V^{(n)} = v)$ *is well-defined for some* $n \in \mathbb{N}_0$ *and* $v, v' \in Q \cup Q\bot$*, then it is equal to the probability in the respective column* $v \to v'$*.*

Proof sketch. Throughout this proof, we fix an (unlabeled) pVPA $\Delta = (Q, q_0, \Gamma, \bot, P)$, where $P = (P_{\text{call}}, P_{\text{int}}, P_{\text{ret}})$ denote the call, internal, and return transition functions, respectively.

We first justify the simpler entries of Table 6.3:

- *Column* $q \to r$ *with* $q \in Q_{\text{ret}}$*:* The conditional probability $\mathbb{P}(V^{(n+1)} = r \mid V^{(n)} = q)$ is never well-defined because steps cannot occur at non-bottom configurations with a return state (any such configuration would pop the topmost stack symbol, contradicting the definition of a step).
- *Column* $q\bot \to r\bot$ *with* $q \in Q_{\text{int}}$*:* Since q is internal, the successor of the bottom configuration $q\bot$ is the bottom configuration $r\bot$ with probability $P_{\text{int}}(q, r)$. Both $q\bot$ and $r\bot$ are necessarily steps (every bottom configuration is a step), so $\mathbb{P}(V^{(n+1)} = r\bot \mid V^{(n)} = q\bot) = P_{\text{int}}(q, r)$.
- *Column* $q\bot \to r\bot$ *with* $q \in Q_{\text{ret}}$*:* A return state at a bottom configuration behaves like an internal transition (since $\bot$ cannot be popped), so $\mathbb{P}(V^{(n+1)} = r\bot \mid V^{(n)} = q\bot) = P_{\text{ret}}(q\bot, r)$ by the same reasoning as in the previous case.
- *Column* $q\bot \to r$ *with* $q \in Q_{\text{int}}$*:* This probability is zero because the unique successor of the bottom configuration $q\bot$ (with q internal) is again a bottom configuration, so the next step must also be at a bottom configuration.
- *Column* $q\bot \to r$ *with* $q \in Q_{\text{ret}}$*:* This probability is zero for the same reason: the successor of $q\bot$ with $q \in Q_{\text{ret}}$ is necessarily a bottom configuration $r\bot$ for some $r \in Q$.
- *Column* $q \to r\bot$ *(all cases):* Every entry is either zero or ill-defined. Indeed, if the nth step occurs at a non-bottom configuration $q\gamma$ with $\gamma \neq \bot$, then γ is never popped (by definition of a step), so all future configurations have stack height at least $|\gamma|$ and hence can never be bottom configurations.

We now focus on the following more involved cases from Table 6.3 (let $r \in Q$ and $n \in \mathbb{N}_0$ be arbitrary):

- *Column* $q \to r$ *with* $q \in Q_{\text{int}}$*:*

$$\mathbb{P}(V^{(n+1)} = r \mid V^{(n)} = q) \quad = \quad \frac{[r\uparrow]}{[q\uparrow]} \cdot P_{\text{int}}(q, r)\,. \tag{6.3}$$

Suppose that the nth step occurs at position i of the run. Since the nth step occurs at an *internal* state $q \in Q_{\text{int}}$, the $(n+1)$th step must occur immediately at position $i+1$, and its state must be the immediate successor of q. The probability of transitioning from q to a successor r *and* having r be a step is $P_{\text{int}}(q, r) \cdot [r\uparrow]$, since the successor configuration must diverge (i.e., never pop the symbol below it) for it to qualify as a step. However, $\sum_{r \in Q} P_{\text{int}}(q, r) \cdot [r\uparrow]$ does not necessarily equal 1—it equals the probability that position $i+1$ is itself a step, which is precisely $[q\uparrow]$. Since we condition on $V^{(n)} = q$ (i.e., on position i being a step), we must normalize by $[q\uparrow]$, yielding $\frac{[r\uparrow]}{[q\uparrow]} \cdot P_{\text{int}}(q, r)$ as claimed.

– *Column $q \to r$ with $q \in Q_{\text{call}}$:*

$$\begin{aligned} &\mathbb{P}(V^{(n+1)} = r \mid V^{(n)} = q) \\ &\quad = \quad \frac{[r\uparrow]}{[q\uparrow]} \cdot \left(\sum_{r', Z} P_{\text{call}}(q, r'Z)[r'\, Z\, r] + \sum_{Z} P_{\text{call}}(q, rZ) \right). \end{aligned} \tag{6.4}$$

In Equation (6.4), the two summands correspond to the following case distinction: Since the nth step occurs at the *call* state $q \in Q_{\text{call}}$, the $(n+1)$th step either (i) occurs at the same stack height as the current step n (which means that the current call has a matching return), or (ii) the stack height at the next step $n+1$ is incremented by 1 compared to the stack height at step n. In case (ii), the next step occurs immediately at the next position, the second summand is thus the transition probability to go from q to r. In case (i), the symbols pushed by the outgoing transitions of q must be eventually popped. For instance, if we assume that q takes a transition to an immediate successor r' and pushes Z on the stack, then the probability that the next step occurs at r is precisely the *return probability* $[r'\, Z\, r]$ (see Definition 6.27). The factor $\frac{[r\uparrow]}{[q\uparrow]}$ is needed for similar reasons as in Equation (6.3).

See Appendix B on page 223 for a more rigorous formalization of the preceding arguments.

The remaining two cases (columns $q\bot \to r\bot$ and $q\bot \to r$ with $q \in Q_{\text{call}}$) are easier variants of case Equation (6.4) and are thus omitted.[51] □

51: Further details can be found in the full version of our FoSSaCS 2022 paper [130, page 30].

6.3.2.3 The Step Chain

Recall that the stochastic process $V^{(0)}, V^{(1)} \ldots$, where $V^{(i)} \in Q \cup Q\bot$ is the extended state at the ith step, has the Markov property.[52] Since

52: See Equation (6.1) on page 170.

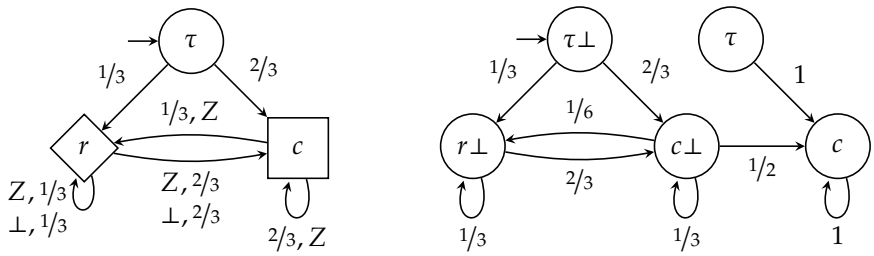


Figure 6.11: *Left:* Example pVPA Δ with the following return and diverge probabilities: $[cZ\,c] = 1/6$, $[cZ\,r] = 1/12$, $[rZ\,r] = 1/3$, $[rZ\,c] = 2/3$, $[c\uparrow] = 3/4$, $[\tau\uparrow] = 1/2$, and $[r\uparrow] = 0$. Recall that, in general, these probabilities may be irrational numbers (see, e.g., Example 3.27 on page 74). *Right:* The step chain $\mathcal{M}_\Delta$ according to Definition 6.29. The transition probabilities can be computed using the return and diverge probabilities and Table 6.3.

$Q \cup Q\bot$ is a finite set, we can now use Lemma 6.28 to construct the underlying finite Markov chain explicitly.

Definition 6.29 ***(Step chain)*** We define $\mathcal{M}_\Delta$ as the DTMC with states

$$M \;=\; \{q \in Q_{\text{call}} \cup Q_{\text{int}} \mid [q\uparrow] > 0\} \;\cup\; Q\bot\,,$$

initial state $q_0\bot$, and for all $v, v' \in M$, the probability of transition $v \to v'$ is defined according to Table 6.3. △

► **Example 6.30** (Step chain) Figure 6.11 depicts a pVPA and its step chain. In this example, transition probabilities are rational numbers. However, there also exist pVPA which give rise to irrational transition probabilities. ◄

The property of the step chain $\mathcal{M}_\Delta$ that is most relevant to us is given by the following Lemma 6.31. For $\rho \in \text{Runs}_\Delta$, we define

$$\rho\Downarrow_{\text{Steps}} \;=\; V^{(0)}(\rho)\,V^{(1)}(\rho)\,\ldots \;\in\; (Q \cup Q\bot)^\omega\,.$$

Notice that this is a slightly different *footprint* than the one introduced in Section 6.1.2: It projects runs on $Q \cup Q\bot$ instead of just Q; that is, it keeps track of whether the steps occur at bottom configurations or not.

Lemma 6.31 (Soundness of $\mathcal{M}_\Delta$) *Let Δ be a pVPA with step chain $\mathcal{M}_\Delta$. Let M be the states of the step chain and let $R \subseteq M^\omega$ be a measurable set. Then*

$$\mathbb{P}_\Delta\big(\{\rho \in \text{Runs}_\Delta \mid \rho\Downarrow_{\text{Steps}} \in R\}\big) \;=\; \mathbb{P}_{\mathcal{M}_\Delta}(R)\,,$$

where $\mathbb{P}_\Delta$ and $\mathbb{P}_{\mathcal{M}_\Delta}$ are the probability measures associated with Δ and $\mathcal{M}_\Delta$, respectively.

Proof. Our proof relies on some basic notions from measure theory; see, e.g., [5]. In fact, Lemma 6.31 is an instance of the following more general statement:

[5]: Ash et al. (2000), *Probability and Measure Theory*.

(♣) *Let* $(X, \mathcal{A}, \mu)$ *and* $(Y, \mathcal{B}, \nu)$ *be probability spaces such that, for some* $\mathcal{G} \subseteq 2^Y$*, we have* $\mathcal{B} = \sigma(\mathcal{G})$ *(i.e., $\mathcal{B}$ is the σ-algebra generated by the sets in $\mathcal{G}$). Assume furthermore that $\mathcal{G}$ is a π-system; that is, $\mathcal{G}$ is nonempty and closed under finite intersections. Let $f : X \to Y$ be such that for all $G \in \mathcal{G}$, $f^{-1}(G) \in \mathcal{A}$ and $\mu(f^{-1}(G)) = \nu(G)$. Then f is a measurable function and the pushforward measure $\nu' = \mu \circ f^{-1}$ coincides with ν.*

We now explain how to prove (♣) using fundamental measure theory. The fact that f is measurable follows because the inverse image f^{-1} preserves set operations.[53] For the claim that $\nu' = \nu$ it suffices to note that, by assumption, we have for all $G \in \mathcal{G}$ that $\nu'(G) = \nu(G)$, and since $\mathcal{G}$ is a π-system, an application of the *π-λ theorem*[54] implies that $\nu' = \nu$.

53: See, e.g., [5, below Definition 1.5.1].

54: See, e.g., [106, Proposition 2.10].

We instantiate (♣) as follows to prove Lemma 6.31: The probability spaces are the ones associated with the measures $\mathbb{P}_\Delta$ and $\mathbb{P}_{\mathcal{M}_\Delta}$. In particular, the σ-algebra relative to which $\mathbb{P}_{\mathcal{M}_\Delta}$ is defined is the one generated by the cylinder sets $\mathcal{C} = \{w \cdot M^\omega \mid w \in M^*\} \subseteq 2^{M^\omega}$. It is easy to see that $\mathcal{G} = \mathcal{C} \cup \{\emptyset\}$ is a π-system, and $\sigma(\mathcal{C}) = \sigma(\mathcal{G})$. We define

$$f : \big(Q \times (\Gamma_{\neg\bot}^* \cdot \bot)\big)^\omega \to M^\omega\,,\ f(\rho) \;=\; \rho{\Downarrow_{\text{Steps}}}\,;$$

that is, f projects a run from Δ to its footprint of steps (which is a run in $\mathcal{M}_\Delta$). To apply (♣) it remains to show that for all cylinder sets $w \cdot M^\omega$, where $w \in M^*$, we have that

(1) $f^{-1}(w \cdot M^\omega)$ is measurable, and
(2) $\mathbb{P}_\Delta(f^{-1}(w \cdot M^\omega)) = \mathbb{P}_{\mathcal{M}_\Delta}(w \cdot M^\omega)$.

For Item (1) notice that $f^{-1}(w \cdot M^\omega) = \bigwedge_{i=0}^{|w|-1}(V^{(i)} = w(i))$ is indeed measurable because it is a finite intersection of measurable events by Lemma 6.25; recall that $V^{(i)}$ denotes the (extended) state at the ith step. Item (2) is trivial in the case where $|w| = 0$, so we let $|w| = n+1$, $n \geq 0$, and exploit the properties of the step chain $\mathcal{M}_\Delta$. If $w(0) \neq q_0\bot$ (the initial state of $\mathcal{M}_\Delta$) then $\mathbb{P}_\Delta(V^{(0)} = w(0)) = \mathbb{P}_{\mathcal{M}_\Delta}(w \cdot M^\omega) = 0$. Otherwise $w(0) = q_0\bot$. In this case, if $n = 0$ (i.e., $|w| = 1$), then $\mathbb{P}_\Delta(V^{(0)} = w(0)) = \mathbb{P}_{\mathcal{M}_\Delta}(w \cdot M^\omega) = 1$. Else, if $n > 0$ (i.e., $|w| > 1$), by

the Markov property from Equation (6.1), we have

$$\begin{aligned} & \mathbb{P}_\Delta(V^{(n)} = w(n) \wedge \ldots \wedge V^{(0)} = w(0)) \\ = \quad & \mathbb{P}_\Delta(V^{(n)} = w(n) \mid V^{(n-1)} = w(n-1)) \cdot \ldots \\ & \qquad \ldots \cdot \mathbb{P}_\Delta(V^{(1)} = w(1) \mid V^{(0)} = w(0)) \\ = \quad & P(w(n-1), w(n)) \cdot \ldots \cdot P(w(0), w(1)) \\ & \qquad\qquad\qquad\qquad\qquad\qquad \text{(By Lemma 6.28)} \\ = \quad & \mathbb{P}_{\mathcal{M}_\Delta}(w \cdot M^\omega)\,, \end{aligned}$$

where P denotes the transition probability matrix of the Markov chain $\mathcal{M}_\Delta$ and the last equality holds by definition of the probability measure $\mathbb{P}_{\mathcal{M}_\Delta}$.[55] □

55: See page 165.

6.3.3 Reduction to the Existential Theory of the Reals

In this section, we prove the technical main result of this chapter, a solution to the probabilistic stair parity DVPA model checking problem:[56]

56: See Definition 6.18 on page 166.

Theorem 6.32 (Checking pVPA against stair parity DVPA) *Let Δ be a pVPA[57] with labeling function λ assigning states to symbols from a pushdown alphabet Σ, and let $\mathcal{D}$ be a stair parity DVPA over Σ. Furthermore, let θ be a rational probability threshold.*

The following problem is decidable in PSPACE*:*

$$\mathbb{P}_\Delta(\{\rho \in \mathrm{Runs}_\Delta \mid \lambda(\rho) \in \mathcal{L}(\mathcal{D})\}) \overset{?}{\geq} \theta\,.$$

57: See Definition 6.16 on page 164.

The idea of the proof is as follows:[58] First, we construct the product automaton $\Delta \times \mathcal{D}$.[59] By Lemma 6.22, it suffices to compute the stair parity acceptance probability of $\Delta \times \mathcal{D}$. Lemma 6.31 further reduces this to a *standard* parity acceptance probability in the step Markov chain $\mathcal{M}_{\Delta\times\mathcal{D}}$.[60]

58: See page 180 for the formal proof.

59: See Definition 6.21 on page 167.

60: See Definition 6.29 on page 175.

Parity acceptance probabilities in DTMCs can be computed using standard techniques: We first determine the *bottom strongly connected components* (BSCCs) of $\mathcal{M}_{\Delta\times\mathcal{D}}$, classifying them as either *good* (if the minimum priority among the BSCC's states is even) or *bad* (otherwise). Then, we conduct a standard reachability analysis (see, e. g., [8, Chapter 10]) toward the good BSCCs; the resulting reachability probability coincides with the parity acceptance probability.[61]

[8]: Baier et al. (2008), *Principles of Model Checking*.

61: See Figure 6.13 on page 180 for an example.

The remaining technical difficulty is that the transition probabilities of the step chain $\mathcal{M}_{\Delta\times\mathcal{D}}$ cannot be computed explicitly as they are

not rational in general. We address this using the fact that these probabilities are algebraic numbers effectively representable in the ETR (existential theory of the reals) [21]. Specifically, we rely on the following key result:

[21]: Brázdil et al. (2013), "Analyzing probabilistic pushdown automata".

Lemma 6.33 ([21, Theorem 2]) *The return probability triples of any given pPDA Δ with rational transition probabilities are expressible in the ETR. Formally, there exists an existentially quantified*[62] *first-order logic formula Φ built from rational constants, addition, multiplication, $\leq$, and free variables $\langle pZq \rangle$, where p and q range over the states and Z ranges over the stack symbols of Δ, such that the only solution (model) of Φ over $\mathbb{R}$ is given by the true return probabilities $[pZq]$. Moreover, Φ can be effectively constructed in polynomial space.*

62: That is, the formula may have a prefix of $\exists$-quantifiers but contains no quantifiers elsewhere.

The key idea of Lemma 6.33 is to express the fundamental PPS (positive polynomial system)[63] of the given pPDA Δ as a first-order logic formula. However, enforcing a unique solution without introducing universal quantifiers as asserted by Lemma 6.33 is subtle and requires additional insights.[64] We do not go into the full details of this issue, but provide some intuition by means of an example.

63: See Theorem 3.11 on page 58.

64: See [44, Proof of Theorem 3.6] for an encoding in the general theory of the reals, i.e., without restrictions on the quantifiers.

▶ **Example 6.34** (Encoding return probabilities in the ETR) We illustrate the construction of the ETR formula Φ from Lemma 6.33 using the pVPA Δ from Figure 6.11 (restated in the margin).

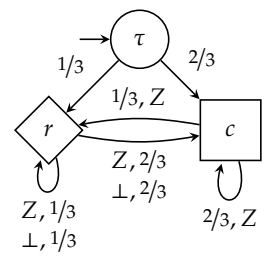


Figure 6.12: The pVPA from Figure 6.11 (left) restated.

For brevity, we consider only a fragment of the fundamental equation system of Δ. We can readily verify that $[rZr] = 1/3$ and $[rZc] = 2/3$. Hence, the equations for the probabilities $[cZc]$ and $[cZr]$ can be expressed via the following ETR formula Φ', where $\langle cZc \rangle$ and $\langle cZr \rangle$ are free variables:

$$\begin{aligned} & \langle cZc \rangle = \frac{2}{3} \cdot \langle cZc \rangle \cdot \langle cZc \rangle + \frac{2}{9} \cdot \langle cZc \rangle + \frac{4}{9} \cdot \langle cZr \rangle + \frac{2}{27} \\ \wedge \quad & \langle cZr \rangle = \frac{2}{3} \cdot \langle cZc \rangle \cdot \langle cZr \rangle + \frac{4}{9} \cdot \langle cZr \rangle + \frac{1}{27} \\ \wedge \quad & 0 \leq \langle cZc \rangle \leq 1 \\ \wedge \quad & 0 \leq \langle cZr \rangle \leq 1 \end{aligned}$$

Notice that Φ' includes constraints ensuring that the solutions are valid probabilities (i.e., belong to the interval $[0, 1]$).

The key challenge is that Φ' does not have a unique solution; indeed, it admits the following *two* solutions:

$$\langle cZc \rangle = \frac{1}{6}, \quad \langle cZr \rangle = \frac{1}{12}$$
$$\langle cZc \rangle = \frac{2}{3}, \quad \langle cZr \rangle = \frac{1}{3}$$

The true return probabilities are the *least* solution, which is $[cZ\,c] = 1/6$ and $[cZ\,r] = 1/12$.[65]

65: See Theorem 3.11 on page 58.

To exclude the spurious solution from Φ', we proceed as follows. First, we determine whether $[cZ\,c] + [cZ\,r] = 1$ holds by testing the satisfiability of the ETR formula Φ'' defined as

$$\Phi'' \quad \stackrel{\text{def}}{=} \quad \Phi' \;\wedge\; \langle cZ\,c\rangle + \langle cZ\,r\rangle \;<\; 1\,.$$

If Φ'' is satisfiable, then we have $[cZ\,c] + [cZ\,r] < 1$ since we are interested in the least solution; otherwise, $[cZ\,c] + [cZ\,r] = 1$. Based on this, we set

- $\Phi \stackrel{\text{def}}{=} \Phi' \wedge \langle cZ\,c\rangle + \langle cZ\,r\rangle < 1$ if $[cZ\,c] + [cZ\,r] < 1$,[66] or
- $\Phi \stackrel{\text{def}}{=} \Phi' \wedge \langle cZ\,c\rangle + \langle cZ\,r\rangle = 1$ otherwise.

66: In this case, Φ is identical to Φ''.

In our example, Φ'' is satisfiable, so $\Phi = \Phi''$, which has a unique solution as desired.

This construction generalizes to arbitrary pPDA [21, Theorem 2] and is guaranteed to yield a formula Φ with a *unique* solution [47, 143]. The latter result relies on a careful analysis of the structure of the set of fixed points of a pPDA's fundamental PPS. ◀

[21]: Brázdil et al. (2013), "Analyzing probabilistic pushdown automata".

[47]: Etessami et al. (2005), "Algorithmic Verification of Recursive Probabilistic State Machines".

[143]: Yannakakis et al. (2005), "Checking LTL Properties of Recursive Markov Chains".

Lemma 6.35 *The transition probabilities of the step Markov chain*[67] *of any given pVPA with rational probabilities are expressible in the ETR (in the sense of Lemma 6.33).*

67: See Definition 6.29 on page 175.

Proof. Table 6.3 shows that the transition probabilities of the step chain can be expressed in terms of the return probabilities of the given pPDA using addition, multiplication, division, and rational constants. By Lemma 6.33, it suffices to note that ETR-expressible numbers are closed under these operations, which is a standard fact. Nevertheless, we include a brief argument for illustration: Let $x, y \in \mathbb{R}$ be expressed by ETR formulas $\Phi(x)$ and $\Phi'(y)$, respectively. Then the formula

$$\Phi''(z) \quad \stackrel{\text{def}}{=} \quad \exists x\colon \exists y\colon \Phi(x) \;\wedge\; \Phi'(y) \;\wedge\; z = x + y\,,$$

where z is a fresh free variable, expresses the sum $x + y$. Multiplication is handled similarly. For division, we observe that

$$\Phi''(z) \quad \stackrel{\text{def}}{=} \quad \exists x\colon \exists y\colon \Phi(x) \;\wedge\; \Phi'(y) \;\wedge\; z \cdot y = x$$

encodes the quotient x/y provided that $y \neq 0$; otherwise, Φ'' does not properly encode a real number as it is either unsatisfiable (if $x \neq 0$) or trivially satisfied by all numbers (if $x = 0$). Note, however, that division by zero cannot occur in Table 6.3 because we only divide by $[q\uparrow]$ for an internal or call state q, and $[q\uparrow] > 0$ holds per definition of the step chain.[68] □

68: See Definition 6.29 on page 175.

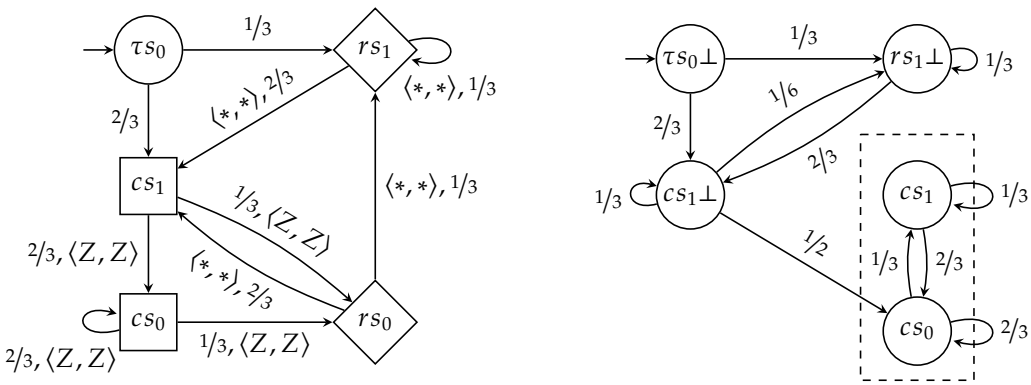


Figure 6.13: *Left:* The product of the pVPA from Figure 6.11 (left) and the DVPA from Figure 6.5 on page 158. The placeholder $*$ stands for an arbitrary stack symbol. *Right:* Step chain of the prodcut pVPA on the left (Definition 6.29). The dashed region is the only BSCC. It violates the parity condition $\Omega(s_0) = 1$ and $\Omega(s_1) = 2$ inherited from the DVPA (see Example 6.9 on page 159) since every run reaching the BSCC visits cs_0 infinitely often with probability 1. Only reachable states are depicted.

We now present the PSPACE procedure proving Theorem 6.32. The example in Figure 6.13 illustrates the main steps of the approach.

Proof of Theorem 6.32. The proof proceeds in five steps.

Step 1: Construct the product We begin by constructing the product pVPA $\tilde{\Delta} = \Delta \times \mathcal{D}$, which can be done in polynomial time.[69] Let Q denote the state set of the product $\tilde{\Delta}$. Recall that $\tilde{\Delta}$ inherits a priority function $\Omega' \colon Q \to \mathbb{N}_0$ from the stair parity DVPA $\mathcal{D}$. By Lemma 6.22, our goal is to determine the probability

$$\mathbb{P}_{\tilde{\Delta}}(\{\rho \in \mathrm{Runs}_{\tilde{\Delta}} \mid \rho{\downarrow_{\mathrm{Steps}}} \in \mathrm{Parity}_{\Omega'}\}) \,, \tag{6.5}$$

that is, the probability that the footprint of a run of the product $\tilde{\Delta}$ satisfies the (standard) parity condition induced by Ω'.

69: See Definition 6.21 on page 167.

Step 2: Exploit the step chain We express the probability in Equation (6.5) using the step chain $\mathcal{M}_{\tilde{\Delta}}$.[70] Let $M \subseteq Q \cup Q\bot$ denote the states of $\mathcal{M}_{\tilde{\Delta}}$. We extend the priority function Ω' to all states in M via $\Omega'' \colon M \to \mathbb{N}_0$, defined by $\Omega''(q) = \Omega''(q\bot) = \Omega'(q)$ for all $q \in Q$. This extended priority function Ω'' induces a parity acceptance set $\mathrm{Parity}_{\Omega''} \subseteq M^\omega$, which is ω-regular and thus measurable.

70: See Definition 6.29 on page 175.

Let $\rho \in \mathrm{Runs}_{\tilde{\Delta}}$. By definition, $\rho{\downarrow_{\mathrm{Steps}}} \in \mathrm{Parity}_{\Omega'}$ if and only if $\rho{\Downarrow_{\mathrm{Steps}}} \in \mathrm{Parity}_{\Omega''}$.[71] Thus, by Lemma 6.31, we obtain

$$\mathbb{P}_{\tilde{\Delta}}(\{\rho \in \mathrm{Runs}_{\tilde{\Delta}} \mid \rho{\downarrow_{\mathrm{Steps}}} \in \mathrm{Parity}_{\Omega'}\}) \;=\; \mathbb{P}_{\mathcal{M}_{\tilde{\Delta}}}(\mathrm{Parity}_{\Omega''}) \,,$$

71: The notation $\rho{\Downarrow_{\mathrm{Steps}}}$ was introduced on page 175.

where the right hand side is the probability that a run of the step chain $\mathcal{M}_{\tilde{\Delta}}$ satisfies the parity condition induced by Ω''. We have thus reduced the problem to computing a standard parity acceptance probability in the finite Markov chain $\mathcal{M}_{\tilde{\Delta}}$.

The remainder of the proof uses standard techniques similar to existing approaches in the literature.[72]

72: See, e. g., the proof of Theorem 5.15 in [44].

Step 3: Construct the step chain graph We construct the *underlying graph* $G_{\tilde{\Delta}}$ of the step chain $\mathcal{M}_{\tilde{\Delta}}$. More precisely, we determine the state set M of the step chain—this requires checking whether $[q\uparrow] > 0$ holds for certain states $q \in Q$,[73] which is decidable in PSPACE—and add a directed edge between states $v, v' \in M$ to the graph if and only if the probability of transition $v \to v'$ is positive. The latter can also be determined in PSPACE since the transition probabilities of $\mathcal{M}_{\tilde{\Delta}}$ are ETR-expressible.[74]

73: See Definition 6.29 on page 175.

74: See Lemma 6.35 on page 179. Also see Table 6.4 on page 183.

Step 4: Determine the good BSCCs We determine the BSCCs of the graph $G_{\tilde{\Delta}}$ from the previous step via standard graph analysis; this is possible in polynomial time. We label each BSCC $B \subseteq M$ as *good* if $\min_{v \in B} \Omega''(v)$ is even, and as *bad* otherwise. Let $\mathcal{G}$ denote the set of good BSCCs. It is well-known that, in the finite Markov chain $\mathcal{M}_{\tilde{\Delta}}$, the probability of satisfying the parity condition $\mathrm{Parity}_{\Omega''}$ coincides with the probability of eventually reaching a good BSCC:

$$\mathbb{P}(\mathrm{Parity}_{\Omega''}) \;=\; \mathbb{P}(\Diamond \bigcup_{B \in \mathcal{G}} B)\,. \tag{6.6}$$

Intuitively, Equation (6.6) holds because, with probability 1, a finite DTMC eventually reaches one of its BSCCs and visits all states in the BSCC infinitely often; therefore, the parity condition is satisfied if and only if the reached BSCC is good. It remains to compute the probability of reaching a good BSCC in $\mathcal{M}_{\tilde{\Delta}}$.

Step 5: Encode the reachability probability We use the previous step to partition the states M of the step chain $\mathcal{M}_{\tilde{\Delta}}$ into three sets: $M_{=0}$ contains all states from which no good BSCC is reachable in the graph $G_{\tilde{\Delta}}$, $M_{=1}$ contains all good BSCCs, and $M_?$ contains all remaining states. We recall that the probabilities of reaching $M_{=1}$ are the *unique* solution of the following linear equation system over variables x_v for $v \in M$:[75]

75: See, e. g., [8, Chapter 10] or Section 2.3.1.

$$x_v \;=\; \begin{cases} 0 & \text{if } v \in M_{=0}\,, \\ 1 & \text{if } v \in M_{=1}\,, \\ \sum_{v \xrightarrow{p} v'} p \cdot x_{v'} & \text{if } v \in M_?\,. \end{cases}$$

We can treat the probabilities $\mathbf{p}$ and the variables $\mathbf{x}$ in this equation system as free variables of an ETR formula $\Phi_{\mathcal{R}}(\mathbf{p}, \mathbf{x})$. By Lemma 6.35, there exists an ETR formula $\Phi(\mathbf{p})$ expressing $\mathbf{p}$. The ETR formula

$$\exists \mathbf{p}, \mathbf{x}\colon \quad \Phi(\mathbf{p}) \quad \wedge \quad \Phi_{\mathcal{R}}(\mathbf{p}, \mathbf{x}) \quad \wedge \quad x_{v_0 \bot} \geq \theta$$

is true if and only if the probability of reaching a good BSCC from the initial state $v_0\bot$ of $\mathcal{M}_{\tilde{\Delta}}$ is at least θ. The truth of this formula can be decided in PSPACE [23].

This concludes the proof that the probabilistic stair parity DVPA model checking problem is decidable in PSPACE. □

6.4 Checking pVPA vs Büchi VPA and CaRet

With Theorems 6.10 and 6.32 on page 160 and on page 177 it follows immediately that quantitative model checking of pVPA against nondeterministic Büchi VPA[76] is decidable in EXPSPACE.

76: See Definition 6.18 on page 166.

In the *qualitative* case, however, we can improve the complexity to EXPTIME. The key observation is that we can construct the underlying graph of a step chain $\mathcal{M}_{\Delta \times \mathcal{D}}$ of a product automaton $\Delta \times \mathcal{D}$ in exponential time in the size of Δ but in *polynomial* time in the size of $\mathcal{D}$.

Theorem 6.36 (Qualitative model checking of pVPA vs Büchi VPA) *Let Δ be a pVPA and $\mathcal{A}$ be a Büchi VPA over the same pushdown alphabet. The problem*

$$\mathbb{P}(\{\rho \in \mathrm{Runs}_\Delta \mid \lambda(\rho) \in \mathcal{L}(\mathcal{A})\}) \stackrel{?}{=} 1$$

is EXPTIME*-complete.*[77]

77: This is the qualitative version of the model checking problem from Definition 6.18.

[47]: Etessami et al. (2005), "Algorithmic Verification of Recursive Probabilistic State Machines".

Proof. The lower bound is due to [47, Theorem 8] and already holds for non-pushdown Büchi automata. We now describe an EXPTIME decision procedure:

– We first determinize $\mathcal{A}$ using Theorem 6.10, which is possible in time exponential in $|\mathcal{A}|$. Let $\mathcal{D}$ be the resulting stair parity DVPA and consider the product $\Delta \times \mathcal{D}$.[78] Note that the product can be constructed in polynomial time in $|\mathcal{D}|$ and $|\Delta|$, and thus in exponential time in the overall input size.

78: See Definition 6.21 on page 167.

79: See Definition 6.27 on page 171.

Table 6.4: The underlying graph of the step chain. The condition in each cell is true if and only if the corresponding transition probability in Table 6.3 is nonzero.

	$\mathbf{q} \to \mathbf{r}$	$\mathbf{q}\bot \to \mathbf{r}$
$q \in Q_{\text{call}}$	$[r\uparrow]{>}0 \wedge \big((\exists r', Z\colon P_{\text{call}}(q, r'Z){>}0 \wedge [r'Z\, r]{>}0) \vee \exists Z\colon P_{\text{call}}(q, rZ){>}0\big)$	$[r\uparrow]{>}0 \wedge \exists Z\colon P_{\text{call}}(q, rZ){>}0$
$q \in Q_{\text{int}}$	$[r\uparrow] > 0 \ \wedge\ P_{\text{int}}(q, r) > 0$	false
$q \in Q_{\text{ret}}$	n/a	false
	$\mathbf{q}\bot \to \mathbf{r}\bot$	$\mathbf{q} \to \mathbf{r}\bot$
$q \in Q_{\text{call}}$	$\exists r', Z\colon P_{\text{call}}(q, r'Z){>}0 \wedge [r'Z\, r]{>}0$	false
$q \in Q_{\text{int}}$	$P_{\text{int}}(q, r) > 0$	false
$q \in Q_{\text{ret}}$	$P_{\text{ret}}(q\bot, r) > 0$	n/a

– The crucial observation for this proof concerns the diverge probabilities[79] in the product: We have

$$[q\uparrow] \;=\; [(q,s)\uparrow]$$

for all states (q, s) of $\Delta\times\mathcal{D}$. This equality holds because, by definition of the product, $\mathcal{D}$ merely *observes* the runs of Δ; therefore, the runs of $\Delta\times\mathcal{D}$ and Δ—and their probabilities—are in a one-to-one correspondence. We compute the set $\text{Div}_\Delta = \{q \mid [q\uparrow] > 0\} \subseteq Q$, where Q denotes the states of Δ, in *exponential time in* $|\Delta|$ using a PSPACE decision procedure for the ETR. Note that computing $\text{Div}_{\Delta\times\mathcal{D}} = \{(q,s) \mid [(q,s)\uparrow] > 0\}$ directly would take *doubly-exponential* time in $|\mathcal{A}|$; this optimization is therefore essential for obtaining the EXPTIME upper bound.
– We determine the set of triples

$$\text{Ret}_{\Delta\times\mathcal{D}} \;=\; \big\{(q, Z, p) \mid [qZ\, p] > 0\big\}$$

in $\Delta \times \mathcal{D}$. Unlike the divergence probabilities, this set can be computed in *polynomial* time in the size of $\Delta \times \mathcal{D}$ (hence exponential in the input size) because we may disregard the exact transition probabilities and conduct a standard reachability analysis in a nondeterministic pushdown automaton [1]; also see [21, page 136].
– The next step is to construct the underlying *graph* $G_{\Delta\times\mathcal{D}}$ of the step chain $\mathcal{M}_{\Delta\times\mathcal{D}}$, i.e., the directed graph that has the same vertices as $\mathcal{M}_{\Delta\times\mathcal{D}}$ and includes an edge (u, v) if and only if the one-step transition probability from u to v is positive in the Markov chain. This can be done in polynomial time in $|\Delta \times \mathcal{D}|$ using the sets Div_Δ and $\text{Ret}_{\Delta\times\mathcal{D}}$ defined above and Table 6.4.
– The final step is, as in Theorem 6.32, to determine the BSCCs of $G_{\Delta\times\mathcal{D}}$, classify them as good or bad according to whether they

[1]: Alur et al. (2018), "Model Checking Procedural Programs".

[21]: Brázdil et al. (2013), "Analyzing probabilistic pushdown automata".

satisfy the (standard) parity condition inherited from $\mathcal{D}$, and then check whether a bad BSCC is reachable from the initial state via a standard graph reachability analysis. All these steps can be performed in polynomial time in $|\Delta \times \mathcal{D}|$. □

Theorems 6.10, 6.15, 6.32 and 6.36 imply the following complexity results for CaRet model checking:

Theorem 6.37 (Complexity of checking pVPA against CaRet) *The quantitative and qualitative probabilistic CaRet model checking problems*[80] *are decidable in* 2EXPSPACE *and* 2EXPTIME*, respectively.*

80: See Definition 6.19 on page 166.

Both problems are known to be EXPTIME-hard [143].

[143]: Yannakakis et al. (2005), "Checking LTL Properties of Recursive Markov Chains".

6.5 Open Questions

1. What is the exact complexity of probabilistic CaRet model checking?[81]
2. How do *certificates*[82] for the model checking problems studied in this chapter look like?

81: See Definition 6.18 on page 166.

82: See Chapter 4.

6.6 Additional Bibliographic Remarks

Mehlhorn was the first to define (deterministic) VPA [99], which he called *input-driven* pushdown automata. The term *visibly pushdown* was coined later by Alur and Madhusudan in their seminal paper on VPLs [3].

Besides CaRet, other logics for describing ω-VPLs have been studied. An example is NWTL$^+$, an FO-complete extension of CaRet [4].

Model checking pPDA (and related models such as recursive Markov chains [49]) against various types of logical and automata-based specifications has been extensively studied in the literature since the mid-2000s. Esparza, Kucera, and Mayr studied *probabilistic computation tree logic* and deterministic Büchi specifications [44]. Their approach was later extended to deterministic Muller automata, which cover all ω-regular properties [20, 86]. Yannakakis and Etessami were the first to study LTL model checking [143], which they proved to be EXPTIME-complete, even in the quantitative case. Brázdil et al.'s survey provides an overview of these results and several others [21].

[99]: Mehlhorn (1980), "Pebbling Moutain Ranges and its Application of DCFL-Recognition".

[3]: Alur et al. (2004), "Visibly pushdown languages".

[4]: Alur et al. (2007), "First-Order and Temporal Logics for Nested Words".

[49]: Etessami et al. (2005), "Recursive Markov Chains, Stochastic Grammars, and Monotone Systems of Nonlinear Equations".

[44]: Esparza et al. (2004), "Model Checking Probabilistic Pushdown Automata".

[20]: Brázdil et al. (2005), "On the Decidability of Temporal Properties of Probabilistic Pushdown Automata".

[86]: Kucera et al. (2006), "Model Checking Probabilistic Pushdown Automata".

[143]: Yannakakis et al. (2005), "Checking LTL Properties of Recursive Markov Chains".

[21]: Brázdil et al. (2013), "Analyzing probabilistic pushdown automata".

More closely related to our work is a paper by Dubslaff, Baier, and Berg, which studies the model checking of pVPA against DVPA with Muller acceptance conditions [39]. However, Muller DVPA do not characterize the full class of ω-VPLs; indeed, the language $\mathcal{L}_{\text{repbdd}}$ from Example 6.9 on page 159 is not recognizable by a Muller DVPA [3, Theorem 7]. Our work solves the model checking problem for general ω-VPLs, represented either as Büchi VPA or stair parity DVPA.

[39]: Dubslaff et al. (2012), "Model checking probabilistic systems against pushdown specifications".

Recently, Pontiggia, Bartocci, and Chiari have studied probabilistic model checking problems for *operator precedence languages* (OPLs), a class of context-free languages more general than VPLs [109]. OPLs can express, for example, characteristic properties of procedural programs with *exceptions* and their corresponding handlers. More precisely, Pontiggia, Bartocci, and Chiari propose a temporal logic called POTLf$_\chi$, whose qualitative model checking problem is EXPTIME-complete and whose quantitative model checking problem is decidable in EXPSPACE (this is in contrast to our 2EXPTIME and 2EXPSPACE upper bounds for CaRet). However, the precise relationship between POTLf$_\chi$ and CaRet remains open [109]; in particular, it is unclear whether the results for POTLf$_\chi$ imply tighter complexity bounds for CaRet than those presented in this chapter. A model checking tool for POTLf$_\chi$ is available [110].

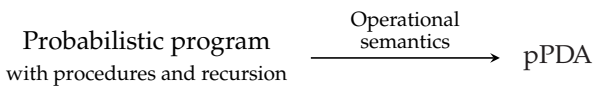
[109]: Pontiggia et al. (2026), "Model Checking Probabilistic Operator Precedence Automata".

[110]: Pontiggia et al. (2025), "POPACheck: A Model Checker for Probabilistic Pushdown Automata".

7 From Programs to Automata

7.1 Basic Probabilistic Programs . . 188
7.1.1 Syntax 189
7.1.2 Markov Chains as an Operational Model . . 190
7.1.3 Semantics . . . 192
7.1.4 Verification via DTMCs 195
7.1.5 Examples 197
7.2 Procedural Probabilistic Programs 199
7.2.1 Syntax 200
7.2.2 pPDA as an Operational Model 202
7.2.3 Semantics . . . 205
7.2.4 Examples 210
7.3 Open Questions 212
7.4 Bibliographic Remarks 213

In this chapter, we formally describe how *probabilistic programs* with recursive procedures can be faithfully modeled as pPDA (probabilistic pushdown automata). The translation is realized via a Plotkin-style *small-step operational semantics* [107] of a simple—yet expressive—procedural probabilistic programming language:



Effective Program Analysis via Finite Pushdown Automata The general idea of modeling recursive programs as pushdown automata and similar systems is standard in the literature [1]. Indeed, program modeling was among the primary motivations when pPDA and their equivalent cousin—*recursive Markov chains*—were first introduced [44, 49]. A program analysis based on such formalisms is appealing because they adequately abstract the unbounded call stack of a recursive program while remaining amenable to algorithmic techniques. In other words, they provide useful finite-state abstractions of (generally) infinite-state systems.

In our concrete setting, the constructed pPDA is guaranteed to be finite if the given program operates on finite-domain variables only (i.e., all program variables are, say, Booleans or bounded integers). The translation can then be automated; in fact, we have implemented it in our tool Pray. The resulting finite pPDA are subject to the techniques described in previous chapters of this thesis and in the literature;[1] in particular, we could compute certificates for various properties (see Chapters 4 and 5) and model check programs against CaRet specifications (see Chapter 6).[2]

1: Brázdil et al. [21] provide an overview of many existing analyses.

2: Our operational pPDA satisfy the *visibility restrictions* needed in Chapter 6; see Remark 7.20 on page 208.

Chapter Outline, Contributions, and Origins This chapter comprises two main sections:

Section 7.1 (Basic Probabilistic Programs) We start by introducing a basic probabilistic programming language without procedures and recursion. Its semantics is defined in terms of DTMCs (discrete-time Markov chains).[3]

3: See Definition 2.38 on page 35.

Section 7.2 (Procedural Probabilistic Programs) We augment the basic language with procedures and—possibly recursive—calls and define its semantics in terms of pPDA.

As usual, we defer open questions and additional bibliographic remarks to Sections 7.3 and 7.4, respectively.

This chapter has not been covered in any of our prior publications [131, 132, 134, 135]. However, our procedural probabilistic language and its pPDA semantics are similar to existing accounts by, e.g., Lehmann [89], Matheja [95], and Olmedo et al. [105].[4] Compared to the existing literature, the novelty of this chapter lies mainly in its presentation, the illustrative examples, and the concrete implementation of the formal semantics in our tool Pray.

[89]: Lehmann (2023), "Pushdown and expectation transformer semantics of probabilistic recursive programs with nested conditioning".

[95]: Matheja (2020), "Automated reasoning and randomization in separation logic".

[105]: Olmedo et al. (2016), "Reasoning about Recursive Probabilistic Programs".

4: We discuss some technical differences to these works in Section 7.4.

7.1 Basic Probabilistic Programs

In this section, we define a basic imperative probabilistic programming language. We do not consider procedures and recursion yet—these features will be added in Section 7.2.

Our basic language incorporates several standard constructs such as `if-else` statements and `while` loops. Basic programs are thus similar to very simple C-style programs. Additionally, our language provides randomness in the form of a *coin flip* command $\{\ldots\}\ [p]\ \{\ldots\}$, which executes the left branch with probability p and the right branch with probability $1-p$.

► **Example 7.1** (Von Neumann's fair coin) The following is an example of a basic program in our language:

```
// x and y implicitly initialized with 0
while (x = y) {
    {x := 0} [0.49] {x := 1} ;
    {y := 0} [0.49] {y := 1} ;
}
```

The program repeatedly flips a slightly *biased* coin twice until obtaining different outcomes. However, it can be shown that, upon program termination, we have $x = 0$ and $x = 1$ with probability *exactly* 0.5 each.[5] The program thus simulates a *fair* coin using a biased[6] one, an idea that goes back to Von Neumann [126]. ◄

5: See Example 7.8 on page 196.

6: The amount of bias is actually irrelevant: The program works for any bias $p \in (0, 1)$.

[126]: Von Neumann (1951), "Various techniques used in connection with random digits".

In the remainder of this section, we describe the syntax and semantics of basic programs formally.

7.1.1 Syntax of Basic Programs

Each program operates on finitely many *variables* taken from a countably infinite[7] set Vars, which we fix throughout the chapter. These variables accept values from a fixed countable set Vals. For the sake of concreteness, we assume that $0 \in \text{Vals} \subseteq \mathbb{Z}$, which is, however, not essential: We could also work with, for example, real-valued, string-valued, or even list-valued variables. Variables do not have to be declared and are implicitly initialized with 0.[8] A *memory state* is a function

$$\sigma : \text{Vars} \to \text{Vals}$$

such that only finitely many $x \in \text{Vars}$ satisfy $\sigma(x) \neq 0$. For memory states, we use notation such as, for example, $\{x \mapsto 0, y \mapsto 1\}$, where we may omit variables mapped to 0 and (if there is no risk of confusion) the curly braces.[9] Σ denotes the set of all memory states.

Expressions over program variables are used to query the current memory state. For simplicity, we do not fix a concrete syntax for expressions and view them directly as functions. We distinguish two types of expressions: *arithmetic* and *Boolean* expressions, which are functions

$$E : \Sigma \to \text{Vals} \qquad \text{and} \qquad B : \Sigma \to \mathbb{B}\,,$$

respectively. We denote the set of all supported arithmetic expressions by $\mathcal{E}$ and the set of Boolean expressions by $\mathcal{B}$. For technical reasons, we assume these sets to be countable.[10] In the examples throughout the text, we use a self-explanatory ad hoc syntax for expressions. For instance, $x + y$ denotes the arithmetic expression $\lambda\sigma.\sigma(x) + \sigma(y)$, whereas $x > 0$ denotes the Boolean expression B satisfying $B(\sigma) = \mathsf{true}$ if and only if $\sigma(x) > 0$, and so on.

Definition 7.2 ***(Syntax of basic programs)*** We define the set Cmd of *(basic) commands* (or *basic programs*) according to the following context-free grammar:[11]

C ::=	`skip`	Effectless program
|	x `:=` E	Assignment ($x \in \text{Vars}$, $E \in \mathcal{E}$)
|	C `;` C	Sequential composition
|	`if (`B`) {`C`} else {`C`}`	Conditional choice ($B \in \mathcal{B}$)
|	`while (`B`) {`C`}`	Loop ($B \in \mathcal{B}$)
|	`{`C`} [`p`] {`C`}`	Probabilistic choice ($p \in (0,1) \cap \mathbb{Q}$)

△

7: We require an *infinite* set Vars to enable writing programs with arbitrarily many variables.

8: Technically, variable initialization is a semantic aspect; it is formalized in Definition 7.6 on page 194.

9: The memory state mapping *all* variables to 0 is often denoted with {}.

10: In practice, where we would (be forced to) work with a concrete syntax for expressions, this assumption is satisfied.

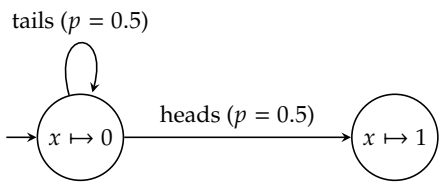
11: C is the only nonterminal symbol in this grammar.

Notice that there are *atomic* commands (`skip`, assignment) and *composite* commands (all the others). Since the sets Vars, $\mathcal{E}$, $\mathcal{B}$, and $(0,1) \cap \mathbb{Q}$ are all countable, Cmd is countable as well.

► **Example 7.3** (Flipping a coin until seeing heads) Consider the following basic program C:

```
while (x = 0) {
    {x := 1} [0.5] {skip}
}
```

Recall that x is implicitly initialized with 0. Intuitively, the program thus models the process of flipping a fair coin until, say, heads comes up for the first time. This is visualized in the following diagram:



The diagram corresponds to a DTMC as introduced in Chapter 2 with a fixed initial distribution assigning probability 1 to the leftmost state.[12] Notice that the DTMC is *improper* because the rightmost state has no outgoing probability mass; the program terminates when reaching that state. ... ◄

12: We recall the definition of DTMCs on on the next page.

In general, the combination of unbounded `while`loops and randomness as showcased in Example 7.3 gives rise to remarkably complex stochastic systems. In the following sections, we formalize a systematic translation from basic programs to DTMCs.

7.1.2 Markov Chains as an Operational Model

There are usually many ways to assign sensible mathematical meaning (semantics) to the programs from a given language [140]. Our basic probabilistic language defined in Section 7.1.1 is no exception to this rule. In this thesis, however, we discuss only *small-step operational semantics*.[13] This semantic model arises from the following postulates:

[140]: Winskel (1993), *The Formal Semantics of Programming Languages: An Introduction*.

13: A popular alternative is *denotational semantics*; see e. g. [35].

1. The meaning of a program is fully defined by describing how it transforms the internal state of an abstract interpreting machine step-by-step.

14: Cf. Equation (6.1) on page 170.

2. These execution sequences have the *Markov property*:[14] The probability of each possible next state depends *only* on the current state. In particular, the probability neither depends on the history of previous states,[15] nor on any external factors.

15: Pictorially speaking, this implies that *coins do not wear off* over time.

In probability theoretical terms, we can thus view a program's semantics as a sequence of *random variables* satisfying the Markov property, that is, as a DTMC. These random variables correspond to the (random) internal states of the interpreting machine.

In contrast to Chapter 2, all DTMCs have a fixed initial distribution in the present chapter; a DTMC can thus be defined as a triple $\mathfrak{M} = (S, M, \mathbf{g})$, where:[16]

16: Cf. Definition 2.38 on page 35.

- S is a countable nonempty set of *states*.
- $M\colon S \times S \to [0,1]$ is a (row-)substochastic *transition probability matrix*, that is,

$$\forall s \in S\colon \quad \sum_{t \in S} M(s,t) \;\leq\; 1\,. \tag{7.1}$$

- $\mathbf{g}\colon S \to [0,1]$ is an *initial distribution* satisfying $\sum_{s \in S} \mathbf{g}(s) = 1$.

Recall further that, if S is finite, then $\mathfrak{M}$ is called *finite(-state)* and if the sum in Equation (7.1) is equal to 1 for all $s \in S$, then $\mathfrak{M}$ is called *proper*. States for which the sum is equal to 0 are called *terminal*. We often write

$$s \xrightarrow{p}_{\mathfrak{M}} t$$

to indicate that $M(s,t) = p$; we may drop the subscript $\mathfrak{M}$ if the DTMC is clear from the context.

► **Example 7.4** (Example 7.3 continued) The diagram from Example 7.3 (restated in the margin) can be viewed as a finite DTMC with states $S = \{\{x \mapsto 0\}, \{x \mapsto 1\}\}$, substochastic transition probability matrix

$$M \;=\; \begin{pmatrix} 0.5 & 0.5 \\ 0 & 0 \end{pmatrix}\,,$$

and initial distribution $\mathbf{g} = (1\ 0)$. For the above tabular representation of M and $\mathbf{g}$, we have employed the state ordering $\{x \mapsto 0\} < \{x \mapsto 1\}$. Notice that state $x \mapsto 1$ is terminal. ◄

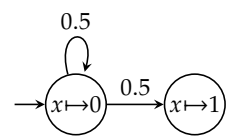


Figure 7.1: Diagram from Example 7.3 restated.

The next definition plays a key role in defining the operational DTMC semantics of basic programs in Section 7.1.3:

Definition 7.5 ***(Reachable fragment)*** The reachable fragment of a DTMC $\mathfrak{M} = (S, M, \mathbf{g})$ is the DTMC $\mathfrak{M}' = (S', M', \mathbf{g}')$, where S' is the smallest set satisfying the following conditions:

1. For all $s \in S$, if $\mathbf{g}(s) \neq 0$, then $s \in S'$.
2. For all $s \in S'$, if $s \xrightarrow{p}_{\mathfrak{M}} t$ for $p \neq 0$ and $t \in S$, then $t \in S'$.

Furthermore, M' and $\mathbf{g}'$ are the restrictions of M and $\mathbf{g}$ to $S' \times S'$ and S', respectively.[17] △

17: It is straightforward to verify that $\mathfrak{M}'$ is again a DTMC.

Intuitively, the reachable fragment comprises the states that can be reached with *positive probability* within arbitrary many steps from the initial distribution. Note that the reachable fragment of an infinite DTMC may be finite.

7.1.3 Semantics of Basic Programs

We now describe the formal translation of basic programs to DTMCs, i.e., the operational semantics. First, we have to clarify how the internal states—subsequently called *programs states*—of the interpreting machine[18] look like. The basic idea is as follows:

18: See page 190.

$$\text{Program state} \;=\; \text{next command (or } \textit{done}\text{) \& memory state.}$$

We thus define the set of all program states formally as

$$S_{\text{Prog}} \quad = \quad \big(\mathsf{Cmd} \cup \{\downarrow\}\big) \times \Sigma\,.$$

19: See Definition 7.2 on page 189.

Here, Cmd is the set of basic programs,[19] and the special symbol $\downarrow$ indicates that there is no next command, that is, the program has terminated. We denote individual program states $\langle C \,|\, \sigma\rangle \in S_{\text{Prog}}$ with angle brackets and a separating vertical bar.

Next, we construct a *global* transition probability matrix

$$M_{\text{Prog}}\colon\; S_{\text{Prog}} \times S_{\text{Prog}} \;\to\; [0,1]\,.$$

20: This is technically more convenient than modelling just an *individual* program.

The idea is that M_{Prog} models the behavior of *all* basic programs.[20] To define M_{Prog}, we need to specify how any given program state transitions to the next one, and with which probability. This is the purpose of the rules in Figure 7.2. Note that these rules are all of the form

$$(\textsc{Name})\;\frac{\text{premise}}{\text{consequence}}$$

$$(\textsc{Skip})\ \frac{}{\langle \texttt{skip} \,|\, \sigma\rangle \xrightarrow{1} \langle \downarrow \,|\, \sigma\rangle} \qquad (\textsc{Assign})\ \frac{}{\langle x := E \,|\, \sigma\rangle \xrightarrow{1} \langle \downarrow \,|\, \sigma \cup \{x \mapsto E\}\rangle}$$

$$(\textsc{Seq}_1)\ \frac{\langle C_1 \,|\, \sigma\rangle \xrightarrow{p} \langle \downarrow \,|\, \sigma'\rangle}{\langle C_1 \,;\, C_2 \,|\, \sigma\rangle \xrightarrow{p} \langle C_2 \,|\, \sigma'\rangle} \qquad (\textsc{Seq}_2)\ \frac{\langle C_1 \,|\, \sigma\rangle \xrightarrow{p} \langle C_1' \,|\, \sigma'\rangle}{\langle C_1 \,;\, C_2 \,|\, \sigma\rangle \xrightarrow{p} \langle C_1' \,;\, C_2 \,|\, \sigma'\rangle}$$

$$(\textsc{If}_1)\ \frac{B(\sigma) = \mathsf{true}}{\langle \texttt{if}\,(B)\,\{C_1\}\,\texttt{else}\,\{C_2\} \,|\, \sigma\rangle \xrightarrow{1} \langle C_1 \,|\, \sigma\rangle} \qquad (\textsc{If}_2)\ \frac{B(\sigma) = \mathsf{false}}{\langle \texttt{if}\,(B)\,\{C_1\}\,\texttt{else}\,\{C_2\} \,|\, \sigma\rangle \xrightarrow{1} \langle C_2 \,|\, \sigma\rangle}$$

$$(\textsc{While}_1)\ \frac{B(\sigma) = \mathsf{true}}{\langle \texttt{while}\,(B)\,\{C\} \,|\, \sigma\rangle \xrightarrow{1} \langle C \,;\, \texttt{while}\,(B)\,\{C\} \,|\, \sigma\rangle}$$

$$(\textsc{While}_2)\ \frac{B(\sigma) = \mathsf{false}}{\langle \texttt{while}\,(B)\,\{C\} \,|\, \sigma\rangle \xrightarrow{1} \langle \downarrow \,|\, \sigma\rangle}$$

$$(\textsc{Prob}_1)\ \frac{C_1 \neq C_2}{\langle \{C_1\}\,[p]\,\{C_2\} \,|\, \sigma\rangle \xrightarrow{p} \langle C_1 \,|\, \sigma\rangle} \qquad (\textsc{Prob}_2)\ \frac{C_1 \neq C_2}{\langle \{C_1\}\,[p]\,\{C_2\} \,|\, \sigma\rangle \xrightarrow{1-p} \langle C_2 \,|\, \sigma\rangle}$$

$$(\textsc{Prob}_3)\ \frac{}{\langle \{C\}\,[p]\,\{C\} \,|\, \sigma\rangle \xrightarrow{1} \langle C \,|\, \sigma\rangle}$$

Figure 7.2: Rules defining the operational DTMC semantics of basic programs.

and should be read as *implications* that are universally quantified in all free symbols.[21] For example, the rule ($\textsc{If}_1$) means that

$$\forall B \in \mathcal{B},\ \sigma \in \Sigma,\ C_1 \in \mathsf{Cmd},\ C_2 \in \mathsf{Cmd}:$$
$$B(\sigma) = \mathsf{true} \implies \langle \texttt{if}\,(B)\,\{C_1\}\,\texttt{else}\,\{C_2\} \,|\, \sigma\rangle \xrightarrow{1} \langle C_1 \,|\, \sigma\rangle\,.$$

In words, if a Boolean expression B holds true in a memory state σ, then for arbitrary commands C_1 and C_2, the transition matrix M_{Prog} has a transition with probability 1 from program state $\langle \texttt{if}\,(B)\,\{C_1\}\,\texttt{else}\,\{C_2\} \,|\, \sigma\rangle$ to $\langle C_1 \,|\, \sigma\rangle$. This resembles our intuition about `if-else` statements.

We briefly go over some of the remaining rules in Figure 7.2. Most of them are intuitive, self-explanatory, and standard in the literature on operational semantics [107].

($\textsc{Assign}$) The atomic command $x := E$ terminates in one step and updates the current memory state $\sigma \in \Sigma$ by setting the value of

21: Notice that it is possible for a rule to have an empty premise.

[107]: Plotkin (2004), "The origins of structural operational semantics".

$x \in \text{Vars}$ to $E(\sigma)$. To this end, we define the notation

$$\sigma \cup \{x \mapsto E\} \stackrel{\text{def}}{=} \lambda y. \begin{cases} E(\sigma) & \text{if } y = x\,, \\ \sigma(y) & \text{if } y \neq x\,. \end{cases}$$

(Seq$_i$) These rules reflect the intuition that in order to execute C_1 ; C_2 we begin by executing C_1 for one step. If C_1 terminates after one step, we continue by executing C_2 (Seq$_1$). If C_1 does not terminate after one step and instead leads us to a program state with command C_1', then we continue with C_1' ; C_2 (Seq$_2$).

(While$_i$) A while loop whose guard evaluates to false in the current memory state terminates in one step without altering the memory (While$_2$). Conversely, if the loop guard evaluates to true, then we execute the loop body at least once, and afterwards again check whether the loop guard holds, and so on. This behavior can be expressed via sequential composition as shown in (Seq$_2$).

In summary, the operational DTMC semantics of basic programs is defined as follows:

Definition 7.6 ***(Operational DTMC semantics of basic programs)***

A. The set of *program states*[22] is defined as

$$S_{\text{Prog}} \;=\; (\mathsf{Cmd} \cup \{\downarrow\}) \times \Sigma\,.$$

B. The *global transition probability matrix*

$$M_{\text{Prog}} \colon S_{\text{Prog}} \times S_{\text{Prog}} \to [0,1]$$

is the unique substochastic matrix that adheres to the rules in Figure 7.2 and in which all program states of the form $\langle\downarrow \mid \sigma\rangle$, $\sigma \in \Sigma$, are terminal.[23]

C. The *operational DTMC* $\mathfrak{M}_C$ of a basic program $C \in \mathsf{Cmd}$ is defined as the reachable fragment[24] of the *global DTMC* $(S_{\text{Prog}}, M_{\text{Prog}}, \mathbf{g}_C)$, where $\mathbf{g}_C$ assigns initial probability 1 to $\langle C \mid \{\}\rangle$, and 0 to all other program states.[25] △

22: Notice that the set of program states is countably infinite.

23: *Terminal* means that these states have no outgoing transitions; see page 191.

24: See Definition 7.5.

25: Recall that {} denotes the memory state where all variables are set to 0.

► **Example 7.7** (Operational DTMC) Reconsider the basic program C from Example 7.3:

```
while (x = 0) { {x := 1} [0.5] {skip} }
```

The operational DTMC $\mathfrak{M}_C$ of C is shown in Figure 7.3. Notice that, compared to the (informal) DTMC from Example 7.3, the formal operational DTMC has several additional states representing intermediate program steps, such as, loop guard evaluations. ◄

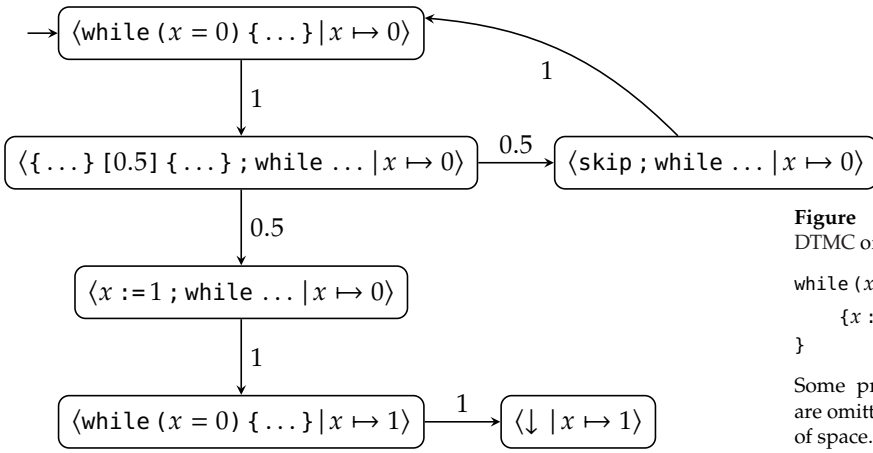


Figure 7.3: Operational DTMC of the basic program

```
while (x = 0) {
    {x := 1} [0.5] {skip}
}
```

Some program fragments are omitted (...) due to lack of space.

7.1.4 Program Verification via Operational DTMCs

Once we have constructed the operational DTMC $\mathfrak{M}_C$ of a basic program C, we can use it to verify various properties of C. In particular, we can compute (termination) probabilities and expected runtimes as explained in detail in Chapter 2.[26] Les us briefly recall how these properties can be encoded as behaviors[27] of the DTMC $\mathfrak{M}_C$:

- The behavior $[\![\mathbf{g}_C \,, \mathfrak{M}_C \,, \mathbf{t}]\!]$, where

$$\mathbf{t} \;=\; \mathbf{1} - M_C \cdot \mathbf{1}$$

  is the vector of *immediate termination probabilities*[28] of $\mathfrak{M}_C$, is the *termination probability* of C, that is, the probability of eventually reaching a program state of the form $\langle\downarrow \mid \sigma\rangle$, where σ is arbitrary.
- The behavior $[\![\mathbf{g}_C \,, \mathfrak{M}_C \,, \mathbf{e}_{\langle\downarrow \mid \sigma\rangle}]\!]$ is the probability that C terminates in a given memory state σ.[29]
- The behavior $[\![\mathbf{g}_C \,, \mathfrak{M}_C \,, \mathbf{1} - \mathbf{t}]\!]$ is the *expected runtime*[30] of C, that is, the expected number of steps until termination (which may be infinite).

We illustrate the verification of basic programs via their operational DTMCs with the following example:

26: See Section 2.3 on page 35.

27: See Definition 2.28 on page 30.

28: See Definition 2.40 on page 37.

29: Recall that $\mathbf{e}_{\langle\downarrow \mid \sigma\rangle}$ is the characteristic (or indicator) vector of $\mathbf{e}_{\langle\downarrow \mid \sigma\rangle}$; it assigns 1 to $\langle\downarrow \mid \sigma\rangle$ and 0 to all other program states.

30: See Definition 2.45 on page 39.

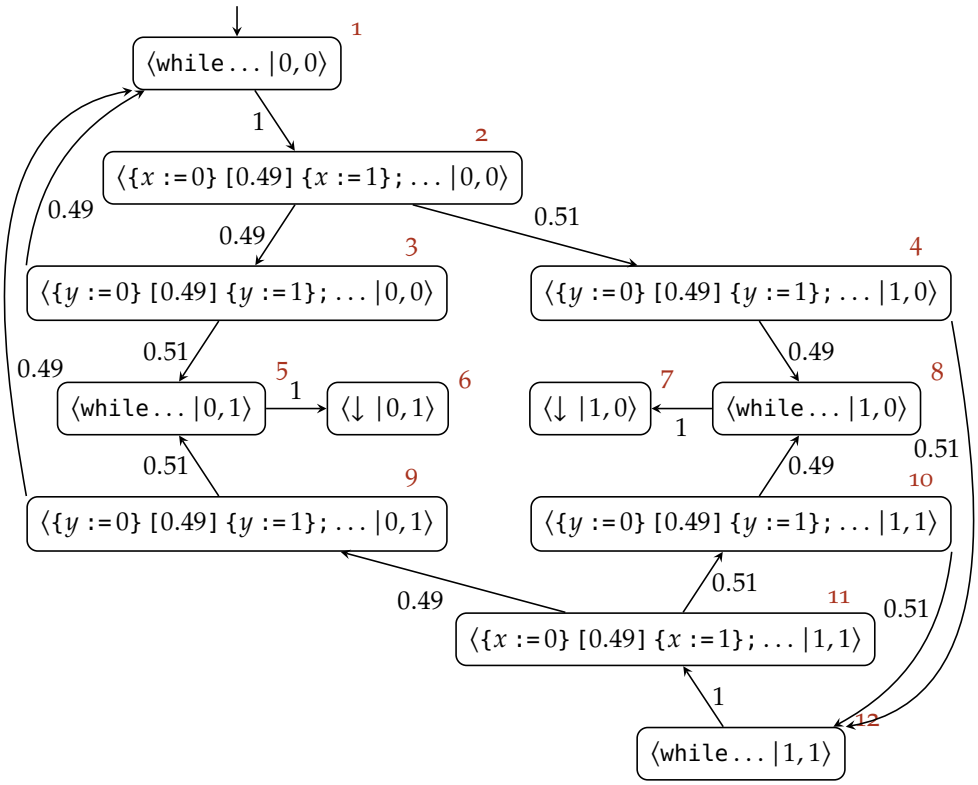


Figure 7.4: Operational DTMC of the basic program from Example 7.1 (Von Neumann's fair coin; restated in Figure 7.5). Memory states are denoted as a, b instead of $\{x \mapsto a, y \mapsto b\}$ to save space. The colored auxiliary labels next to the states are used in Example 7.8.

► **Example 7.8** (Correctness of Von Neumann's coin) We formally prove that the basic program C from Example 7.1 on page 188 (restated in the margin) terminates with probability 1 and that the probability of terminating in a memory state satisfying $x = 0$ is equal to the probability of terminating in a memory state satisfying $x = 1$.

```
while (x = y) {
  {x := 0} [0.49] {x := 1} ;
  {y := 0} [0.49] {y := 1} ;
}
```

Figure 7.5: Von Neumann's program from Example 7.1 restated.

We begin by constructing the operational DTMC $\mathfrak{M}_C$ of C. It has 12 states and is shown in Figure 7.4.

Next, to compute the termination probability (i. e., the behavior $[\![\mathbf{g}_C, \mathfrak{M}_C, \mathbf{t}]\!]$), we recall the following standard approach from Chapter 2: We first determine the *backward behavior*[31] $(\!|\mathfrak{M}_C, \mathbf{t}|\!)$ and then obtain the termination probability as $[\![\mathbf{g}_C, \mathfrak{M}_C, \mathbf{t}]\!] = \mathbf{g}_C{}^{\top} \cdot (\!|\mathfrak{M}_C, \mathbf{t}|\!)$. The backward behavior is a vector—with one entry per state of $\mathfrak{M}_C$—constituting the (componentwise) least nonnegative solution of the system of equations $M_C \cdot \mathbf{x} + \mathbf{t} = \mathbf{x}$,[32] which can be written explicitly as follows (the subscripts correspond to the

31: See Definition 2.28 on page 30.

32: See Proposition 2.31 on page 32.

colored labels in Figure 7.4):

$$\begin{aligned} x_1 &= x_2 & x_7 &= 1 \\ x_2 &= 0.49x_3 + 0.51x_4 & x_8 &= x_7 \\ x_3 &= 0.51x_5 + 0.49x_1 & x_9 &= 0.51x_5 + 0.49x_1 \\ x_4 &= 0.49x_8 + 0.51x_{12} & x_{10} &= 0.49x_8 + 0.51x_{12} \\ x_5 &= x_6 & x_{11} &= 0.49x_9 + 0.51x_{10} \\ x_6 &= 1 & x_{12} &= x_{11} \end{aligned}$$

In this example, the system has a unique solution, the constant vector $\mathbf{x} = \mathbf{1}$. Thus, $\langle\!\langle \mathfrak{M}_C , \mathbf{t} \rangle\!\rangle = \mathbf{1}$ and consequently $[\![\mathbf{g}_C , \mathfrak{M}_C , \mathbf{t}]\!] = 1$, that is, C terminates with probability 1 as claimed.

To prove that the program indeed simulates a fair coin, we compute the behaviors $[\![\mathbf{g}_C , \mathfrak{M}_C , \mathbf{e}_{\langle\downarrow|0,1\rangle}]\!]$ and $[\![\mathbf{g}_C , \mathfrak{M}_C , \mathbf{e}_{\langle\downarrow|1,0\rangle}]\!]$, which we claim are both equal to 0.5. This can be shown similarly to the previous calculation; in fact, we only have to replace the equations $x_7 = 1$ by $x_7 = 0$ and $x_6 = 1$ by $x_6 = 0$, respectively. In the first case, the solution is

$$\begin{aligned} &\langle\!\langle \mathfrak{M}_C , \mathbf{e}_{\langle\downarrow|0,1\rangle}]\!] \\ &\quad = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & \frac{151}{200} & \frac{51}{200} & 1 & 1 & 0 & 0 & \frac{151}{200} & \frac{51}{200} & \frac{1}{2} & \frac{1}{2} \end{pmatrix}^\top \end{aligned}$$

and thus $[\![\mathbf{g}_C , \mathfrak{M}_C , \mathbf{e}_{\langle\downarrow|0,1\rangle}]\!] = 1/2$. In the second case, the solution is similar and also satisfies $[\![\mathbf{g}_C , \mathfrak{M}_C , \mathbf{e}_{\langle\downarrow|1,0\rangle}]\!] = 1/2$. ◀

In praxis, computations on DTMCs as in the preceding Example 7.8 can be performed efficiently with probabilistic model checking tools such as Storm [67] or PRISM [88]. This approach requires rephrasing the desired behaviors (termination probabilities, expected runtimes, etc.) as a suitable reachability or expected reward property.[33]

[67]: Hensel et al. (2022), "The probabilistic model checker Storm".

[88]: Kwiatkowska et al. (2011), "PRISM 4.0: Verification of Probabilistic Real-Time Systems".

33: See Remark 2.43 on page 38.

7.1.5 Examples: Infinite and Exploding State Spaces

In this section, we discuss some slightly more advanced example programs and their operational DTMC semantics.

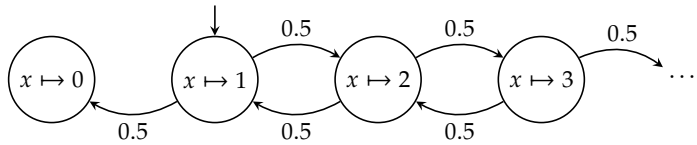
► **Example 7.9** (Infinitely many reachable program states) The program

```
x := 1 ;
while (x > 0) {
    {x := x − 1} [0.5] {x := x + 1}
}
```

encodes a DTMC which is structurally similar[34] to the following one, a symmetric random walk on $\mathbb{N}$ (this example assumes that $\mathbb{N} \subseteq$ Vals):[35]

34: The formal operational DTMC has additional intermediate states.

35: Cf. Example 2.50 on page 41.



Notice that unlike the previous examples, this DTMC has *infinitely many states*. It is thus impossible to construct and analyze the program's operational DTMC explicitly. ◀

Given the existence of infinite-state programs as in Example 7.9, we note that our programming language, albeit simple, is *Turing-complete*—at least if we assume that $\mathbb{N} \subseteq \text{Vals}$. Indeed, it is straightforward to encode a (non-probabilistic) *two-counter machine* as a `while` loop with two counter variables, an additional variable encoding the state of the machine, and appropriate `if-else` constructs to encode the transitions. It is thus undecidable whether a given basic program terminates, even if the program is deterministic. In the probabilistic case, termination is in fact even *more undecidable* (in a precise sense formalized via the *arithmetical hierarchy*) [75].

[75]: Kaminski et al. (2019), "On the hardness of analyzing probabilistic programs".

On the other hand, if we restrict to variables that may take on at most finitely many distinct values (i.e., we assume $|\text{Vals}| < \infty$), then all operational DTMCs are effectively finite. More precisely, let C be a basic program, define $|C|$ as the number of commands in C, and let $\text{Vars}(C)$ denote the variables occurring in C. Then the operational DTMC $\mathfrak{M}_C = (S, M)$ of C satisfies

$$|S| \in O\left(|C| \cdot |\text{Vals}|^{|\text{Vars}(C)|}\right) . \tag{7.2}$$

Consequently, the halting problem—along with myriad other verification problems—becomes decidable for such finite-state programs; they are amenable to standard model checking techniques for finite DTMCs (e.g., [8, Chapter 10]).

[8]: Baier et al. (2008), *Principles of Model Checking*.

Note that the bound in Equation (7.2) is exponential in the number of variables, leading to the well-known *state explosion problem* in model checking (see, e.g., [34]). The practical implication of state explosion is that programs with only a handful of variables may already give rise to large and complex state spaces. For instance, the program in the next example has three variables—one of which remains essentially constant—yet its behavior is rather intricate:

[34]: Clarke et al. (2011), "Model Checking and the State Explosion Problem".

► **Example 7.10** (From coins to dice) Consider the following program, the *Fast Dice Roller* algorithm [92]:[36]

[92]: Lumbroso (2013), "Optimal Discrete Uniform Generation from Coin Flips, and Applications".

36: The variant presented here, which adheres to our syntax, is due to Zilken [144].

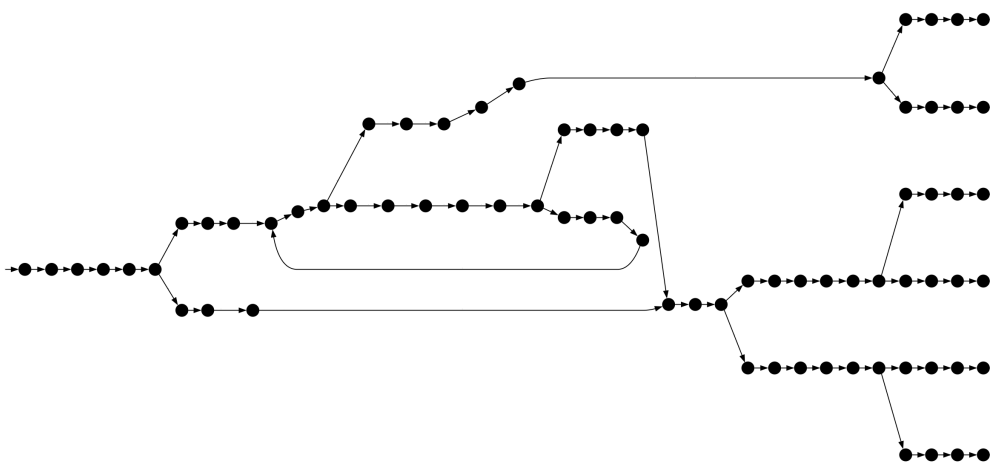


Figure 7.6: Operational DTMC of the *Fast Dice Roller* algorithm from Example 7.10.Transition probabilities and state labels are omitted to avoid clutter. The six possible outcomes correspond to the six terminal states on the right. The figure was generated automatically with PRAY.

```
n := 6 ;
v := 1 ; c := 0 ;
while (v < n) {
    v := 2v ;
    {c := 2c} [0.5] {c := 2c + 1} ;
    if (c ≥ n) {
        v := v − n ;
        c := c − n
    } else { skip }
}
```

It can be shown that the program generates a *uniform* distribution over the n possible final values $0, 1, \ldots, n-1$ of the variable c [92].[37] The version shown above fixes $n = 6$ and thus simulates rolling a six-sided die using only a fair coin. The finite operational DTMC of the program is depicted in Figure 7.6. In our JAR 2025 paper [101], we analyzed the DTMC for values of n up to 128,000—the resulting DTMCs have over 6.5 million states—using techniques implemented in the probabilistic model checker STORM [67]. ◀

37: The formal proof is non-trivial and beyond the scope of this thesis; see [145].

[101]: Mertens et al. (2025), "Computing Expected Visiting Times and Stationary Distributions in Markov Chains: Fast and Accurate".

7.2 Procedural Probabilistic Programs

One of the most common approaches to incorporate recursion into a programming language is via *procedures*.[38] A procedure is an

38: Procedures are also known as *functions* or *(sub-)routines*.

encapsulated piece of code that may contain—possibly recursive—*calls* to other procedures or even to itself. Furthermore, procedures may accept inputs and produce outputs.

In this section, we augment the syntax and semantics of our basic language from Section 7.1 with procedures, calls, and hence recursion. Let us look at an example before diving into the formal definitions.

► **Example 7.11** (Von Neumann's coin via recursion) Reconsider Von Neumann's fair coin simulator from Example 7.1 on page 188 (restated in the margin). We can rewrite the `while` loop using a recursive procedure:

```
f(out x) {
    {x := 0} [0.49] {x := 1} ;
    {y := 0} [0.49] {y := 1} ;
    if (x = y) {
        f(x) // recursive call
    } else {skip}
}
```

The use of *out-parameters*—which are somewhat non-standard—will be explained in detail further below. Suffice it to say here that calling the above procedure as `f(z)`, where z is some program variable, establishes either $z = 0$ or $z = 1$ with probability 0.5 each upon return of f. ◄

```
while (x = y) {
  {x := 0} [0.49] {x := 1} ;
  {y := 0} [0.49] {y := 1} ;
}
```

Figure 7.7: Von Neumann's program from Example 7.1 restated.

7.2.1 Syntax of Procedural Programs

In the remainder, we fix a countably infinite set Proc of *procedure identifiers* ranged over by f, g, and so on.[39]

Definition 7.12 ***(Syntax of procedural programs)***

A. A *procedure declaration* is a syntactic construct of the form

$$f(\texttt{in}\, x_1, \ldots, \texttt{in}\, x_n, \texttt{out}\, y_1, \ldots, \texttt{out}\, y_m)\,\{C\}\,,$$

where

- $f \in \mathrm{Proc}$ is a procedure identifier,
- $x_1, \ldots, x_n \in \mathrm{Vars}$, $n \geq 0$, are pairwise distinct *in-parameters*,
- $y_1, \ldots, y_m \in \mathrm{Vars}$, $m \geq 0$, are *out-parameters*,[40] and
- $C \in \mathsf{Cmd}^{+\mathrm{call}}$ (defined in Item B) is the *procedure body*.

39: Recall that we have already fixed a countably infinite set Vars of program variables and a set $\mathcal{E}$ of arithmetic expressions in Section 7.1.1.

40: We neither require `out`-parameters to be pairwise distinct nor `in`- and `out`-parameters to be disjoint. See Example 7.21 on page 209.

b. The set $\mathsf{Cmd}^{+\mathrm{call}}$ of *commands with calls* is defined by extending the syntax of basic commands from Definition 7.2 as follows:

$$\begin{aligned} C ::= {} & \ldots && \text{(Rules from Def. 7.2)} \\ & \mid f(E_1, \ldots, E_n, z_1, \ldots, z_m), && \text{Procedure call} \end{aligned}$$

where $f \in \mathrm{Proc}$ is an identifier, $E_1, \ldots, E_n \in \mathcal{E}$, $n \geq 0$, are arithmetic expressions, and $z_1, \ldots, z_m \in \mathrm{Vars}$, $m \geq 0$, are *pairwise distinct* program variables. These expressions and variables are called *arguments*.

c. A *procedural program* is a pair $(\mathit{Decls}, C_{\mathrm{main}})$, where

- *Decls* is a finite nonempty set of procedure declarations with pairwise distinct identifiers,[41]
- C_{main} is the *main procedure call*, and
- the main call and all calls in the bodies of the procedures in *Decls* have a matching procedure declaration in *Decls*.[42] △

41: This means that our procedures cannot be *overloaded*. This assumption is for technical convenience and not essential.

42: Our programs are thus *well-typed* by definition.

The intuitive meaning of the `in`- and `out`-parameters is as follows:

- When a procedure is called, the `in`-arguments are copied to the corresponding `in`-parameters.
- Dually, when a procedure returns, the `out`-parameters are copied to the corresponding `out`-arguments (the latter have to be variables).

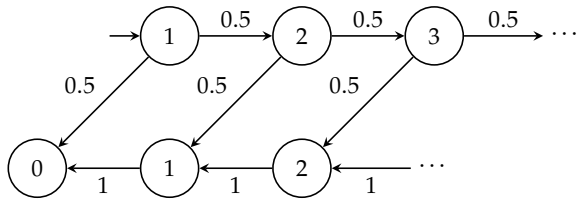
The rest of the syntax is straightforward and familiar. We define the formal semantics of procedural programs in Section 7.2.3.

We represent procedural programs syntactically by listing their declarations and the main call in any arbitrary order.

► **Example 7.13** (A simple probabilisic recursive program) The following procedure f calls itself recursively with probability 0.5:

```
f() {
    { f() } [0.5] { skip }
}
f() // main call
```

Intuitively, the program behaves similar to the following (informal) DTMC where the integers in the states indicate the number of pending calls, that is, the current height of the *call stack*:



The program is similar to the basic one from Example 7.3 (restated in the margin): Both programs simulate flipping a fair coin until seeing, say, heads for the first time. From the perspective of operational semantics, however, there is a crucial difference: The recursive program keeps track of the number of attempts made so far in its call stack. Since this number is unbounded—it is a sample from a *geometric distribution*, which has infinite support—the program gives rise to an *infinite* DTMC. This is in stark contrast to basic programs, which may lead to infinite state spaces only in the presence of variables with an infinite value domain.[43] ◀

```
while (x = 0) {
  {x := 1} [0.5] {skip}
}
```

Figure 7.8: Program from Example 7.3 restated.

43: See Example 7.9 on page 197.

Example 7.13 shows that even rather simple recursive programs can give rise to infinite-state DTMCs, which are difficult to analyze explicitly. This calls for a refined operational model.

7.2.2 Pushdown Automata as an Operational Model

pPDA (probabilistic pushdown automata)[44] extend DTMCs with a data structure: a stack. Therefore, pPDA are a natural candidate for an operational model of our procedural programs. Roughly, the idea is that the pPDA pushes an appropriate piece of data on the stack whenever a procedure call occurs and pops this data upon return of that procedure. We will formalize this intuition in Section 7.2.3. In this section, we recall the basics of pPDA from Chapter 3 for convenience; we also define the notion of a pPDA's *reachable fragment*.

44: pPDA were discussed in detail in Section 3.2 of Chapter 3. We briefly recall the definition below.

7.2.2.1 pPDA Refresher

We recall the definition of pPDA from Section 3.2 (unlike in Section 3.2, the pPDA in this chapter always have a fixed initial state and stack symbol): A pPDA is a 5-tuple

$$\Delta = (Q, \Gamma, P, q_0, Z_0) ,$$

where:

– Q is a countable nonempty set of *states*.

- Γ is a countable nonempty *stack alphabet*. The pairs from $Q \times \Gamma$ and $Q \times \Gamma^*$ are called *heads* and *configurations*, respectively. These pairs are written in juxtaposition and without parentheses.
- P is a stochastic *pushdown transition matrix*, that is,

$$P\colon\ (Q \times \Gamma) \times \left(Q \times \Gamma^{\leq 2}\right) \ \to\ [0,1]$$

such that for all heads qZ we have

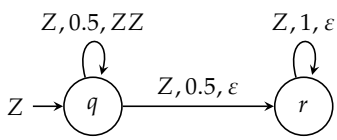
$$\sum_{r\gamma \in Q \times \Gamma^{\leq 2}} P(qZ, r\gamma) \ \leq\ 1\,.$$

We also use the notation

$$qZ \xrightarrow{p}_{\Delta} r\gamma \quad \iff \quad P(qZ, r\gamma) = p\,.$$

- $q_0 \in Q$ is an *initial state* and $Z_0 \in \Gamma$ is an *initial stack symbol*.

► **Example 7.14** (A simple pPDA) The following is (the graphical representation of) a pPDA, which (informally) models the procedural program from Example 7.13 (restated in the margin):



The graphical notation for transitions is as defined Chapter 3; for example, the label $Z, 0.5, ZZ$ on the self loop at state q stands for the transition $qZ \xrightarrow{0.5} qZZ$. In addition to the conventions from Chapter 3, we mark the initial state (here q) with a dangling arrow labeled with the initial stack symbol (here Z). ◄

```
f() {
  { f() } [0.5] { skip }
}
```

Figure 7.9: Program from Example 7.13 restated.

We already saw in Chapter 3 that every pPDA Δ generates a certain countably infinite DTMC $\mathfrak{M}_\Delta$, the *induced DTMC*, which is obtained by encoding the stack's content into the state space.

► **Example 7.15** (Induced DTMC) The DTMC from Example 7.13 on page 201 is the induced DTMC of the pPDA from Example 7.14: Every state n in the upper row of the DTMC corresponds to the configuration qZ^n in the pPDA, and every state n in the lower row corresponds to rZ^n. ◄

7.2.2.2 The Reachable Fragment of a pPDA

We now define the *reachable fragment* of a pPDA, which plays a crucial role in defining the semantics of procedural programs (Section 7.2.3).

As in DTMCs, some states of a pPDA may be unreachable from its initial configuration. Moreover, some stack symbols may never be actually pushed on the stack (e.g., because they are only pushed by an outgoing transition of an unreachable state). Finally, a pPDA may have transitions that never fire because the required combination of state and topmost stack symbol (head) never occurs. Notably, these phenomena are closely intertwined: Whether a given stack symbol can occur on top of the stack at a given state depends on that state's possible predecessors—and thus on the reachable states, which in turn depend again on the reachable transitions, and so on.

Despite these complications, the reachable fragment of a pPDA can be readily defined via its induced DTMC as follows:

Definition 7.16 ***(Reachable fragment of a pPDA)*** Consider a pPDA $\Delta = (Q, \Gamma, P, q_0, Z_0)$.

A. A head $qZ \in Q \times \Gamma$ is *reachable* if some configuration $qZ\gamma$, where $\gamma \in \Gamma^*$ is arbitrary, is reachable in the underlying DTMC $\mathfrak{M}_\Delta$.[45]
B. The *reachable fragment* of Δ is the pPDA $(Q', \Gamma', P', q_0, Z_0)$, where:
   - $Q' = \{q \in Q \mid \exists Z \in \Gamma\colon qZ \text{ is reachable in } \Delta\}$ are the *reachable states* of Δ.
   - $\Gamma' = \{Z \in \Gamma \mid \exists q \in Q\colon qZ \text{ is reachable in } \Delta\}$ are the *reachable stack symbols* of Δ.
   - The pushdown transition matrix P' is defined as[46]

$$P'(qZ, r\gamma) = \begin{cases} P(qZ, r\gamma) & \text{if } qZ \text{ is reachable in } \Delta\,, \\ 0 & \text{otherwise}\,, \end{cases}$$

   where $qZ \in Q' \times \Gamma'$ and $r\gamma \in Q' \times \Gamma'^{\leq 2}$. △

45: See Definition 7.5 on page 192.

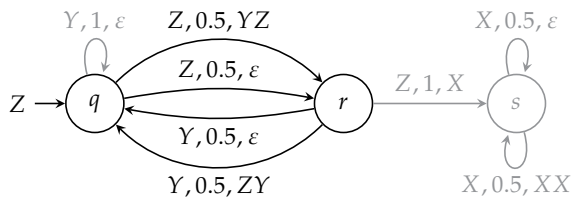
46: The case distinction is necessary because $Q' \times \Gamma'$ may be a *strict* subset of the reachable heads.

It can be verified that a pPDA behaves in the same way as its reachable fragment in the sense that the reachable fragments of their induced DTMCs coincide.

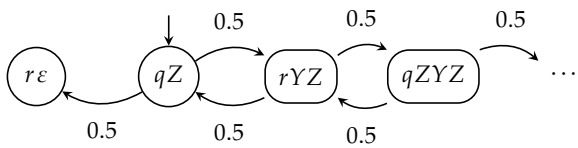
The reachable fragment of a pPDA can also be constructed *algorithmically*, which, however, is not entirely trivial [46]. The crux is the existential quantification over the *infinite* set Γ^* in Definition 7.16A. In particular, it is not at all obvious how to prove that a given head qZ is *not* reachable: Naively, we would have to check that *none* of the—infinitely many—configurations $qZ\gamma$, $\gamma \in \Gamma^*$, are reachable in the underlying infinite DTMC. Existing algorithms for reachability in (non-probabilistic) pushdown systems exploit the fact that the set of reachable configurations is an effectively *regular* language [46]. We do not go into full detail on this topic, but provide some basic intuition by means of an example:

[46]: Esparza et al. (2000), "Efficient Algorithms for Model Checking Pushdown Systems".

► **Example 7.17** (Constructing the reachable fragment of a pPDA) Consider the following pPDA Δ with states $Q = \{q, r, s\}$ and stack alphabet $\Gamma = \{Z, Y, X\}$:



The reachable fragment of Δ comprises the black parts only. To see this, we first observe that the induced DTMC of Δ is as follows:



In particular, the reachable configurations are given by the regular expression

$$qZ(YZ)^* + r(YZ)^* .$$

Thus, following the terminology of Definition 7.16, $Q' = \{q, r\}$ are the reachable states of Δ, $\Gamma' = \{Z, Y\}$ are the reachable stack symbols of Δ, and the set of reachable heads is $\{qZ, rY\}$ (which is a strict subset of $Q' \times \Gamma'$). It follows that the gray parts of Δ can be discarded; they are unreachable. ◄

7.2.3 Semantics of Procedural Programs

In this section, we define the formal operational pPDA semantics of procedural programs.

7.2.3.1 Definition of the pPDA Semantics

We proceed as for basic programs:[47] We first define a *global* infinite pPDA transition matrix capturing the behavior of *all* procedural programs, and then obtain the semantics of a given program as an appropriate reachable fragment.

47: See Section 7.1.3 on page 192.

Definition 7.18 *(Operational semantics of procedural programs)*

A. The set Q_{Prog} of *local program states* and the set of Γ_{Prog} of *stack frames* are both defined identically as

$$Q_{\text{Prog}} \;=\; \Gamma_{\text{Prog}} \;=\; (\mathsf{Cmd}^{+\text{call}} \cup \{\downarrow\}) \times \Sigma\,.$$

Local states $\langle C \,|\, \sigma \rangle \in Q_{\text{Prog}}$ and stack frames $[C \,|\, \sigma] \in \Gamma_{\text{Prog}}$ are denoted with angle and square brackets, respectively.

B. The pushdown transition matrix

$$P_{\text{Prog}}\colon\ \big(Q_{\text{Prog}} \times \Gamma_{\text{Prog}}\big) \times \big(Q_{\text{Prog}} \times \Gamma_{\text{Prog}}^{\leq 2}\big) \;\to\; [0,1]$$

is the unique substochastic matrix that satisfies the following:

(1) P_{Prog} is an extension of M_{Prog}[48] in the sense that if $q \xrightarrow{p} r$ is a transtion in M_{Prog}, then for all $\bigstar \in \Gamma_{\text{Prog}}$, we have a transition $q\bigstar \xrightarrow{p} r\bigstar$ in P_{Prog}.
(2) P_{Prog} adheres to the additional rules in Figure 7.10.

C. Let $(\mathit{Decs}, C_{\text{main}} = f(E_1, \ldots, E_n, z_1, \ldots, z_m))$ be a procedural program. Recall that by Definition 7.12C,

$$f(\texttt{in}\, x_1, \ldots, \texttt{in}\, x_n, \texttt{out}\, y_1, \ldots, \texttt{out}\, y_m)\,\{C\} \in \mathit{Decs}$$

for some $C \in \mathsf{Cmd}^{+\text{call}}$ and $x_1, \ldots, x_n, y_1, \ldots, y_m \in \text{Vars}$. The *operational pPDA* of $(\mathit{Decs}, C_{\text{main}})$ is defined as the reachable fragment[49] of the pPDA

$$\big(\underbrace{Q_{\text{Prog}}}_{\text{states}},\ \underbrace{\Gamma_{\text{Prog}}}_{\text{stack symbols}},\ \underbrace{P_{\text{Prog}}}_{\text{transitions}},\ \underbrace{\langle C \,|\, \sigma \rangle}_{\text{initial state}},\ \underbrace{[C_{\text{main}} \,|\, \{\}]}_{\text{initial stack symbol}}\big)\,,$$

where $\sigma = \{x_1 \mapsto E_1, \ldots, x_n \mapsto E_n\}$ is the variable valuation of the initial (local) program state. △

48: M_{Prog} is the global transition matrix of basic programs defined by the rules in Figure 7.2 on page 193.

49: See Definition 7.16 on page 204.

Let us examine the preceding definition in detail and provide some intuition.

- The main idea is to generalize the DTMC semantics of basic programs.[50] As in the DTMC case, the state space Q_{Prog} of our operational pPDA comprises tuples of commands—including calls and the special termination marker $\downarrow$—and memory states. In the procedural case, these tuples are called *local* program states; they model only the inner state of the currently active procedure.
- To account for the *global* program state, all currently pending calls must be recorded as well. This is achieved using the pPDA's stack: The pPDA pushes local states invoking a procedure call onto the stack and pops these local states upon return of the same procedure. This is formalized in the rules in Figure 7.10, which

50: See Definition 7.6 on page 194.

$$(\textsc{Call})\ \frac{f(\texttt{in}\,x, \texttt{out}\,y)\,\{C\} \in \mathit{Decs} \qquad \bigstar \in \Gamma_{\mathrm{Prog}}}{\langle f(E,z)\,|\,\sigma\rangle\bigstar \xrightarrow{1} \langle C\,|\,\{x \mapsto E(\sigma)\}\rangle\,\big[f(E,z)\,|\,\sigma\big]\,\bigstar}$$

$$(\textsc{Return})\ \frac{f(\texttt{in}\,x, \texttt{out}\,y)\,\{C\} \in \mathit{Decs}}{\langle\downarrow\,|\,\sigma\rangle\,\big[f(E,z)\,|\,\sigma'\big] \xrightarrow{1} \langle\downarrow\,|\,\sigma' \cup \{z \mapsto \sigma(y)\}\rangle}$$

$$(\textsc{SeqPush})\ \frac{\langle C_1\,|\,\sigma\rangle\bigstar \xrightarrow{p} \langle C\,|\,\sigma'\rangle[C_1\,|\,\sigma]\,\bigstar \qquad \bigstar \in \Gamma_{\mathrm{Prog}}}{\langle C_1\,;\,C_2\,|\,\sigma\rangle\bigstar \xrightarrow{p} \langle C\,|\,\sigma'\rangle[C_1\,;\,C_2\,|\,\sigma]\,\bigstar}$$

$$(\textsc{SeqPop}_1)\ \frac{\langle\downarrow\,|\,\sigma\rangle[C_1\,|\,\sigma'] \xrightarrow{p} \langle\downarrow\,|\,\sigma''\rangle}{\langle\downarrow\,|\,\sigma\rangle[C_1;C_2\,|\,\sigma'] \xrightarrow{p} \langle C_2\,|\,\sigma''\rangle} \qquad (\textsc{SeqPop}_2)\ \frac{\langle\downarrow\,|\,\sigma\rangle[C_1\,|\,\sigma'] \xrightarrow{p} \langle C_1'\,|\,\sigma''\rangle}{\langle\downarrow\,|\,\sigma\rangle[C_1;C_2\,|\,\sigma'] \xrightarrow{p} \langle C_1';C_2\,|\,\sigma''\rangle}$$

Figure 7.10: Rules for defining the operational semantics of procedural programs. To avoid clutter, the rules (Call) and (Return) are stated for procedures with exactly one `in`- and one `out`-parameter only. Their generalization to procedures with arbitrarily many parameters is canonical.

we discuss in greater detail in Section 7.2.3.2. The stack alphabet Γ_{Prog} is thus equal to the set Q_{Prog} of local program states.

– The operational pPDA of a given procedural program starts in the local state corresponding to the first command of the main procedure's body and the memory state resulting from the `in`-arguments of the main call C_{main}. The initial stack frame $[C_{\mathrm{main}}\,|\,\{\}]$ consists of the main call itself and the all-zero memory state.

We now consider a first example.

► **Example 7.19** (Operational pPDA) The operational pPDA of the procedural program

```
f() {
    { f() } [0.5] { skip }
}
f() // main call
```

from Example 7.13 on page 201 is depicted in Figure 7.11. Its state space comprises the four local states

$$\langle\{f()\}\,[0.5]\,\{\texttt{skip}\}\,|\,\{\}\rangle\,,\ \langle f()\,|\,\{\}\rangle\,,\ \langle \texttt{skip}\,|\,\{\}\rangle\,,\ \text{and}\ \langle\downarrow\,|\,\{\}\rangle\,,$$

and its stack alphabet contains only the symbol $\big[f()\,|\,\{\}\big]$, which, intuitively, accounts for the number of pending calls to f.

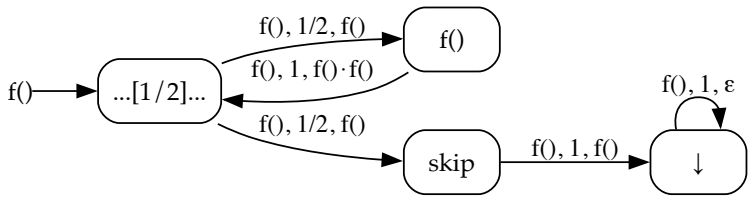


Figure 7.11: Operational pPDA of the program

```
f() {
  { f() } [0.5] { skip }
}
f() // main call
```

The memory states are omitted since the program does not use any variables. The graphical representation was generated automatically with Pray.

Notice that the pPDA is similar to our ad hoc model from Example 7.14 on page 203 except for some intermediate states. ◀

▷ **Remark 7.20** (On visibility) Our operational semantics gives rise to pPDA in which the outgoing transitions of a given state *all perform the same type of stack action*; that is, they all either push a symbol onto the stack, pop a symbol from the stack, or leave the stack unchanged. The states can thus be partitioned into three classes. Indeed, local program states of the form

$$\langle f(\ldots) \mid \ldots \rangle \quad \text{and} \quad \langle f(\ldots); \ldots \mid \ldots \rangle,$$

where $f(\ldots)$ is a procedure call, are *push states*; those of the form $\langle \downarrow \mid \ldots \rangle$ are *pop states*; and all other local states are *internal states*. Our operational pPDA thus satisfy the *visibility restrictions* defined in Section 6.2—they are pVPA (probabilistic visibly pushdown automata)[51] by construction. Procedural programs are therefore within the scope of the model checking techniques presented in Chapter 6; for example, we can determine the probability that a program generates a trajectory satisfying a given CaRet formula.[52] ◁

51: See Definition 6.16 on page 164.

52: Technically, this requires decorating the states of the operational pPDA with suitable labels respecting the states' visibility types (call, pop, internal).

7.2.3.2 Detailed Explanation of the Rules from Figure 7.10

We now explain the rules from Figure 7.10 in greater detail.

The rule

$$(\textsc{Call})\ \frac{f(\texttt{in}\,x, \texttt{out}\,y)\,\{C\} \in \mathit{Decs} \qquad \bigstar \in \Gamma_{\text{Prog}}}{\langle f(E, z) \mid \sigma \rangle \bigstar \xrightarrow{1} \langle C \mid \{x \mapsto E(\sigma)\} \rangle \left[f(E, z) \mid \sigma \right] \bigstar}$$

states that if the current local program state is a call to a procedure f in some memory state σ, then this local state is pushed onto the stack. Furthermore, the pPDA transitions to a local state corresponding to the body of f, where the `in`-parameters are initialized with the corresponding arguments of the call evaluated in σ.

Dually, the rule

$$(\textsc{Return})\ \frac{f(\texttt{in}\,x, \texttt{out}\,y)\,\{C\} \in \mathit{Decs}}{\langle\downarrow \mid \sigma\rangle\left[f(E, z) \mid \sigma'\right] \xrightarrow{1} \langle\downarrow \mid \sigma' \cup \{z \mapsto \sigma(y)\}\rangle}$$

triggers a pop transition when the current local program state is $\downarrow$ (i.e., the current procedure call has terminated) and there is still a pending call on the stack. The pop transition leads to a local successor state where the memory state is updated by copying the values of the `out`-parameters of the called procedure to the corresponding `out`-arguments.

The preceding two rules are insufficient for procedural programs where calls occur in *sequential composition* with other commands (including further calls). The remaining three rules extend them to handle this situation.

First, the rule

$$(\textsc{SeqPush})\ \frac{\langle C_1 \mid \sigma\rangle\bigstar \xrightarrow{p} \langle C \mid \sigma'\rangle[C_1 \mid \sigma]\bigstar \qquad \bigstar \in \Gamma_{\text{Prog}}}{\langle C_1\,;\,C_2 \mid \sigma\rangle\bigstar \xrightarrow{p} \langle C \mid \sigma'\rangle[C_1\,;\,C_2 \mid \sigma]\bigstar}$$

extends rule (Call) to the situation where the current local program state is not merely an *atomic* call command, but a sequential composition of a call followed by some other command (possibly another call).

Second, the rules

$$(\textsc{SeqPop}_1)\ \frac{\langle\downarrow \mid \sigma\rangle[C_1 \mid \sigma'] \xrightarrow{p} \langle\downarrow \mid \sigma''\rangle}{\langle\downarrow \mid \sigma\rangle[C_1;C_2 \mid \sigma'] \xrightarrow{p} \langle C_2 \mid \sigma''\rangle}\qquad (\textsc{SeqPop}_2)\ \frac{\langle\downarrow \mid \sigma\rangle[C_1 \mid \sigma'] \xrightarrow{p} \langle C_1' \mid \sigma''\rangle}{\langle\downarrow \mid \sigma\rangle[C_1;C_2 \mid \sigma'] \xrightarrow{p} \langle C_1';C_2 \mid \sigma''\rangle}$$

are for returns what (SeqPush) is for calls. There are *two* such rules to distinguish between the case where a pop returns to the *end* of a procedure, that is, to a local program state of the form $\langle\downarrow \mid \ldots\rangle$, and the case where execution continues with another command.[53]

Lehmann [89] has shown that a set of rules nearly identical to the ones presented here is consistent, in a precise sense, with an alternative program semantics based on *weakest pre-expectations*.

► **Example 7.21** (Semantics of `in`- and `out`-parameters) Recall that the only restriction regarding parameter declarations is that all `in`-parameters must be pairwise distinct.[54] The procedure declaration

$$\mathit{copy}(\texttt{in}\,x, \texttt{out}\,x, \texttt{out}\,x)\,\{\,\texttt{skip}\,\}$$

53: Alternatively, if we identified "$\downarrow\,;C$" with C, then the rule ($\textsc{SeqPop}_2$) alone would suffice.

[89]: Lehmann (2023), "Pushdown and expectation transformer semantics of probabilistic recursive programs with nested conditioning".

54: See Definition 7.12 on page 200.

is thus syntactically correct. The effect of calling *copy*(`E, y, z`), where E is an expression and y and z are distinct program variables, is to assign the value of E to *both* y and z simultaneously, thereby creating two independent copies of this value. Indeed, upon calling *copy*, the value of the `in`-argument E is copied to the `in`-parameter x, which coincides with the two `out`-parameters. Upon return of *copy*(`E, y, z`), the value of x is then copied to the two `out`-*arguments* y and z.

Similarly, we can declare a procedure

swap(`in` x`, in` y`, out` y`, out` x`) { skip }`

and call it as *swap*(`x, y, x, y`) to swap the values of x and y. Such applications of `in`- and `out`-parameters are a useful programming idiom that adds little complexity to the operational semantics. ◀

7.2.4 Additional Examples

We conclude the technical part of this chapter by presenting some additional illustrative examples of procedural programs and their operational pPDA semantics.

► **Example 7.22** (Random walk revisited) Consider the following procedural program:

```
f() {
    { f() ; f() } [0.5] { skip }
}
f() // main call
```

The procedure f either calls itself twice in sequence or terminates immediately, both with equal probability 0.5. Intuitively, the procedure thus either increases or decreases the height of the call stack by one, thereby performing the familiar symmetric random walk on the stack height.[55]

55: Cf. Example 2.50 on page 41 and Example 7.9 on page 197.

The program's formal operational pPDA is depicted in Figure 7.12. It has five states and two stack symbols: $[f() \mid \{\}]$ and $[f() ; f() \mid \{\}]$ (abbreviated to "$f()$" and "$f() ; \ldots$", respectively). Due to the complexity of the pPDA—it accounts meticulously for all intermediate program steps and, in particular, keeps track of *which* of the two calls to f is currently active—the random walk-like behavior is perhaps not immediately obvious. In fact, the automaton performs a random walk on the *number of occurrences of the symbol "*$f() ; \ldots$*"* on the stack. To see this, notice that the inner cycle (between states $\{\ldots\}\ [1/2]\ \{\ldots\}$ and $f() ; \ldots$ in Figure 7.12) pushes "$f() ; \ldots$" onto the stack, whereas the overall effect of the larger outer cycle is to pop exactly one "$f() ; \ldots$" from the stack; both cycles are taken with probability 0.5.

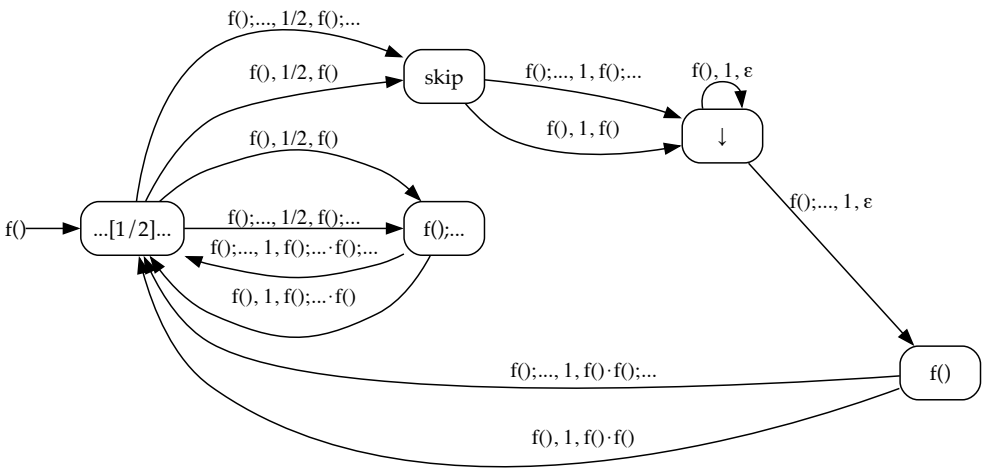


Figure 7.12: Operational pPDA of the random walk program from Example 7.22. The graphical representation was automatically generated with Pray.

Despite these intricacies, with the operational pPDA at hand, it is straightforward to verify several key properties of the program by encoding them as suitable pPDA behaviors, analogous to Section 7.1.4. For example, we can verify that the program terminates with probability 1 and that its expected runtime is infinite. We omit these calculations here for brevity. See Section 3.2 for examples of similar calculations. ◀

► **Example 7.23** (Combining recursion with loops) The following procedure g invokes itself a random, geometrically distributed[56] number of times:

56: Cf. Example 7.13 on page 201.

```
g() {
    while (x = 0) {
        { x := 1 } [0.5] { g() }
    }
}
g() // main call
```

The program thus models a population where each organism produces a geometrically distributed number of offspring, each of which behaves identically to its parent (initially, there is only one organism).

The program's operational pPDA is depicted in Figure 7.13. Notice that the pPDA has inner and outer cyclic structures; intuitively, the former account for the `while` loop and the latter for the recursive calls. Since g calls itself exactly once *on average*, the program terminates with probability 1, a fact which can be verified automatically and formally using the pPDA semantics.[57] ◀

57: In fact, we used a similar program as a benchmark in Chapter 5; see Figure A.2 on page 220.

While the examples presented in this section give rise to relatively

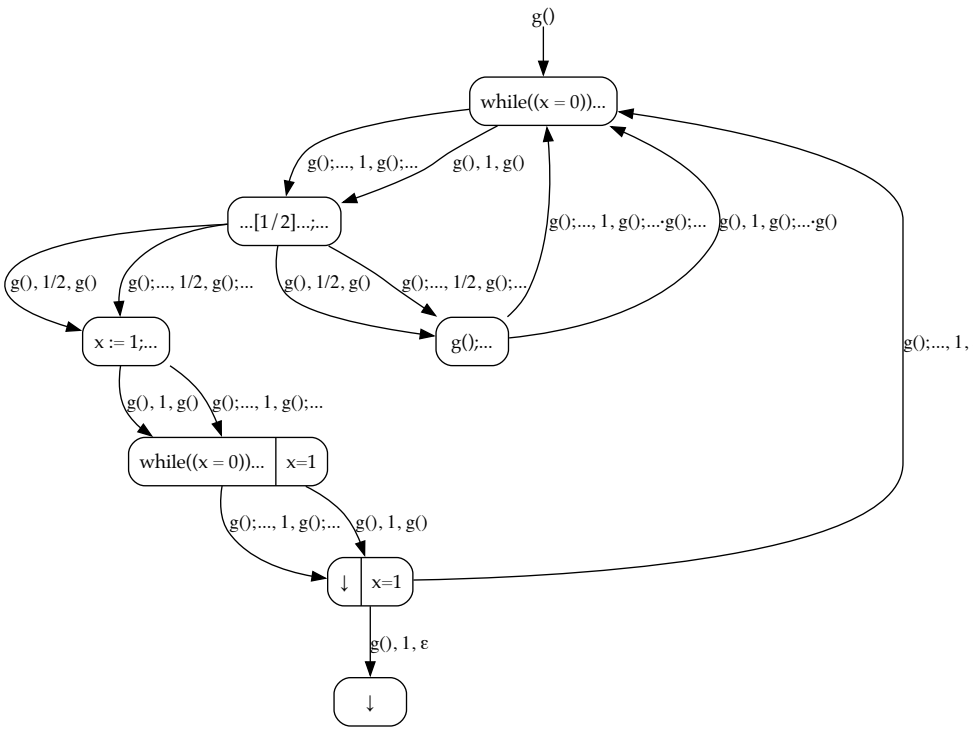


Figure 7.13: Operational pPDA of the program from Example 7.23. The omitted memory states correspond to the all-zero variable valuation. The graphical representation was generated automatically with Pray.

small—yet intricate—pPDA, the remarks about state space explosion (and even infinite state spaces) from Section 7.1.4 still apply in general.

7.3 Open Questions

1. In Chapter 4 we discussed certificates for pPDA. Can we also certify the translation from procedural programs to pPDA?
2. Our operational semantics may lead to automata with many intermediate states, which are irrelevant for certain properties, such as termination. Can we design a more efficient, possibly more abstract, translation to mitigate this problem?
3. Is there a tractable operational model for recursive probabilistic programs with *continuous* distributions?
4. Can we generalize the operational semantics toward procedural *weighted programs* [17]?

[17]: Batz et al. (2022), "Weighted programming: a programming paradigm for specifying mathematical models".

7.4 Additional Bibliographic Remarks

In this section, we discuss only related works specific to the translation of probabilistic programs to automata-like models. See Section 1.3 for a broader discussion of related work on (recursive) probabilistic programs.

The first account of small-step operational semantics for *non-recursive* probabilistic programs similar to the one we presented in Section 7.1 is due to Gretz, Katoen, and McIver [59]. This idea was refined and extended in several follow-up works [12, 14, 16, 72, 74]. Olmedo et al. [105] present a pPDA semantics for recursive probabilistic programs similar to ours, but which does not support procedures with input or output parameters. Matheja [95] presents a richer procedural language but does not provide a semantics in terms of pushdown automata. Closest to our semantics is the one defined in Lehmann's Master's thesis [89], which, however, is limited to one output parameter per procedure and does not explicitly mention the challenges in connection with computing reachable fragments.[58]

The recent tool POPACheck [31, 110] implements a translation from a probabilistic programming language called *MiniProb* to *probabilistic operator precedence automata*, a subclass of pPDA. Besides recursion, MiniProb supports more advanced features such as (nested) conditioning via dedicated `query` and `observe` constructs, which we do not consider here. The translation from MiniProb to automata also relies on an operational semantics [31], which, however, is not described formally in the cited papers.

[59]: Gretz et al. (2014), "Operational versus weakest pre-expectation semantics for the probabilistic guarded command language".

[105]: Olmedo et al. (2016), "Reasoning about Recursive Probabilistic Programs".

[95]: Matheja (2020), "Automated reasoning and randomization in separation logic".

[89]: Lehmann (2023), "Pushdown and expectation transformer semantics of probabilistic recursive programs with nested conditioning".

58: See Section 7.2.2.2 on page 203.

[31]: Chiari et al. (2023), "A Model Checker for Operator Precedence Languages".

[110]: Pontiggia et al. (2025), "POPACheck: A Model Checker for Probabilistic Pushdown Automata".

8 Conclusion and Outlook

We have studied several theoretical and practical aspects of the formal quantitative analysis and verification of probabilistic recursive programs, modeled as probabilistic pushdown automata (pPDA):

- *Certificates* enhance existing pPDA algorithms by providing simple, independently checkable proofs of correctness for their results. We have demonstrated their practical feasibility through a set of pPDA models representing intricate probabilistic recursive programs.[1]
- Our results on *model checking pPDA against ω-visibly pushdown languages* complete a long line of research on the verification of less expressive classes of temporal specifications.[2]

1: See Chapters 4 and 5.

2: See Chapter 6 and references therein.

This thesis thus advances the state-of-the-art in pPDA analysis and, consequently, the verification of probabilistic recursive programs.

Future Work[3] The automata-theoretic approach to probabilistic program verification offers a full algorithmic treatment, but this strength also defines its main limitation: It is not applicable if the program cannot be faithfully modeled as an automaton. In the context of this work, this limitation arises, for example, when a program draws samples from a normal (Gaussian) distribution, requiring $\mathbb{R}$-valued program variables. Another challenge occurs with programs that receive unbounded input (e.g., arbitrarily long lists or arrays, as is common in randomized algorithms); such programs cannot be modeled directly as finite pPDA.

3: The individual chapters contain additional, more specific open questions for future work.

There are at least two ways to tackle more general programs: First, *abstraction* could be used to obtain a finite pPDA model of the program (e.g., by discretizing continuous variables). Second, and perhaps more fundamentally, the class of programs could be broadened by exploring models more expressive than pPDA. For example, there already exists work on pPDA with *nondeterminism*, which can model probabilistic programs with recursion and nondeterministic branching [51]—a useful tool for program abstraction.

On the more practical side, it is worthwhile to explore how pPDA techniques could be integrated as backends into existing verification tools, such as the probabilistic model checkers Storm [67] and mcsta [62]. Indeed, the input language of mcsta, the *Modest modeling language*, supports recursive probabilistic programs and could thus

[51]: Etessami et al. (2015), "Recursive Markov Decision Processes and Recursive Stochastic Games".

[67]: Hensel et al. (2022), "The probabilistic model checker Storm".

[62]: Hartmanns et al. (2014), "The Modest Toolset: An Integrated Environment for Quantitative Modelling and Verification".

benefit from pPDA-based verification methods. Similarly, the deductive probabilistic program verifier Caesar [118] allows recursive programs; in this context, *certificates* are particularly promising for enabling a *formally verified* pPDA backend.

[118]: Schröer et al. (2023), "A Deductive Verification Infrastructure for Probabilistic Programs".

Finally, an intriguing question for future research is whether certificates could be generated using machine learning methods or generative AI tools, such as *large language models*. Certificates are particularly appealing in this context, as they enable formal verification of the claims produced by AI models. A key question is whether such an AI-based approach could be competitive with, or even outperform, traditional algorithms (such as those discussed in Chapter 5).[4] Overall, it remains to be seen whether AI-assisted formal verification—enabled by formally checked proof certificates—will become a new standard in the field.

4: A thorough comparison should also account for factors such as the energy cost of training and deploying AI models.

Appendices

Details on the Experiments | A

In this appendix, I provide additional details on the experimental evaluation of the OVI algorithm presented in Chapter 5.

A.1 Probabilistic Program Benchmarks

The source code of the probabilistic recursive programs whose pPDA models we used as benchmarks in Chapter 5 (see Table 5.1 on page 144) is shown in Figure A.2.

```
bool and() {
  prob {
    1/2: return
      (1/2: true | 1/2: false);
    1/2: {
      if(!or()) return false;
      else return or();
    }
  }
}

{ and(); } // main block
```

(a) and-or

```
bool or() {
  prob {
    1/2: return
      (1/2: true | 1/2: false);
    1/2: {
      if(and()) return true;
      else return and();
    }
  }
}
```

```
void f() {
  if(flip(p)) {
    f(); f();
  }
}

// main block
{ f(); }
```

(b) rw-p

```
void f() {
  if(flip(1/2)) {
    f(); f(); f();
  }
}

// main block
{ f(); }
```

(c) golden

```
void offspring() {
  while(flip(2/5)) {
    offspring();
    while(flip(3/5)) {
      offspring();
    }
  }
}

// main block
{ offspring(); }
```

(d) geom-offspring

```
void gen_operator() {
  uniform(4);
}

void gen_expression() {
  prob {
    4/10: uniform(10);
    3/10: { }
    3/10: {
      gen_operator();
      gen_expression();
      gen_expression();
    }
  }
}

void gen_function() {
  gen_operator();
  gen_expression();
  gen_expression();
}

// main block
{ gen_function(); }
```

(e) gen-fun

```
void young() {
  int y = uniform(4);
  while(y > 0) {
    young();
    y = y-1; }
  int e = uniform(3);
  while(e > 0) {
    elder();
    e = e-1; }
}

void elder() {
  int y = uniform(2);
  while(y > 0) {
    young();
    y = y-1; }
  int e = uniform(5);
  while(e > 0) {
    elder();
    e = e-1; }
}

// main block
{ young(); }
```

(f) virus

```
bool f() {
  prob {
  1/2:
    return flip(1/2);
  1/2:
    if(f())  {
      return f();
    } else {
      return false;
    }
  }
}

// main block
{
    bool res1 = f();
    bool res2 = f();
    ...
    bool resN = f();
}
```

(g) sequential$N$

```
int f(int n, int m) {
  prob {
    (n+1)/(n+2) : {
      f((n + 1) % m,
    m);
      f((n + 1) % m,
    m);
      return 0;
    }
    1/(n+2) : return 0;
  }
}

// main block
{ f(0, N); }
```

(h) escape$N$

```
void f(int n) {
  while(n > 0) {
    prob {
      2/3: f(n-1);
      1/3: f((n+1) % N
    );
    }
    n = n-1;
  }
}

// main block
{ f(1); }
```

(i) mod$N$

Figure A.2: The nine probabilistic recursive programs whose pPDA models we used as benchmarks in Chapter 5 (see Table 5.1 on page 144). We adapted some of these programs from the literature; see Section 5.4.2.1 on page 143 for details. Note that several programs are parameterized: rw-p is parameterized by the probability p used in the `flip` statement; sequential$N$, escape$N$, and mod$N$ are parameterized by a positive integer N, which influences the size of the resulting pPDA.

A.2 Comparison of z3 and cvc5

Table A.1 compares the performance of z3 and cvc5 as backends for the SMT-baseline approach.[1]

1: See Section 5.4.1.2 on page 142.

Overall, z3 outperforms cvc5 on our benchmarks, solving more than twice as many instances (z3: 21/28; cvc5: 10/28) within the time limit. Notably, however, z3 and cvc5 perform comparably on the second benchmark set (grammars), where cvc5 is approximately three times faster than z3 on 2 of the 3 instances (out of 8 total) that both solvers can handle within the time limit.

Benchmark	**Time**$_{z3}$ (ms)	**Time**$_{cvc5}$ (ms)
rw-0.499	**11**	92
rw-0.500	**10**	87
rw-0.501	**12**	104
geom-offspring	**16**	4,687
golden	**14**	1,097
and-or	**15,260**	TO
gen-fun	**141**	TO
virus	**139**	163,727
escape10	**48**	12,031
escape25	**15,958**	TO
escape50	**410**	TO
escape75	TO	TO
escape100	**939**	TO
escape200	TO	TO
sequential5	**1,074**	TO
sequential7	**12,822**	TO
sequential10	**453,718**	TO
mod5	**34,150**	TO
mod7	**443**	TO
mod10	**1,245**	TO
brown	TO	TO
lemonde	TO	TO
negra	37,248	**10,144**
swbd	TO	Error
tiger	17,454	**16,118**
tuebadz	15,323	**5,534**
wsj	TO	TO
random	TO	TO

Table A.1: Comparison of z3 and cvc5 as backends for the SMT-baseline approach (Section 5.4.1.2 on page 142). The table includes both benchmark sets described Section 5.4.2.1: programs (top) and grammars (bottom). See Tables 5.1 and 5.2 on page 144 and on page 145 for size of the respective PPSs. The timeout (TO) was set to 10 minutes.

Omitted Proofs B

This appendix contains various (technical) proofs that were omitted from the main text.

B.1 Proof of Lemma 3.37 . . . 223
B.2 Proof of Lemma 6.22 . . . 224
B.3 Proof of Lemma 6.28 . . . 226

B.1 Proof of Lemma 3.37

Lemma 3.37 *Let $A \in \mathbb{R}_{\geq 0}^{n \times n}$ be a matrix and let $\mathbf{b}, \mathbf{c} \in \mathbb{R}_{\geq 0}^{n}$ be vectors. Suppose that $\boldsymbol{x}$ and $\mathbf{y}$ are the least solutions to the linear equation systems*

$$\boldsymbol{x} \;=\; A\boldsymbol{x} + \mathbf{b} \quad \textit{and} \quad \boldsymbol{y} \;=\; A\boldsymbol{y} + \mathbf{c}\,,$$

respectively, over the extended *nonnegative reals. Furthermore, assume that $\boldsymbol{x}$ and $\mathbf{y}$ satisfy $\boldsymbol{x}_i = 0$ iff $\mathbf{y}_i = 0$ for all $i \in [1..n]$. Then it holds that*

$$\boldsymbol{x} < \infty \quad \Longleftrightarrow \quad \mathbf{y} < \infty\,;$$

that is, either both systems have a nonnegative finite *solution, or neither does.*

Proof. The proof crucially relies on Lemma 4.31 on page 114, which states that every linear PPS that is both clean and feasible has a convergent matrix (i.e., is nondegenerate).

Let $\boldsymbol{x}' = A'\boldsymbol{x}' + \mathbf{b}'$ and $\boldsymbol{y}' = A'\boldsymbol{y}' + \mathbf{c}'$ denote the cleaned-up variants of the two linear PPSs. Note that, by the assumption that $\boldsymbol{x}_i = 0$ iff $\boldsymbol{y}_i = 0$, the two cleaned-up systems indeed have the *same* matrix A'.

We can now argue as follows:

$\boldsymbol{x} = A\boldsymbol{x} + \mathbf{b}$ is feasible (i.e., has a finite nonneg. solution)
$\Longrightarrow$ The cleaned-up variant $\boldsymbol{x}' = A'\boldsymbol{x}' + \mathbf{b}'$ is feasible
$\Longrightarrow$ A' is convergent (Lemma 4.31)
$\Longrightarrow$ $\boldsymbol{y}' = A'\boldsymbol{y}' + \mathbf{c}'$ has a unique finite nonnegative solution (Lemma 4.31)
$\Longrightarrow$ $\boldsymbol{y} = A\boldsymbol{y} + \mathbf{c}$ is feasible

The reverse direction follows by symmetry. □

B.2 Proof of Lemma 6.22

Lemma 6.22 (Soundness of $\Delta \times \mathcal{D}$) *Let Δ be a pVPA and $\mathcal{D}$ be a stair parity DVPA with priority function Ω, both over the same pushdown alphabet Σ. Then the product pVPA $\Delta \times \mathcal{D}$ with priority function Ω' as in Definition 6.21 satisfies*

$$\begin{aligned} &\mathbb{P}\big(\{\rho \in \mathrm{Runs}_\Delta \mid \lambda(\rho) \in \mathcal{L}(\mathcal{D})\}\big) \\ = \ &\mathbb{P}\big(\{\rho \in \mathrm{Runs}_{\Delta\times\mathcal{D}} \mid \rho{\downarrow_{\mathrm{Steps}}} \in \mathrm{Parity}_{\Omega'}\}\big)\ , \end{aligned}$$

where $\mathrm{Parity}_{\Omega'}$ *denotes the set of words from* $(Q \times S)^\omega$ *that satisfy the* standard *parity condition induced by* Ω'*. Moreover, we can construct* $\Delta \times \mathcal{D}$ *in polynomial time.*

Proof. Let

$$\Delta \ = \ (Q, q_0, \Gamma, \bot, P, \Sigma, \lambda) \quad \text{and} \quad \mathcal{D} \ = \ (\Sigma, S, s_0, \Gamma', \bot, \delta)$$

be the pVPA and the DVPA, respectively. In this proof, let $\tilde{\Delta} = \Delta \times \mathcal{D}$ denote their product.[1]

1: See Definition 6.21 on page 167.

We first show that $\tilde{\Delta}$ is indeed an (unlabeled) pVPA. To this end, note that

$$(Q_{\mathrm{call}} \times S) \uplus (Q_{\mathrm{int}} \times S) \uplus (Q_{\mathrm{ret}} \times S)$$

is a partition of the product states that satisfies the visibility condition, that is, call states only have push transitions, internal states only have internal transitions, and return states only have pop transitions. This follows immediately from the definition of the transition function $P_{\tilde{\Delta}}$.

To verify that $P_{\tilde{\Delta}}$ is a well-defined *probabilistic* transition function, observe that for every fixed $(q, s) \in Q_{\mathrm{int}} \times S$,

$$\sum_{(r,t) \in Q_{\mathrm{int}} \times S} P_{\tilde{\Delta},\mathrm{int}}((p,s),(r,t)) \quad = \quad 1$$

because for each internal transition $q \xrightarrow{p}_\Delta r$ in Δ and state $s \in S$, there exists *exactly one* $t \in S$ such that $s \xrightarrow{\lambda(q)}_{\mathcal{D}} t$ is an internal transition in $\mathcal{D}$. This follows from the fact that $\mathcal{D}$ is *deterministic* and from the visibility condition on the labeling function λ (internal states must be labeled with symbols from Σ_{int}). Indeed, if $\mathcal{D}$ were nondeterministic, the product would in general not be a proper *probabilistic* automaton (i.e., there could be states whose outgoing

transitions have probability mass strictly greater or less than one). The argument for call and return transitions is analogous.

It remains to show the equation

$$\begin{aligned} &\mathbb{P}(\{\rho \in \mathrm{Runs}_\Delta \mid \lambda(\rho) \in \mathcal{L}(\mathcal{D})\}) \\ = \ &\mathbb{P}(\{\rho \in \mathrm{Runs}_{\tilde{\Delta}} \mid \rho{\downarrow_{\mathrm{Steps}}} \in \mathrm{Parity}_{\Omega'}\}) \end{aligned}$$

from Lemma 6.22. To this end, we define a function

$$\phi \colon \mathrm{Runs}_\Delta \to \mathrm{Runs}_{\tilde{\Delta}}$$

as follows:

$$\phi(q_0\gamma_0, q_1\gamma_1, \ldots) \ = \ (q_0, s_0)\langle\gamma_0\gamma_0'\rangle \, , \, (q_1, s_1)\langle\gamma_1\gamma_1'\rangle \, , \, \ldots \, ,$$

where $\gamma_0 = \gamma_0' = \bot$. Moreover, for all $i \geq 0$:

- If $q_i \in Q_{\mathrm{call}}$, then s_{i+1} is defined by $s_i \xrightarrow{\lambda(q_i)}_{\mathcal{D}} s_{i+1}Y$ and $\gamma'_{i+1} = \gamma'_i Y$.
- If $q_i \in Q_{\mathrm{int}}$, then s_{i+1} is defined by $s_i \xrightarrow{\lambda(q_i)}_{\mathcal{D}} s_{i+1}$ and $\gamma'_{i+1} = \gamma'_i$.
- If $q_i \in Q_{\mathrm{ret}}$, then s_{i+1} is defined by $s_i Y \xrightarrow{\lambda(q_i)}_{\mathcal{D}} s_{i+1}$ and $\gamma'_{i+1} Y = \gamma'_i$.

ϕ thus maps every of run ρ of Δ to its corresponding run in the product $\tilde{\Delta}$ by feeding the labeling sequence $\lambda(\rho)$ as input into the *deterministic* VPA $\mathcal{D}$. Thus, ϕ is a bijection.

Moreover, ϕ preserves probabilities, that is, $\mathbb{P}(R) = \mathbb{P}(\phi(R))$ for all measurable sets $R \subseteq \mathrm{Runs}_\Delta$. This follows because all transition probabilities occurring along a run $\rho \in \mathrm{Runs}_\Delta$ are exactly the same as those occurring along $\phi(\rho)$. The set $\{\rho \in \mathrm{Runs}_\Delta \mid \lambda(\rho) \in \mathcal{L}(\mathcal{D})\}$ is measurable [90], and we have

$$\phi(\{\rho \in \mathrm{Runs}_\Delta \mid \lambda(\rho) \in \mathcal{L}(\mathcal{D})\}) \ = \ \{\rho \in \mathrm{Runs}_{\tilde{\Delta}} \mid \rho{\downarrow_{\mathrm{Steps}}} \in \mathrm{Parity}_{\Omega'}\}$$

because of the following: $\lambda(\rho) \in \mathcal{L}(\mathcal{D})$ means that the run ρ' of $\mathcal{D}$ on $\lambda(\rho)$ satisfies the stair parity condition Ω on $\mathcal{D}$, that is, its footprint $\rho'{\downarrow_{\mathrm{Steps}}}$ satisfies Ω as a standard parity condition Parity_Ω. However, the step positions of ρ' (and ρ) coincide with those of the corresponding run $\phi(\rho)$ in the product, and the modified priority function Ω' considers only the DVPA states. Thus, $\phi(\rho){\downarrow_{\mathrm{Steps}}}$ satisfies the standard parity condition Ω', which implies the claim. □

Table B.1: (Table 6.3 restated) Next-step probabilities of the step Markov chain.

	$\mathbf{q} \to \mathbf{r}$	$\mathbf{q}\bot \to \mathbf{r}\bot$	$\mathbf{q}\bot \to \mathbf{r}$	$\mathbf{q} \to \mathbf{r}\bot$
$q \in Q_{\text{call}}$	$\frac{[r\uparrow]}{[q\uparrow]}\left(\sum_{r',Z} P_{\text{call}}(q,r'Z)[r'Z\,r] + \sum_{Z} P_{\text{call}}(q,rZ)\right)$	$\sum_{r',Z} P_{\text{call}}(q,r'Z)[r'Z\,r]$	$\sum_{Z} P_{\text{call}}(q,rZ)[r\uparrow]$	0
$q \in Q_{\text{int}}$	$\frac{[r\uparrow]}{[q\uparrow]} P_{\text{int}}(q,r)$	$P_{\text{int}}(q,r)$	0	0
$q \in Q_{\text{ret}}$	undef.	$P_{\text{ret}}(q\bot,r)$	0	undef.

B.3 Proof of Lemma 6.28

In this section, I provide a more formal proof of Lemma 6.28 on page 173, which is restated here for convenience (Table 6.3 is also restated as Table B.1):

Lemma 6.28 *The conditional next-step probabilities in Table 6.3 are correct in the sense that whenever $\mathbb{P}(V^{(n+1)} = v' \mid V^{(n)} = v)$ is well-defined for some $n \in \mathbb{N}_0$ and $v, v' \in Q \cup Q\bot$, then it is equal to the probability in the respective column $v \to v'$.*

Proof. In the proof sketch on page 173, we have already justified several entries of Table 6.3.

We now give the formal proofs for Equations (6.3) and (6.4), which we restate here for convenience as the following Equations (B.1) and (B.2). Let $r \in Q$ and $n \in \mathbb{N}$. Then, for all $q \in Q_{\text{int}}$ we have to show that

$$\mathbb{P}(V^{(n+1)} = r \mid V^{(n)} = q) \;=\; \frac{[r\uparrow]}{[q\uparrow]} \cdot P_{\text{int}}(q,r)\,. \tag{B.1}$$

Similarly, for all $q \in Q_{\text{call}}$ we will show that

$$\begin{aligned}&\mathbb{P}(V^{(n+1)} = r \mid V^{(n)} = q)\\ &\quad= \frac{[r\uparrow]}{[q\uparrow]} \cdot \left(\sum_{r',Z} P_{\text{call}}(q,r'Z)[r'Z\,r] + \sum_{Z} P_{\text{call}}(q,rZ)\right).\end{aligned} \tag{B.2}$$

In the following, we frequently use equations involving conditional probabilities (e.g., $\mathbb{P}(A \mid B) = \mathbb{P}(C \mid D)$). Recall that conditional probabilities are ill-defined if the conditioning event has probability zero. The meaning our equations is that they hold whenever all probabilities involved are well-defined. We shall also make frequent use of the basic laws of conditional probabilities, in particular $\mathbb{P}(A \mid B) \cdot \mathbb{P}(B) = \mathbb{P}(A \wedge B)$.

We begin by introducing some notation and proving some auxiliary lemmas.

Definition B.1 ***(Selected Events Along Runs)*** Let $i \in \mathbb{N}_0$ and $q \in Q$. We define the following events:

- $q@i$ is the set of all runs $\rho \in \mathrm{Runs}_\Delta$ such that $\rho(i) = q\gamma$ with $\gamma \neq \bot$, i.e., the runs whose ith configuration has state q and stack unequal to $\bot$.
- Similarly, $q\bot@i$ denotes the set of all runs $\rho \in \mathrm{Runs}_\Delta$ such that $\rho(i) = q\bot$, i.e., the runs whose ith configuration is a *bottom* configuration with state q.
- step@i denotes the set of all runs such that the ith configuration is a step.
- We define $\mathrm{sh}@i = \mathrm{height}(\rho(i)) \in \mathbb{N}_0$, that is, the stack height of the ith configuration. Strictly speaking, sh@i is a random variable, not an event. Note that step@i is by definition equivalent to $\forall j > i \colon \mathrm{sh}@j \geq \mathrm{sh}@i$.

These events are all measurable. △

Lemma B.2 *For all $i \in \mathbb{N}_0$, and $q \in Q$, the following identities hold:*

$$\mathbb{P}(\mathrm{step}@i \mid q@i) \;=\; [q\uparrow] \tag{B.3}$$

$$\mathbb{P}(\mathrm{step}@i \mid q\bot@i) \;=\; 1 \tag{B.4}$$

Furthermore, for $q \in Q_{\mathrm{int}}$ and $r \in Q$:

$$\mathbb{P}(r@(i{+}1) \mid q@i) \;=\; P_{\mathrm{int}}(q, r) \tag{B.5}$$

$$\begin{aligned}&\mathbb{P}(\mathrm{step}@(i{+}1) \wedge \mathrm{step}@i \mid r@(i{+}1) \wedge q@i)\\ &\quad= \quad \mathbb{P}(\mathrm{step}@(i{+}1) \mid r@(i{+}1))\end{aligned} \tag{B.6}$$

Proof. The first three equations follow immediately from the definitions. For Equation (B.6) we have:

$$\begin{aligned}&\mathbb{P}(\mathrm{step}@(i{+}1) \wedge \mathrm{step}@i \mid r@(i{+}1) \wedge q@i)\\ =\;&\mathbb{P}(\mathrm{step}@(i{+}1) \mid r@(i{+}1) \wedge q@i)\\ =\;&\mathbb{P}(\mathrm{step}@(i{+}1) \mid r@(i{+}1))\end{aligned}$$

The first equation holds because if $q \in Q_{\mathrm{int}}$ and $q@i$, then step@$(i{+}1) \wedge$ step@i is already implied by step@$(i{+}1)$, and the second equation holds because the probability that step@$(i{+}1)$ depends only on the state at position $i{+}1$, not on the state at position i. □

We now prove Equation (B.1). By time homogeneity[2] and the def-

2: See Equation (6.2) on page 170.

inition of $V^{(n)}$, we have for all $i, n \in \mathbb{N}_0$, $q \in Q_{\text{int}}$, and $r \in Q$ that

$$\begin{aligned} &\mathbb{P}(V^{(n+1)} = r \mid V^{(n)} = q) \\ =\ &\mathbb{P}(r@(i{+}1) \wedge \text{step}@(i{+}1) \mid q@i \wedge \text{step}@i)\ . \end{aligned} \tag{B.7}$$

We now argue as follows:

$$\begin{aligned} &\mathbb{P}(V^{(n+1)} = r \mid V^{(n)} = q) \\ =\ &\mathbb{P}(r@(i{+}1) \wedge \text{step}@(i{+}1) \mid q@i \wedge \text{step}@i) \quad \text{(Equation (B.7))} \\ =\ &\frac{\mathbb{P}(r@(i{+}1) \wedge \text{step}@(i{+}1) \wedge q@i \wedge \text{step}@i)}{\mathbb{P}(q@i \wedge \text{step}@i)} \\ &\qquad \text{(Law of conditional probabilities)} \\ =\ &\frac{\mathbb{P}(r@(i{+}1) \wedge \text{step}@(i{+}1) \wedge \text{step}@i \mid q@i) \cdot \mathbb{P}(q@i)}{\mathbb{P}(\text{step}@i \mid q@i) \cdot \mathbb{P}(q@i)} \\ &\qquad \text{(Law of conditional probabilities)} \\ =\ &\frac{\mathbb{P}(r@(i{+}1) \wedge \text{step}@(i{+}1) \wedge \text{step}@i \mid q@i)}{\mathbb{P}(\text{step}@i \mid q@i)} \quad \text{(Simplification)} \\ =\ &\frac{\mathbb{P}(\text{step}@(i{+}1) \wedge \text{step}@i \mid r@(i{+}1) \wedge q@i) \cdot \mathbb{P}(r@(i{+}1) \mid q@i)}{\mathbb{P}(\text{step}@i \mid q@i)} \\ &\qquad \text{(Law of conditional probabilities)} \\ =\ &\frac{\mathbb{P}(\text{step}@(i{+}1) \mid r@(i{+}1)) \cdot \mathbb{P}(r@(i{+}1) \mid q@i)}{\mathbb{P}(\text{step}@i \mid q@i)} \\ &\qquad \text{(Equation (B.6))} \\ =\ &\frac{[r\uparrow] \cdot P_{\text{int}}(q, r)}{[q\uparrow]}\ . \qquad \text{(Equations (B.3) and (B.5))} \end{aligned}$$

This concludes the proof of Equation (B.1).

We now consider Equation (B.2). For all $i, n \in \mathbb{N}_0$, $q \in Q_{\text{call}}$ and $r \in Q$, it holds that

$$\begin{aligned} &\mathbb{P}(V^{(n+1)} = r \mid V^{(n)} = q) \\ =\ &\mathbb{P}(\exists k{>}i : \text{step}@k \wedge r@k \wedge \forall i{<}j{<}k : \neg\text{step}@j \mid \text{step}@i \wedge q@i) \\ &\qquad \text{(by time homogeneity and definition of } V^{(n)}) \\ =\ &\mathbb{P}(\exists k{>}i{+}1 : \text{step}@k \wedge r@k \wedge \forall i{<}j{<}k : \neg\text{step}@j \mid \text{step}@i \wedge q@i) \\ &\quad +\ \mathbb{P}(r@(i{+}1) \wedge \text{step}@(i{+}1) \mid q@i \wedge \text{step}@i) \end{aligned} \tag{B.8}$$

The last equality results from splitting into two disjoint events. For

the second summand in Equation (B.8), it can be shown that

$$\mathbb{P}(r@(i{+}1) \wedge \text{step}@(i{+}1) \mid q@i \wedge \text{step}@i) \;=\; \frac{[r\uparrow]}{[q\uparrow]} \sum_{Z} P_{\text{call}}(q, rZ)$$

by an argument similar to the one used for Equation (B.1). Note that the sum over all stack symbols Z arises because $q \in Q_{\text{call}}$, so there may be multiple—up to $|\Gamma_{\text{-}\perp}|$—direct transitions from q to r.

To derive an expression for the first summand in Equation (B.8), we need several auxiliary equations. Those are summarized in the next lemma:

Lemma B.3 *For all $q \in Q_{\text{call}}$, $r \in Q$, and $i \in \mathbb{N}_0$ it holds that:*

$$\begin{aligned} &\sum_{r',Z} P_{\text{call}}(q, r'Z)[r'Z\, r] \\ &= \sum_{k>i+1} \mathbb{P}(r@k \wedge \forall i{<}j{<}k : \text{sh}@j > \text{sh}@k = \text{sh}@i \mid q@i)\,. \end{aligned} \tag{B.9}$$

Moreoever, for all $q \in Q_{\text{call}}$, $i \in \mathbb{N}_0$, and $k \in \mathbb{N}_0$, we have:

$$\begin{aligned} &\mathbb{P}(\text{step}@k \wedge \forall i{<}j{<}k : \neg\text{step}@j \wedge \text{step}@i \mid q@i) \\ &= \mathbb{P}(\text{step}@k \wedge \forall i{<}j{<}k : \text{sh}@j > \text{sh}@i = \text{sh}@k \mid q@i) \end{aligned} \tag{B.10}$$

and

$$\begin{aligned} &\mathbb{P}(\forall i{<}j{<}k : \text{sh}@j > \text{sh}@i = \text{sh}@k \mid q@i \wedge r@k \wedge \text{step}@k) \\ &= \mathbb{P}(\forall i{<}j{<}k : \text{sh}@j > \text{sh}@i = \text{sh}@k \mid q@i \wedge r@k)\,. \end{aligned} \tag{B.11}$$

Proof. For Equation (B.9), note that $\sum_{r',Z} P_{\text{call}}(q, r'Z)[r'Z\, r]$ is the probability to go from q (with empty stack) to a successor state r', push $Z \in \Gamma$, and then later reach r with empty stack within finitely many steps. Assuming $q@i$, this is equivalent to summing over all positions $k > i + 1$ (we can exclude $k = i + 1$ because pushing and popping Z within a single transition is impossible) such that $r@k$ and for all $i < j < k$ the stack height is greater than at positions i and k. Positions i and k have the same stack height because the symbol Z is pushed in the transition from i, and k is the position directly after Z is popped. In between these two transitions, the stack below Z cannot change, so the stack height is the same at both positions.

For Equation (B.10), we argue as follows:

$$\begin{aligned}
&\mathbb{P}(\text{step@}k \wedge \forall i{<}j{<}k : \neg\text{step@}j \wedge \text{step@}i \mid q\text{@}i) \\
=\ &\mathbb{P}(\text{step@}k \wedge \forall i{<}j{<}k : \text{sh@}j > \text{sh@}i = \text{sh@}k \wedge \text{step@}i \mid q\text{@}i) \\
=\ &\mathbb{P}(\text{step@}k \wedge \forall i{<}j{<}k : \text{sh@}j{>}\text{sh@}i = \text{sh@}k \mid q\text{@}i)
\end{aligned}$$

The first equality holds because if no position between i and k is a step, then the stack at those positions must be higher than at position k. Furthermore, since i is a step, we have $\text{sh@}i \leq \text{sh@}k$; moreover, since $i{+}1$ is not a step and q is a call state, we have $\text{sh@}i = \text{sh@}k$. The second equality holds because if i and k have the same stack height and all positions between them have a higher stack, then i is a step if and only if k is a step.

For Equation (B.11), the result may seem counter-intuitive because conditioning on step@k is corresponds to *conditioning on the future*: The stack height *after* position k should never be smaller than at position k. Knowing that step@k provides information about the (extended) state at position k. However, in Equation (B.11) we also condition on r@k, i.e., at position k the run is in state r and the topmost stack symbol is not $\bot$. Hence, in the context of Equation (B.11), we can drop step@k from the condition. □

We conclude the proof of Equation (B.2)—and thus of the whole Lemma 6.28—by deriving the desired equation for the first summand in Equation (B.8):

$$\begin{aligned}
&\mathbb{P}(\exists k{>}i{+}1 : \text{step@}k \wedge r\text{@}k \wedge \forall i{<}j{<}k : \neg\text{step@}j \mid \text{step@}i \wedge q\text{@}i) \\
=\ &\frac{\mathbb{P}(\exists k{>}i{+}1 : \text{step@}k \wedge r\text{@}k \wedge \forall i{<}j{<}k : \neg\text{step@}j{\wedge}\text{step@}i \mid q\text{@}i) \cdot \mathbb{P}(q\text{@}i)}{\mathbb{P}(\text{step@}i \mid q\text{@}i) \cdot \mathbb{P}(q\text{@}i)} \\
&\qquad\qquad \text{(Law of conditional probabilities, twice)} \\
=\ &\frac{\mathbb{P}(\exists k{>}i{+}1 : \text{step@}k \wedge r\text{@}k \wedge \forall i{<}j{<}k : \neg\text{step@}j \wedge \text{step@}i \mid q\text{@}i)}{\mathbb{P}(\text{step@}i \mid q\text{@}i)} \\
&\qquad\qquad \text{(Simplification)} \\
=\ &\sum_{k>i+1} \frac{\mathbb{P}(\text{step@}k \wedge r\text{@}k \wedge \forall i{<}j{<}k : \neg\text{step@}j \wedge \text{step@}i \mid q\text{@}i)}{\mathbb{P}(\text{step@}i \mid q\text{@}i)} \\
&\qquad\qquad \text{(Split disjoint events)} \\
=\ &\sum_{k>i+1} \frac{\mathbb{P}(\text{step@}k \wedge r\text{@}k \wedge \forall i{<}j{<}k : \text{sh@}j > \text{sh@}i = \text{sh@}k \mid q\text{@}i)}{\mathbb{P}(\text{step@}i \mid q\text{@}i)} \\
&\qquad\qquad \text{(By Equation (B.10))}
\end{aligned}$$

$$= \sum_{k>i+1} \Bigg(\mathbb{P}(\text{step@}k \wedge r@k \mid q@i) \cdot \frac{\mathbb{P}(\forall i<j<k : \text{sh@}j > \text{sh@}i = \text{sh@}k \mid q@i \wedge \text{step@}k \wedge r@k)}{\mathbb{P}(\text{step@}i \mid q@i)} \Bigg)$$

(Law of conditional probabilities)

$$= \sum_{k>i+1} \Bigg(\mathbb{P}(\text{step@}k \wedge r@k \mid q@i) \cdot \frac{\mathbb{P}(\forall i<j<k : \text{sh@}j > \text{sh@}i = \text{sh@}k \mid q@i \wedge r@k)}{\mathbb{P}(\text{step@}i \mid q@i)} \Bigg)$$

(By Equation (B.11))

$$= \sum_{k>i+1} \Bigg(\frac{\mathbb{P}(\text{step@}k \mid q@i \wedge r@k)}{\mathbb{P}(\text{step@}i \mid q@i)} \cdot \mathbb{P}(\forall i<j<k : \text{sh@}j > \text{sh@}i = \text{sh@}k \wedge r@k \mid q@i) \Bigg)$$

(Rewriting)

$$= \frac{[r\uparrow]}{[q\uparrow]} \sum_{k>i+1} \mathbb{P}(r@k \wedge \forall i<j<k : \text{sh@}j > \text{sh@}i = \text{sh@}k \mid q@i)$$

(Equation (B.3) and using that $\mathbb{P}(\text{step@}k \mid q@i \wedge r@k) = \mathbb{P}(\text{step@}k \mid r@k)$)

$$= \frac{[r\uparrow]}{[q\uparrow]} \sum_{r',Z} P_{\text{call}}(q, r'\,Z)[r'\,Z\,r].$$

(By Equation (B.9))

□

List of Co-Authored Publications

C

The papers covered in this thesis are highlighted in boldface. The order is chronological.

1. Tobias Winkler, Sebastian Junges, Guillermo A. Pérez, and Joost-Pieter Katoen. "On the Complexity of Reachability in Parametric Markov Decision Processes". In: *CONCUR*. Vol. 140. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019

2. Pranav Ashok, Krishnendu Chatterjee, Jan Kretínský, Maximilian Weininger, and Tobias Winkler. "Approximating Values of Generalized-Reachability Stochastic Games". In: *LICS*. ACM, 2020

3. Krishnendu Chatterjee, Joost-Pieter Katoen, Maximilian Weininger, and Tobias Winkler. "Stochastic Games with Lexicographic Reachability-Safety Objectives". In: *CAV (2)*. Vol. 12225. Lecture Notes in Computer Science. Springer, 2020

4. Lutz Klinkenberg, Kevin Batz, Benjamin Lucien Kaminski, Joost-Pieter Katoen, Joshua Moerman, and Tobias Winkler. "Generating Functions for Probabilistic Programs". In: *LOPSTR*. Vol. 12561. Lecture Notes in Computer Science. Springer, 2020

5. Tobias Winkler and Maximilian Weininger. "Stochastic Games with Disjunctions of Multiple Objectives". In: *GandALF*. Vol. 346. EPTCS. 2021

6. Sebastian Junges, Joost-Pieter Katoen, Guillermo A. Pérez, and Tobias Winkler. "The complexity of reachability in parametric Markov decision processes". In: *J. Comput. Syst. Sci.* 119 (2021)

7. Tobias Winkler, Johannes Lehmann, and Joost-Pieter Katoen. "Out of Control: Reducing Probabilistic Models by Control-State Elimination". In: *VMCAI*. Vol. 13182. Lecture Notes in Computer Science. Springer, 2022

8. **Tobias Winkler, Christina Gehnen, and Joost-Pieter Katoen. "Model Checking Temporal Properties of Recursive Probabilistic Programs". In: *FoSSaCS*. Vol. 13242. Lecture Notes in Computer Science. Springer, 2022**

9. Mingshuai Chen, Joost-Pieter Katoen, Lutz Klinkenberg, and Tobias Winkler. "Does a Program Yield the Right Distribution? - Verifying Probabilistic Programs via Generating Functions". In:

CAV (1). Vol. 13371. Lecture Notes in Computer Science. Springer, 2022

10. Kevin Batz, Adrian Gallus, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Tobias Winkler. "Weighted programming: a programming paradigm for specifying mathematical models". In: *Proc. ACM Program. Lang.* 6.OOPSLA1 (2022)

11. **Tobias Winkler and Joost-Pieter Katoen. "Certificates for Probabilistic Pushdown Automata via Optimistic Value Iteration". In: *TACAS (2)*. Vol. 13994. Lecture Notes in Computer Science. Springer, 2023**

12. **Tobias Winkler and Joost-Pieter Katoen. "On Certificates, Expected Runtimes, and Termination in Probabilistic Pushdown Automata". In: *LICS*. IEEE, 2023**

13. **Tobias Winkler, Christina Gehnen, and Joost-Pieter Katoen. "Model Checking Temporal Properties of Recursive Probabilistic Programs". In: *Log. Methods Comput. Sci.* 19.4 (2023)**

14. Krishnendu Chatterjee, Joost-Pieter Katoen, Stefanie Mohr, Maximilian Weininger, and Tobias Winkler. "Stochastic games with lexicographic objectives". In: *Formal Methods Syst. Des.* 63.1 (2024)

15. Kevin Batz, Tom Jannik Biskup, Joost-Pieter Katoen, and Tobias Winkler. "Programmatic Strategy Synthesis: Resolving Nondeterminism in Probabilistic Programs". In: *Proc. ACM Program. Lang.* 8.POPL (2024)

16. Hannah Mertens, Joost-Pieter Katoen, Tim Quatmann, and Tobias Winkler. "Accurately Computing Expected Visiting Times and Stationary Distributions in Markov Chains". In: *TACAS (2)*. Vol. 14571. Lecture Notes in Computer Science. Springer, 2024

17. Raphaël Berthon, Joost-Pieter Katoen, and Tobias Winkler. "Markov Decision Processes with Sure Parity and Multiple Reachability Objectives". In: *RP*. Vol. 15050. Lecture Notes in Computer Science. Springer, 2024

18. Kevin Batz, Benjamin Lucien Kaminski, Christoph Matheja, and Tobias Winkler. "J-P: MDP. FP. PP - Characterizing Total Expected Rewards in Markov Decision Processes as Least Fixed Points with an Application to Operational Semantics of Probabilistic Programs". In: *Principles of Verification (1)*. Vol. 15260. Lecture Notes in Computer Science. Springer, 2024

19. Krishnendu Chatterjee, Tim Quatmann, Maximilian Schäffeler, Maximilian Weininger, Tobias Winkler, and Daniel Zilken. "Fixed Point Certificates for Reachability and Expected Rewards in

MDPs". In: *TACAS (2)*. Vol. 15697. Lecture Notes in Computer Science. Springer, 2025

20. Kevin Batz, Joost-Pieter Katoen, Francesca Randone, and Tobias Winkler. "Foundations for Deductive Verification of Continuous Probabilistic Programs: From Lebesgue to Riemann and Back". In: *Proc. ACM Program. Lang.* 9.OOPSLA1 (2025)

21. Lina Gerlach, Tobias Winkler, Erika Ábrahám, Borzoo Bonakdarpour, and Sebastian Junges. "Efficient Probabilistic Model Checking for Relational Reachability". In: *CAV (1)*. Vol. 15931. Lecture Notes in Computer Science. Springer, 2025

22. Hannah Mertens, Joost-Pieter Katoen, Tim Quatmann, and Tobias Winkler. "Computing Expected Visiting Times and Stationary Distributions in Markov Chains: Fast and Accurate". In: *J. Autom. Reason.* 69.3 (2025)

23. Dominik Geißler and Tobias Winkler. "Weighted Automata for Exact Inference in Discrete Probabilistic Programs". In: *ICTAC*. Vol. 16237. Lecture Notes in Computer Science. Springer, 2025

24. Darion Haase, Kevin Batz, Adrian Gallus, Benjamin Lucien Kaminski, Joost-Pieter Katoen, Lutz Klinkenberg, and Tobias Winkler. "Generating Functions Meet Occupation Measures: Invariant Synthesis for Probabilistic Loops". In: *ESOP (1)*. Vol. 16501. Lecture Notes in Computer Science. Springer, 2026

25. Daniel Zilken, Kevin Batz, Joost-Pieter Katoen, and Tobias Winkler. "Verifying Sampling Algorithms via Distributional Invariants". In: *FM (1)*. Vol. 16556. Lecture Notes in Computer Science. Springer, 2026

Bibliography

[1] Rajeev Alur, Ahmed Bouajjani, and Javier Esparza. "Model Checking Procedural Programs". In: *Handbook of Model Checking*. Ed. by Edmund M. Clarke et al. Springer, 2018, pp. 541–572. DOI: 10.1007/978-3-319-10575-8_17 (cited on pages 163, 183, 187).

[2] Rajeev Alur, Kousha Etessami, and P. Madhusudan. "A Temporal Logic of Nested Calls and Returns". In: *TACAS*. Vol. 2988. Lecture Notes in Computer Science. Springer, 2004, pp. 467–481. DOI: 10.1007/978-3-540-24730-2_35 (cited on pages 8, 150, 153, 161, 162).

[3] Rajeev Alur and P. Madhusudan. "Visibly pushdown languages". In: *STOC*. ACM, 2004, pp. 202–211. DOI: 10.1145/1007352.1007390 (cited on pages 6, 150, 151, 154–157, 160, 184, 185).

[4] Rajeev Alur et al. "First-Order and Temporal Logics for Nested Words". In: *LICS*. IEEE Computer Society, 2007, pp. 151–160. DOI: 10.1109/LICS.2007.19 (cited on pages 163, 184).

[5] Robert B Ash and Catherine A Doléans-Dade. *Probability and Measure Theory*. Academic press, 2000 (cited on page 176).

[6] Pranav Ashok et al. "Approximating Values of Generalized-Reachability Stochastic Games". In: *LICS*. ACM, 2020, pp. 102–115. DOI: 10.1145/3373718.3394761 (cited on page 233).

[7] Muqsit Azeem et al. "Optimistic and Topological Value Iteration for Simple Stochastic Games". In: *ATVA*. Vol. 13505. Lecture Notes in Computer Science. Springer, 2022, pp. 285–302. DOI: 10.1007/978-3-031-19992-9_18 (cited on page 147).

[8] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, 2008 (cited on pages 9, 35, 38, 151, 156, 161, 163, 165, 177, 181, 198).

[9] Christel Baier et al. "Model Checking Probabilistic Systems". In: *Handbook of Model Checking*. Ed. by Edmund M. Clarke et al. Springer, 2018, pp. 963–999. DOI: 10.1007/978-3-319-10575-8_28 (cited on page 9).

[10] Haniel Barbosa et al. "cvc5: A Versatile and Industrial-Strength SMT Solver". In: *TACAS (1)*. Vol. 13243. Lecture Notes in Computer Science. Springer, 2022, pp. 415–442. DOI: 10.1007/978-3-030-99524-9_24 (cited on page 142).

[11] Gilles Barthe, Joost-Pieter Katoen, and Alexandra Silva, eds. *Foundations of Probabilistic Programming*. Cambridge University Press, 2020. DOI: 10.1017/9781108770750 (cited on page 9).

[12] Kevin Batz. "Automated deductive verification of probabilistic programs". PhD thesis. RWTH Aachen University, 2024. DOI: 10.18154/RWTH-2025-00473 (cited on pages 9, 38, 83, 213).

[13] Kevin Batz et al. "Foundations for Deductive Verification of Continuous Probabilistic Programs: From Lebesgue to Riemann and Back". In: *Proc. ACM Program. Lang.* 9.OOPSLA1 (2025), pp. 421–448. DOI: 10.1145/3720429 (cited on page 235).

[14] Kevin Batz et al. “J-P: MDP. FP. PP - Characterizing Total Expected Rewards in Markov Decision Processes as Least Fixed Points with an Application to Operational Semantics of Probabilistic Programs”. In: *Principles of Verification (1)*. Vol. 15260. Lecture Notes in Computer Science. Springer, 2024, pp. 255–302. DOI: 10.1007/978-3-031-75783-9_11 (cited on pages 83, 213, 234).

[15] Kevin Batz et al. “Programmatic Strategy Synthesis: Resolving Nondeterminism in Probabilistic Programs”. In: *Proc. ACM Program. Lang.* 8.POPL (2024), pp. 2792–2820. DOI: 10.1145/3632935 (cited on page 234).

[16] Kevin Batz et al. “Quantitative separation logic: a logic for reasoning about probabilistic pointer programs”. In: *Proc. ACM Program. Lang.* 3.POPL (2019), 34:1–34:29. DOI: 10.1145/3290347 (cited on page 213).

[17] Kevin Batz et al. “Weighted programming: a programming paradigm for specifying mathematical models”. In: *Proc. ACM Program. Lang.* 6.OOPSLA1 (2022), pp. 1–30. DOI: 10.1145/3527310 (cited on pages 48, 212, 234).

[18] Raphaël Berthon, Joost-Pieter Katoen, and Tobias Winkler. “Markov Decision Processes with Sure Parity and Multiple Reachability Objectives”. In: *RP*. Vol. 15050. Lecture Notes in Computer Science. Springer, 2024, pp. 203–220. DOI: 10.1007/978-3-031-72621-7_14 (cited on page 234).

[19] Olivier Bournez and Florent Garnier. “Proving Positive Almost-Sure Termination”. In: *RTA*. Vol. 3467. Lecture Notes in Computer Science. Springer, 2005, pp. 323–337. DOI: 10.1007/11805618_27 (cited on page 39).

[20] Tomás Brázdil, Antonín Kucera, and Oldrich Strazovský. “On the Decidability of Temporal Properties of Probabilistic Pushdown Automata”. In: *STACS*. Vol. 3404. Lecture Notes in Computer Science. Springer, 2005, pp. 145–157. DOI: 10.1007/978-3-540-31856-9_12 (cited on pages 10, 184).

[21] Tomás Brázdil et al. “Analyzing probabilistic pushdown automata”. In: *Formal Methods Syst. Des.* 43.2 (2013), pp. 124–163. DOI: 10.1007/S10703-012-0166-0 (cited on pages 10, 50, 60, 84, 144, 178, 179, 183, 184, 187).

[22] Tomás Brázdil et al. “Runtime analysis of probabilistic programs with unbounded recursion”. In: *J. Comput. Syst. Sci.* 81.1 (2015), pp. 288–310. DOI: 10.1016/J.JCSS.2014.06.005 (cited on pages 1, 10, 84, 140, 143).

[23] John F. Canny. “Some Algebraic and Geometric Computations in PSPACE”. In: *STOC*. ACM, 1988, pp. 460–467. DOI: 10.1145/62212.62257 (cited on pages 152, 182).

[24] Bob Carpenter et al. “Stan: A probabilistic programming language”. In: *Journal of statistical software* 76 (2017), pp. 1–32. DOI: 10.18637/jss.v076.i01 (cited on page 9).

[25] Aleksandar Chakarov and Sriram Sankaranarayanan. “Probabilistic Program Analysis with Martingales”. In: *CAV*. Vol. 8044. Lecture Notes in Computer Science. Springer, 2013, pp. 511–526. DOI: 10.1007/978-3-642-39799-8_34 (cited on page 9).

[26] Krishnendu Chatterjee et al. “Fixed Point Certificates for Reachability and Expected Rewards in MDPs”. In: *TACAS (2)*. Vol. 15697. Lecture Notes in Computer Science. Springer, 2025, pp. 130–151. DOI: 10.1007/978-3-031-90653-4_7 (cited on pages 122, 147, 234).

[27] Krishnendu Chatterjee et al. “Stochastic games with lexicographic objectives”. In: *Formal Methods Syst. Des.* 63.1 (2024), pp. 40–80. DOI: 10.1007/S10703-023-00411-4 (cited on page 234).

[28] Krishnendu Chatterjee et al. “Stochastic Games with Lexicographic Reachability-Safety Objectives”. In: *CAV (2)*. Vol. 12225. Lecture Notes in Computer Science. Springer, 2020, pp. 398–420. DOI: 10.1007/978-3-030-53291-8_21 (cited on page 233).

[29] Mingshuai Chen et al. “Does a Program Yield the Right Distribution? - Verifying Probabilistic Programs via Generating Functions”. In: *CAV (1)*. Vol. 13371. Lecture Notes in Computer Science. Springer, 2022, pp. 79–101. DOI: 10.1007/978-3-031-13185-1_5 (cited on page 233).

[30] David Chiang, Colin McDonald, and Chung-chieh Shan. “Exact Recursive Probabilistic Programming”. In: *Proc. ACM Program. Lang.* 7.OOPSLA1 (2023), pp. 665–695. DOI: 10.1145/3586050 (cited on page 10).

[31] Michele Chiari et al. “A Model Checker for Operator Precedence Languages”. In: *ACM Trans. Program. Lang. Syst.* 45.3 (2023), 19:1–19:66. DOI: 10.1145/3608443 (cited on page 213).

[32] Dmitry Chistikov, Rupak Majumdar, and Philipp Schepper. “Subcubic certificates for CFL reachability”. In: *Proc. ACM Program. Lang.* 6.POPL (2022), pp. 1–29. DOI: 10.1145/3498702 (cited on page 122).

[33] Frank Ciesinski et al. “Reduction Techniques for Model Checking Markov Decision Processes”. In: *QEST*. IEEE Computer Society, 2008, pp. 45–54. DOI: 10.1109/QEST.2008.45 (cited on page 147).

[34] Edmund M. Clarke et al. “Model Checking and the State Explosion Problem”. In: *LASER Summer School*. Vol. 7682. Lecture Notes in Computer Science. Springer, 2011, pp. 1–30. DOI: 10.1007/978-3-642-35746-6_1 (cited on page 198).

[35] Fredrik Dahlqvist, Alexandra Silva, and Dexter Kozen. “Semantics of Probabilistic Programming: A Gentle Introduction”. In: *Foundations of Probabilistic Programming*. Ed. by Gilles Barthe, Joost-Pieter Katoen, and Alexandra Silva. Cambridge University Press, 2020, pp. 1–42. DOI: 10.1017/9781108770750.002 (cited on page 190).

[36] Manfred Droste and Werner Kuich. “Semirings and Formal Power Series”. In: *Handbook of Weighted Automata*. Ed. by Manfred Droste, Werner Kuich, and Heiko Vogler. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 3–28. DOI: 10.1007/978-3-642-01492-5_1 (cited on pages 15, 20).

[37] Manfred Droste, Werner Kuich, and Heiko Vogler. *Handbook of Weighted Automata*. Springer Science & Business Media, 2009 (cited on page 48).

[38] Manfred Droste and Dietrich Kuske. “Weighted automata”. In: *Handbook of Automata Theory*. Ed. by Jean-Éric Pin. European Mathematical Society Publishing House, Zürich, Switzerland, 2021, pp. 113–150. DOI: 10.4171/AUTOMATA-1/4 (cited on page 48).

[39] Clemens Dubslaff, Christel Baier, and Manuela Berg. “Model checking probabilistic systems against pushdown specifications”. In: *Inf. Process. Lett.* 112.8-9 (2012), pp. 320–328. DOI: 10.1016/j.ipl.2012.01.006 (cited on pages 151, 166, 185).

[40] Philippe Duchon et al. “Boltzmann Samplers for the Random Generation of Combinatorial Structures”. In: *Comb. Probab. Comput.* 13.4-5 (2004), pp. 577–625. DOI: 10.1017/S0963548304006315 (cited on page 1).

[41] Zoltán Ésik. “Iteration Semirings”. In: *Developments in Language Theory*. Vol. 5257. Lecture Notes in Computer Science. Springer, 2008, pp. 1–20. DOI: 10.1007/978-3-540-85780-8_1 (cited on page 48).

[42] Zoltán Ésik and Werner Kuich. “Finite Automata”. In: *Handbook of Weighted Automata*. Ed. by Manfred Droste, Werner Kuich, and Heiko Vogler. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 69–104. DOI: 10.1007/978-3-642-01492-5_3 (cited on pages 14, 157).

[43] Javier Esparza, Stefan Kiefer, and Michael Luttenberger. “Computing the Least Fixed Point of Positive Polynomial Systems”. In: *SIAM J. Comput.* 39.6 (2010), pp. 2282–2335. DOI: 10.1137/090749591 (cited on pages 86, 89, 98, 101, 102, 104, 114, 122, 124, 126, 136, 147).

[44] Javier Esparza, Antonín Kucera, and Richard Mayr. “Model Checking Probabilistic Pushdown Automata”. In: *LICS*. IEEE Computer Society, 2004, pp. 12–21. DOI: 10.1109/LICS.2004.1319596 (cited on pages 3, 10, 49, 50, 52, 54, 56, 58, 84, 151, 152, 158, 168–170, 178, 181, 184, 187).

[45] Javier Esparza, Antonín Kucera, and Richard Mayr. “Quantitative Analysis of Probabilistic Pushdown Automata: Expectations and Variances”. In: *LICS*. IEEE Computer Society, 2005, pp. 117–126. DOI: 10.1109/LICS.2005.39 (cited on pages 10, 84, 121).

[46] Javier Esparza et al. “Efficient Algorithms for Model Checking Pushdown Systems”. In: *CAV*. Vol. 1855. Lecture Notes in Computer Science. Springer, 2000, pp. 232–247. DOI: 10.1007/10722167_20 (cited on page 204).

[47] Kousha Etessami and Mihalis Yannakakis. “Algorithmic Verification of Recursive Probabilistic State Machines”. In: *TACAS*. Vol. 3440. Lecture Notes in Computer Science. Springer, 2005, pp. 253–270. DOI: 10.1007/978-3-540-31980-1_17 (cited on pages 179, 182).

[48] Kousha Etessami and Mihalis Yannakakis. “Model Checking of Recursive Probabilistic Systems”. In: *ACM Trans. Comput. Log.* 13.2 (2012), 12:1–12:40. DOI: 10.1145/2159531.2159534 (cited on page 10).

[49] Kousha Etessami and Mihalis Yannakakis. “Recursive Markov Chains, Stochastic Grammars, and Monotone Systems of Nonlinear Equations”. In: *STACS*. Vol. 3404. Lecture Notes in Computer Science. Springer, 2005, pp. 340–352. DOI: 10.1007/978-3-540-31856-9_28 (cited on pages 10, 147, 184, 187).

[50] Kousha Etessami and Mihalis Yannakakis. “Recursive Markov chains, stochastic grammars, and monotone systems of nonlinear equations”. In: *J. ACM* 56.1 (2009), 1:1–1:66. DOI: 10.1145/1462153.1462154 (cited on pages 3, 10, 86, 92, 93, 121, 122, 124, 136).

[51] Kousha Etessami and Mihalis Yannakakis. “Recursive Markov Decision Processes and Recursive Stochastic Games”. In: *J. ACM* 62.2 (2015), 11:1–11:69. DOI: 10.1145/2699431 (cited on pages 10, 215).

[52] Nathanaël Fijalkow. "Undecidability results for probabilistic automata". In: *ACM SIGLOG News* 4.4 (2017), pp. 10–17. DOI: 10.1145/3157831.3157833 (cited on page 13).

[53] Otto Forster. *Analysis 1: Differential-und Integralrechnung einer Veränderlichen*. Springer, 2012. DOI: 10.1007/978-3-8348-9464-9 (cited on page 101).

[54] Florian Funke, Simon Jantsch, and Christel Baier. "Farkas Certificates and Minimal Witnesses for Probabilistic Reachability Constraints". In: *TACAS (1)*. Vol. 12078. Lecture Notes in Computer Science. Springer, 2020, pp. 324–345. DOI: 10.1007/978-3-030-45190-5_18 (cited on page 122).

[55] Christina Gehnen. "Automata-based model checking of recursive systems". Published on the publication server of RWTH Aachen University in 2024. Bachelor's thesis. RWTH Aachen University, 2021. DOI: 10.18154/RWTH-2024-02598 (cited on pages 11, 163).

[56] Dominik Geißler and Tobias Winkler. "Weighted Automata for Exact Inference in Discrete Probabilistic Programs". In: *ICTAC*. Vol. 16237. Lecture Notes in Computer Science. Springer, 2025, pp. 261–278. DOI: 10.1007/978-3-032-11176-0_16 (cited on page 235).

[57] Lina Gerlach et al. "Efficient Probabilistic Model Checking for Relational Reachability". In: *CAV (1)*. Vol. 15931. Lecture Notes in Computer Science. Springer, 2025, pp. 127–147. DOI: 10.1007/978-3-031-98668-0_6 (cited on page 235).

[58] Noah D Goodman and Andreas Stuhlmüller. *The Design and Implementation of Probabilistic Programming Languages*. http://dippl.org. Accessed: 2026-11-2. 2014 (cited on page 9).

[59] Friedrich Gretz, Joost-Pieter Katoen, and Annabelle McIver. "Operational versus weakest pre-expectation semantics for the probabilistic guarded command language". In: *Perform. Evaluation* 73 (2014), pp. 110–132. DOI: 10.1016/J.PEVA.2013.11.004 (cited on page 213).

[60] Darion Haase et al. "Generating Functions Meet Occupation Measures: Invariant Synthesis for Probabilistic Loops". In: *ESOP (1)*. Vol. 16501. Lecture Notes in Computer Science. Springer, 2026, pp. 314–343. DOI: 10.1007/978-3-032-22720-1_12 (cited on page 235).

[61] Sergiu Hart, Micha Sharir, and Amir Pnueli. "Termination of Probabilistic Concurrent Programs". In: *POPL*. ACM Press, 1982, pp. 1–6. DOI: 10.1145/582153.582154 (cited on page 36).

[62] Arnd Hartmanns and Holger Hermanns. "The Modest Toolset: An Integrated Environment for Quantitative Modelling and Verification". In: *TACAS*. Vol. 8413. Lecture Notes in Computer Science. Springer, 2014, pp. 593–598. DOI: 10.1007/978-3-642-54862-8_51 (cited on page 215).

[63] Arnd Hartmanns and Benjamin Lucien Kaminski. "Optimistic Value Iteration". In: *CAV (2)*. Vol. 12225. Lecture Notes in Computer Science. Springer, 2020, pp. 488–511. DOI: 10.1007/978-3-030-53291-8_26 (cited on pages 122–124, 140, 142, 147).

[64] Arnd Hartmanns et al. "A Practitioner's Guide to MDP Model Checking Algorithms". In: *TACAS (1)*. Vol. 13993. Lecture Notes in Computer Science. Springer, 2023, pp. 469–488. DOI: 10.1007/978-3-031-30823-9_24 (cited on page 123).

[65] Arnd Hartmanns et al. “The revised practitioner’s guide to MDP model checking algorithms”. In: *International Journal on Software Tools for Technology Transfer* (2025). DOI: 10.1007/s10009-026-00848-y (cited on page 147).

[66] Linus Heck et al. “Gradient-Descent for Randomized Controllers Under Partial Observability”. In: *VMCAI*. Vol. 13182. Lecture Notes in Computer Science. Springer, 2022, pp. 127–150. DOI: 10.1007/978-3-030-94583-1_7 (cited on page 34).

[67] Christian Hensel et al. “The probabilistic model checker Storm”. In: *Int. J. Softw. Tools Technol. Transf.* 24.4 (2022), pp. 589–610. DOI: 10.1007/S10009-021-00633-Z (cited on pages 9, 197, 199, 215).

[68] Marijn JH Heule and Armin Biere. “Proofs for satisfiability problems”. In: *All about Proofs, Proofs for all* 55.1 (2015), pp. 1–22 (cited on page 85).

[69] C. A. R. Hoare. “Quicksort”. In: *Comput. J.* 5.1 (1962), pp. 10–15. DOI: 10.1093/COMJNL/5.1.10 (cited on page 1).

[70] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation, 2nd Edition*. Addison-Wesley series in computer science. Addison-Wesley-Longman, 2001 (cited on pages 56, 59).

[71] Thomas F Icard. “Calibrating generative models: The probabilistic Chomsky–Schützenberger hierarchy”. In: *Journal of Mathematical Psychology* 95 (2020), p. 102308. DOI: 10.1016/j.jmp.2019.102308 (cited on page 13).

[72] Nils Jansen et al. “Conditioning in Probabilistic Programming”. In: *MFPS*. Vol. 319. Electronic Notes in Theoretical Computer Science. Elsevier, 2015, pp. 199–216. DOI: 10.1016/J.ENTCS.2015.12.013 (cited on page 213).

[73] Sebastian Junges et al. “The complexity of reachability in parametric Markov decision processes”. In: *J. Comput. Syst. Sci.* 119 (2021), pp. 183–210. DOI: 10.1016/J.JCSS.2021.02.006 (cited on page 233).

[74] Benjamin Lucien Kaminski. “Advanced weakest precondition calculi for probabilistic programs”. PhD thesis. RWTH Aachen University, Germany, 2019. DOI: 10.18154/RWTH-2019-01829 (cited on pages 9, 38, 213).

[75] Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. “On the hardness of analyzing probabilistic programs”. In: *Acta Informatica* 56.3 (2019), pp. 255–285. DOI: 10.1007/S00236-018-0321-1 (cited on page 198).

[76] Benjamin Lucien Kaminski et al. “Weakest Precondition Reasoning for Expected Run-Times of Probabilistic Programs”. In: *ESOP*. Vol. 9632. Lecture Notes in Computer Science. Springer, 2016, pp. 364–389. DOI: 10.1007/978-3-662-49498-1_15 (cited on page 38).

[77] Jan-Christoph Kassing and Jürgen Giesl. “From Innermost to Full Probabilistic Term Rewriting: Almost-Sure Termination, Complexity, and Modularity”. In: *Log. Methods Comput. Sci.* 21.4 (2025). DOI: 10.46298/LMCS-21(4:28)2025 (cited on page 10).

[78] Joost-Pieter Katoen et al. “Linear-Invariant Generation for Probabilistic Programs: - Automated Support for Proof-Based Methods”. In: *SAS*. Vol. 6337. Lecture Notes in Computer Science. Springer, 2010, pp. 390–406. DOI: 10.1007/978-3-642-15769-1_24 (cited on page 9).

[79] Stephen Cole Kleene. *Introduction to Metamathematics*. New York: D. Van Nostrand Company, 1952 (cited on page 90).

[80] Lutz Klinkenberg et al. “Generating Functions for Probabilistic Programs”. In: *LOPSTR*. Vol. 12561. Lecture Notes in Computer Science. Springer, 2020, pp. 231–248. DOI: 10.1007/978-3-030-68446-4_12 (cited on page 233).

[81] Naoki Kobayashi, Ugo Dal Lago, and Charles Grellois. “On the Termination Problem for Probabilistic Higher-Order Recursive Programs”. In: *Log. Methods Comput. Sci.* 16.4 (2020) (cited on page 10).

[82] Bram Kohlen et al. “A Formally Verified IEEE 754 Floating-Point Implementation of Interval Iteration for MDPs”. In: *CAV (2)*. Vol. 15932. Lecture Notes in Computer Science. Springer, 2025, pp. 122–146. DOI: 10.1007/978-3-031-98679-6_6 (cited on page 7).

[83] Dexter Kozen. “A Probabilistic PDL”. In: *STOC*. ACM, 1983, pp. 291–297. DOI: 10.1145/800061.808758 (cited on page 9).

[84] Dexter Kozen. “A Probabilistic PDL”. In: *J. Comput. Syst. Sci.* 30.2 (1985), pp. 162–178. DOI: 10.1016/0022-0000(85)90012-1 (cited on page 38).

[85] Dexter Kozen. “Semantics of Probabilistic Programs”. In: *FOCS*. IEEE Computer Society, 1979, pp. 101–114. DOI: 10.1109/SFCS.1979.38 (cited on page 9).

[86] Antonín Kucera, Javier Esparza, and Richard Mayr. “Model Checking Probabilistic Pushdown Automata”. In: *Log. Methods Comput. Sci.* 2.1 (2006). DOI: 10.2168/LMCS-2(1:2)2006 (cited on pages 6, 10, 152, 184).

[87] Werner Kuich. “Semirings and Formal Power Series: Their Relevance to Formal Languages and Automata”. In: *Handbook of Formal Languages, Volume 1: Word, Language, Grammar*. Ed. by Grzegorz Rozenberg and Arto Salomaa. Springer, 1997, pp. 609–677. DOI: 10.1007/978-3-642-59136-5_9 (cited on pages 13–15, 19–22, 26–28, 30–32, 34, 48–51, 54, 56, 58, 83, 84).

[88] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. “PRISM 4.0: Verification of Probabilistic Real-Time Systems”. In: *CAV*. Vol. 6806. Lecture Notes in Computer Science. Springer, 2011, pp. 585–591. DOI: 10.1007/978-3-642-22110-1_47 (cited on pages 9, 197).

[89] Johannes Lehmann. “Pushdown and expectation transformer semantics of probabilistic recursive programs with nested conditioning”. Published on the publication server of RWTH Aachen University in 2024. Master’s thesis. RWTH Aachen University, 2023. DOI: 10.18154/RWTH-2024-02603 (cited on pages 188, 209, 213).

[90] Christof Löding, P. Madhusudan, and Olivier Serre. “Visibly Pushdown Games”. In: *FSTTCS*. Vol. 3328. Lecture Notes in Computer Science. Springer, 2004, pp. 408–420. DOI: 10.1007/978-3-540-30538-5_34 (cited on pages 151, 152, 156, 159, 160, 166, 169, 225).

[91] Ulrich Loup and Erika Ábrahám. “GiNaCRA: A C++ Library for Real Algebraic Computations”. In: *NASA Formal Methods*. Vol. 6617. Lecture Notes in Computer Science. Springer, 2011, pp. 512–517. DOI: 10.1007/978-3-642-20398-5_41 (cited on page 97).

[92] Jérémie O. Lumbroso. “Optimal Discrete Uniform Generation from Coin Flips, and Applications”. In: *CoRR* abs/1304.1916 (2013). DOI: 10.48550/ARXIV.1304.1916 (cited on pages 198, 199).

[93] Rupak Majumdar and V. R. Sathiyanarayana. "Sound and Complete Proof Rules for Probabilistic Termination". In: *Proc. ACM Program. Lang.* 9.POPL (2025), pp. 1871–1902. DOI: 10.1145/3704899 (cited on page 121).

[94] George Markowsky. "Chain-complete posets and directed sets with applications". In: *Algebra universalis* 6.1 (1976), pp. 53–68 (cited on page 17).

[95] Christoph Matheja. "Automated reasoning and randomization in separation logic". PhD thesis. RWTH Aachen University, Germany, 2020. DOI: 10.18154/RWTH-2020-00940 (cited on pages 188, 213).

[96] Ross M. McConnell et al. "Certifying algorithms". In: *Comput. Sci. Rev.* 5.2 (2011), pp. 119–161. DOI: 10.1016/J.COSREV.2010.09.009 (cited on pages 7, 86, 122).

[97] Annabelle McIver and Carroll Morgan. *Abstraction, Refinement and Proof for Probabilistic Systems*. Monographs in Computer Science. Springer, 2005. DOI: 10.1007/B138392 (cited on pages 9, 38).

[98] Annabelle McIver et al. "A new proof rule for almost-sure termination". In: *Proc. ACM Program. Lang.* 2.POPL (2018), 33:1–33:28. DOI: 10.1145/3158121 (cited on page 121).

[99] Kurt Mehlhorn. "Pebbling Moutain Ranges and its Application of DCFL-Recognition". In: *ICALP*. Vol. 85. Lecture Notes in Computer Science. Springer, 1980, pp. 422–435. DOI: 10.1007/3-540-10003-2_89 (cited on pages 153, 184).

[100] Hannah Mertens et al. "Accurately Computing Expected Visiting Times and Stationary Distributions in Markov Chains". In: *TACAS (2)*. Vol. 14571. Lecture Notes in Computer Science. Springer, 2024, pp. 237–257. DOI: 10.1007/978-3-031-57249-4_12 (cited on page 234).

[101] Hannah Mertens et al. "Computing Expected Visiting Times and Stationary Distributions in Markov Chains: Fast and Accurate". In: *J. Autom. Reason.* 69.3 (2025), p. 23. DOI: 10.1007/S10817-025-09736-7 (cited on pages 147, 199, 235).

[102] Dimitrios Michail et al. "JGraphT—A Java Library for Graph Data Structures and Algorithms". In: *ACM Trans. Math. Softw.* 46.2 (2020), 16:1–16:29. DOI: 10.1145/3381449 (cited on page 141).

[103] Barton P. Miller, Lars Fredriksen, and Bryan So. "An Empirical Study of the Reliability of UNIX Utilities". In: *Commun. ACM* 33.12 (1990), pp. 32–44. DOI: 10.1145/96267.96279 (cited on page 1).

[104] Leonardo Mendonça de Moura and Nikolaj S. Bjørner. "Z3: An Efficient SMT Solver". In: *TACAS*. Vol. 4963. Lecture Notes in Computer Science. Springer, 2008, pp. 337–340. DOI: 10.1007/978-3-540-78800-3_24 (cited on page 142).

[105] Federico Olmedo et al. "Reasoning about Recursive Probabilistic Programs". In: *LICS*. ACM, 2016, pp. 672–681. DOI: 10.1145/2933575.2935317 (cited on pages 10, 45, 143, 188, 213).

[106] Prakash Panangaden. *Labelled Markov Processes*. Imperial College Press, 2009 (cited on page 176).

[107] Gordon D. Plotkin. "The origins of structural operational semantics". In: *J. Log. Algebraic Methods Program.* 60-61 (2004), pp. 3–15. DOI: 10.1016/J.JLAP.2004.03.009 (cited on pages 187, 193).

[108] Amir Pnueli. "The Temporal Logic of Programs". In: *FOCS*. IEEE Computer Society, 1977, pp. 46–57. DOI: 10.1109/SFCS.1977.32 (cited on page 150).

[109] Francesco Pontiggia, Ezio Bartocci, and Michele Chiari. "Model Checking Probabilistic Operator Precedence Automata". In: *ACM Trans. Comput. Logic* 27.4 (Aug. 2026). DOI: 10.1145/3822593 (cited on page 185).

[110] Francesco Pontiggia, Ezio Bartocci, and Michele Chiari. "POPACheck: A Model Checker for Probabilistic Pushdown Automata". In: *CAV (2)*. Vol. 15932. Lecture Notes in Computer Science. Springer, 2025, pp. 105–121. DOI: 10.1007/978-3-031-98679-6_5 (cited on pages 185, 213).

[111] William W. Pugh. "Skip Lists: A Probabilistic Alternative to Balanced Trees". In: *Commun. ACM* 33.6 (1990), pp. 668–676 (cited on page 1).

[112] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley Series in Probability and Statistics. Wiley, 1994. DOI: 10.1002/9780470316887 (cited on page 123).

[113] Uriel G Rothblum. "Nonnegative Matrices and Stochastic Matrices". In: *Handbook of Linear Algebra*. Ed. by Leslie Hogben. CRC press, 2006. Chap. 10. DOI: 10.1201/9781420010572.ch9 (cited on pages 103, 105, 125, 126).

[114] Grzegorz Rozenberg and Arto Salomaa, eds. *Handbook of Formal Languages, Volume 1: Word, Language, Grammar*. Springer, 1997. DOI: 10.1007/978-3-642-59136-5 (cited on page 48).

[115] Yousef Saad. *Numerical Methods for Large Eigenvalue Problems*. Society for Industrial and Applied Mathematics, 2011. DOI: 10.1137/1.9781611970739 (cited on page 137).

[116] Nasser Saheb-Djahromi. "Probabilistic LCF". In: *MFCS*. Vol. 64. Lecture Notes in Computer Science. Springer, 1978, pp. 442–451. DOI: 10.1007/3-540-08921-7_92 (cited on page 36).

[117] Jacques Sakarovitch. "Rational and Recognisable Power Series". In: *Handbook of Weighted Automata*. Ed. by Manfred Droste, Werner Kuich, and Heiko Vogler. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 105–174. DOI: 10.1007/978-3-642-01492-5_4 (cited on page 21).

[118] Philipp Schröer et al. "A Deductive Verification Infrastructure for Probabilistic Programs". In: *Proc. ACM Program. Lang.* 7.OOPSLA2 (2023), pp. 2052–2082. DOI: 10.1145/3622870 (cited on pages 9, 216).

[119] Marcel Paul Schützenberger. "On the Definition of a Family of Automata". In: *Inf. Control.* 4.2-3 (1961), pp. 245–270. DOI: 10.1016/S0019-9958(61)80020-X (cited on page 48).

[120] Alistair Mark Stewart. "Efficient algorithms for infinite-state recursive stochastic models and Newton's method". PhD thesis. University of Edinburgh, 2015 (cited on page 10).

[121] Andreas Stuhlmüller and Noah D. Goodman. "Reasoning about reasoning by nested conditioning: Modeling theory of mind with probabilistic programs". In: *Cogn. Syst. Res.* 28 (2014), pp. 80–99. DOI: 10.1016/J.COGSYS.2013.07.003 (cited on page 1).

[122] Alfred Tarski. "A lattice-theoretical fixpoint theorem and its applications". In: *Pacific Journal of Mathematics* 5.2 (1955), pp. 285–309. DOI: 10.2140/pjm.1955.5.285 (cited on page 95).

[123] Alan Mathison Turing. "Checking a Large Routine". In: *Report of a Conference on High Speed Automatic Calculating Machines*. Cambridge: University Mathematical Laboratory, 1949, pp. 67–69 (cited on page 1).

[124] Jan-Willem van de Meent et al. "An Introduction to Probabilistic Programming". In: *CoRR* abs/1809.10756 (2018) (cited on pages 9, 143).

[125] Moshe Y. Vardi. "Automatic Verification of Probabilistic Concurrent Finite-State Programs". In: *FOCS*. IEEE Computer Society, 1985, pp. 327–338. DOI: 10.1109/SFCS.1985.12 (cited on page 9).

[126] John Von Neumann. "Various techniques used in connection with random digits". In: *Appl. Math Ser* 12.36-38 (1951), p. 5 (cited on page 188).

[127] Di Wang, Jan Hoffmann, and Thomas W. Reps. "Central moment analysis for cost accumulators in probabilistic programs". In: *PLDI*. ACM, 2021, pp. 559–573. DOI: 10.1145/3453483.3454062 (cited on pages 34, 35).

[128] Herbert S Wilf. *generatingfunctionology*. CRC press, 2005 (cited on pages 23, 63).

[129] Simon Wimmer and Joshua von Mutius. "Verified Certification of Reachability Checking for Timed Automata". In: *TACAS (1)*. Vol. 12078. Lecture Notes in Computer Science. Springer, 2020, pp. 425–443. DOI: 10.1007/978-3-030-45190-5_24 (cited on page 122).

[130] Tobias Winkler, Christina Gehnen, and Joost-Pieter Katoen. "Model Checking Temporal Properties of Recursive Probabilistic Programs". In: *CoRR* abs/2111.03501v2 (2021) (cited on page 174).

[131] Tobias Winkler, Christina Gehnen, and Joost-Pieter Katoen. "Model Checking Temporal Properties of Recursive Probabilistic Programs". In: *FoSSaCS*. Vol. 13242. Lecture Notes in Computer Science. Springer, 2022, pp. 449–469. DOI: 10.1007/978-3-030-99253-8_23 (cited on pages 6, 10, 143, 152, 188, 233).

[132] Tobias Winkler, Christina Gehnen, and Joost-Pieter Katoen. "Model Checking Temporal Properties of Recursive Probabilistic Programs". In: *Log. Methods Comput. Sci.* 19.4 (2023). DOI: 10.46298/LMCS-19(4:24)2023 (cited on pages 6, 10, 152, 188, 234).

[133] Tobias Winkler and Joost-Pieter Katoen. *ARTIFACT EVALUATION FOR TACAS '23 SUBMISSION 103: "Certificates for Probabilistic Pushdown Automata via Optimistic Value Iteration" (Version 1)*. Jan. 2023. DOI: 10.5281/ZENODO.7506305 (cited on pages 140, 141).

[134] Tobias Winkler and Joost-Pieter Katoen. "Certificates for Probabilistic Pushdown Automata via Optimistic Value Iteration". In: *TACAS (2)*. Vol. 13994. Lecture Notes in Computer Science. Springer, 2023, pp. 391–409. DOI: 10.1007/978-3-031-30820-8_24 (cited on pages 7, 10, 122, 125, 141, 142, 188, 234).

[135] Tobias Winkler and Joost-Pieter Katoen. "On Certificates, Expected Runtimes, and Termination in Probabilistic Pushdown Automata". In: *LICS*. IEEE, 2023, pp. 1–13. DOI: 10.1109/LICS56636.2023.10175714 (cited on pages 7, 10, 15, 50, 67, 84, 86, 122, 188, 234).

[136] Tobias Winkler and Joost-Pieter Katoen. "On Certificates, Expected Runtimes, and Termination in Probabilistic Pushdown Automata". In: *CoRR* abs/2304.09997 (2023). DOI: 10.48550/arXiv.2304.09997 (cited on page 84).

[137] Tobias Winkler, Johannes Lehmann, and Joost-Pieter Katoen. "Out of Control: Reducing Probabilistic Models by Control-State Elimination". In: *VMCAI*. Vol. 13182. Lecture Notes in Computer Science. Springer, 2022, pp. 450–472. DOI: 10.1007/978-3-030-94583-1_22 (cited on page 233).

[138] Tobias Winkler and Maximilian Weininger. "Stochastic Games with Disjunctions of Multiple Objectives". In: *GandALF*. Vol. 346. EPTCS. 2021, pp. 83–100. DOI: 10.4204/EPTCS.346.6 (cited on page 233).

[139] Tobias Winkler et al. "On the Complexity of Reachability in Parametric Markov Decision Processes". In: *CONCUR*. Vol. 140. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 14:1–14:17. DOI: 10.4230/LIPICS.CONCUR.2019.14 (cited on page 233).

[140] Glynn Winskel. *The Formal Semantics of Programming Languages: An Introduction*. Foundation of computing series. MIT Press, 1993 (cited on pages 17, 190).

[141] Dominik Wojtczak. "Recursive probabilistic models : efficient analysis and implementation". PhD thesis. University of Edinburgh, UK, 2009 (cited on pages 10, 141).

[142] Dominik Wojtczak and Kousha Etessami. "PReMo: An Analyzer for Probabilistic Recursive Models". In: *TACAS*. Vol. 4424. Lecture Notes in Computer Science. Springer, 2007, pp. 66–71. DOI: 10.1007/978-3-540-71209-1_7 (cited on pages 140, 144).

[143] Mihalis Yannakakis and Kousha Etessami. "Checking LTL Properties of Recursive Markov Chains". In: *QEST*. IEEE Computer Society, 2005, pp. 155–165. DOI: 10.1109/QEST.2005.8 (cited on pages 179, 184).

[144] Daniel Zilken. "Distributional invariants for probabilistic programs". Master's thesis. RWTH Aachen University, 2024 (cited on page 198).

[145] Daniel Zilken et al. "Verifying Sampling Algorithms via Distributional Invariants". In: *FM (1)*. Vol. 16556. Lecture Notes in Computer Science. Springer, 2026, pp. 259–278. DOI: 10.1007/978-3-032-26204-2_14 (cited on pages 199, 235).

List of Abbreviations

A

AI Artificial Intelligence. 1, 216

AST Almost-Sure Termination. 36–39, 41, 42, 45–47, 59, 71–75, 77–79, 119–121

B

BSCC Bottom Strongly Connected Component. 177, 180–182, 184

C

CaRet Temporal Logic of Nested Calls and Returns. iii, v, x, 8, 149–153, 161–163, 166, 182, 184, 185, 187, 208

cPAST Conditional Positive Almost-Sure Termination. 42–47, 50, 79–82, 114–118, 121

D

DFA Deterministic Finite Automaton. 13, 14

DTMC Discrete-Time Markov Chain. xi, 29, 35–45, 47, 49, 54, 55, 59, 70–72, 75, 76, 79, 147, 151, 152, 164, 165, 170, 175, 177, 181, 187, 190–199, 201–206

DVPA Deterministic Visibly Pushdown Automaton. x, 149, 151, 152, 156, 158–160, 166–168, 177, 180, 182, 185, 224, 225

E

ETR Existential Theory of the Reals. 149, 152, 178, 179, 181–183

F

FPS Formal Power Series. 20–24, 63, 64

G

GFP Greatest Fixed Point. 95, 110, 121

L

LFP Least Fixed Point. x, 90–100, 105, 109, 110, 112–117, 122–125, 131–133, 135, 139, 140, 143

LTL Linear Temporal Logic. iii, v, 150, 161–163, 172, 184

N

NFA Nondeterministic Finite Automaton. 28, 31, 48

O

OPL Operator Precedence Language. 185

OVI Optimistic Value Iteration. x, 123, 124, 133, 135, 137, 139–142, 144–147, 149, 219

P

PAST Positive Almost-Sure Termination. x, 7, 39–42, 45–47, 75–77, 85, 86, 109, 114, 117–121

PDA Pushdown Automaton. 3, 6, 49, 53, 56, 58, 59

pPDA Probabilistic Pushdown Automaton. iii, v, ix, x, 3–10, 49, 50, 54, 55, 59, 62, 70–83, 85, 86, 88, 109, 114–124, 139, 140, 143, 144, 146, 149–152, 163, 164, 168, 172, 178, 179, 184, 187, 188, 202–208, 210–213, 215, 216, 219, 220

PPS Positive Polynomial System. x, 85, 89–119, 121–136, 138–147, 152, 178, 179, 221, 223

Pray Probabilistic Recursion Analyzer. 9, 140–146, 187, 188, 199, 208, 211, 212

pVPA Probabilistic Visibly Pushdown Automaton. x, 6, 149, 151–153, 164–168, 170–173, 175, 177–180, 182, 184, 185, 208, 224

S

SAT (Boolean) Satisfiability. 85, 86

SCC Strongly Connected Component. 144, 145

SCFG Stochastic Context-Free Grammar. 144, 145

SMT Satisfiability Modulo Theories. 140, 142, 145, 146, 221

V

VPA Visibly Pushdown Automaton. x, 6, 8, 149, 151–160, 163, 164, 166, 167, 169, 182, 184, 185, 225

VPL Visibly Pushdown Language. iii, v, x, 149–151, 154–157, 160, 161, 163, 166, 167, 169, 184, 185

W

WPDA Weighted Pushdown Automaton. 8, 49–56, 58–63, 65, 67, 69, 70, 82, 83, 122