Parameter Synthesis in Markov Models

Sebastian Junges

Parameter Synthesis in Markov Models

Sebastian Junges
orcid.org/0000-0003-0978-8466

Submitted: September 2019
Defended: February 2020
Published: February 2020

Advisor:
Prof. Dr. Ir. Dr. h.c. Joost-Pieter Katoen
Examiners:
Prof. Dr. Alessandro Abate
Prof. Dr. Dr. h.c. Javier Esparza

Cover adapted from Piet Mondriaan, *Tableau I*, 1921.

Parameter Synthesis in Markov Models

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

vorgelegt von

Sebastian Junges, M. Sc.
aus Veldhoven, Niederlande

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

Tag der mündlichen Prüfung: 13. Februar 2020

Diese Dissertation ist auf den Internetseiten der Universitätsbibliothek online verfügbar.

Abstract

Markov models comprise states with probabilistic transitions. The analysis of these models is ubiquitous and studied in, among others, reliability engineering, artificial intelligence, systems biology, and formal methods. Naturally, their analysis crucially depends on the transition probabilities. Often, these probabilities are approximations based on data or reflect configurable parts of a modelled system. To represent the uncertainty about the probabilities, we study *parametric* Markov models, in which the probabilities are symbolic expressions rather than concrete values.

More precisely, we consider parametric Markov decision processes (pMDPs) and parametric Markov chains (pMCs) as special case. Substitution of the parameters yields classical, parameter-free Markov decision processes (MDPs) and Markov chains (MCs). A pMDP thus induces uncountably many MDPs. Each MDP may satisfy reachability and reward properties, such as *the maximal probability that the system reaches an 'offline' state is less than* 0.01%, or *the maximal expected energy consumption is less than* 20 *kWh*. Lifting these properties to pMDPs yields fundamental problems asking: *Is there an induced MDP satisfying the property?* (feasibility), its dual *Do all induced MDPs satisfy the property?* (validity), and advanced problems such as *What is a concise representation for all induced MCs satisfying the property?* We study these problems on a conceptual level, and design and implement both improved and novel algorithms.

On the conceptual side, a thorough discussion of the feasibility problem yields new results, such as: (1) that answering various variants of the feasibility problem is—in terms of complexity—as hard as finding roots of a multivariate polynomial, and (2) that these problems are tightly connected to the analysis of memoryless strategies in partially observable MDPs, a famous model in artificial intelligence. Additionally, we introduce family MCs (fMCs), a subclass of pMCs with finitely many induced MCs. Among others, fMCs define fundamental problems underlying the quantitative analysis of software product lines and sketching of probabilistic programs.

On the algorithmic side, we (1) present and analyse improved but previously known approaches, and combine them to meet practical needs. Their analysis inspired (2) three new and orthogonal approaches, utilising advances in convex optimisation, as well as adapting prominent ideas such as inductive synthesis and abstraction-refinement to the particular setting. All methods efficiently exploit advances in the off-the-shelf analysis of MDPs. On the empirical side, the new methods improve the state-of-the-art considerably, handling hundreds of parameters and millions of states. All the approaches we present have been implemented in open-source tools.

Zusammenfassung

Markow-Modelle umfassen Zustände mit wahrscheinlichkeitsbehafteten Transitionen. Die Analyse dieser Modelle ist allgegenwärtig. Sie ist Forschungsgegenstand, u.a. in der Zuverlässigkeitstechnik, der künstlichen Intelligenz, der Systembiologie und in den formalen Methoden. Die Analyse ist naturgemäß stark abhängig von den Transitionswahrscheinlichkeiten. Diese basieren oft auf Schätzungen oder spiegeln konfigurierbare Komponenten eines Systems wider. Um diesen Ungenauigkeiten gerecht zu werden, betrachten wir parametrische Markow-Modelle, in denen Wahrscheinlichkeiten durch symbolische Ausdrücke statt durch konkrete Werte dargestellt werden.

Insbesondere betrachten wir parametrische Markow-Entscheidungsprozesse (pMDPs), sowie parametrische Markow-Ketten (pMCs) als einen Spezialfall. Das Ersetzen der Parameter induziert die klassischen parameterfreien Markow-Entscheidungsprozesse (MDPs) und Markow-Ketten (MCs). Ein pMDP beschreibt demnach überabzählbar viele MDPs. Jeder MDP kann Erreichbarkeits- und Rewardeigenschaften erfüllen, beispielsweise - *Die maximale Wahrscheinlichkeit, dass das System offline geht, beträgt weniger als 0,01%* oder - *der maximal erwartete Energieverbrauch ist geringer als 20 kWh.* Für solche Eigenschaften und parametrischen Modelle ergeben sich dann folgende fundamentale Fragen: - *Gibt es einen induzierten MDP, der die Eigenschaft erfüllt?* (Feasibility) beziehungsweise das Duale: - *Erfüllen alle induzierten MDPs die Eigenschaft?* (Validity), sowie weiterführende Fragen, zum Beispiel: *Gibt es eine kompakte Darstellung aller induzierten MCs, die eine Eigenschaft erfüllen?* Wir untersuchen diese Fragestellungen konzeptuell, und entwerfen und implementieren sowohl verbesserte als auch neue Algorithmen.

Auf der konzeptuellen Seite bietet eine ausführliche Diskussion neue Ergebnisse, wie zum Beispiel (1) das Beantworten von Varianten von Feasibility ist — von der Komplexität her — genau so schwer wie das Finden von Nullstellen eines multivariaten Polynoms und (2) diese Fragestellungen sind eng verwandt mit der Analyse von memoryless Strategien auf partially observable MDPs, einem elementaren Modell in der künstlichen Intelligenz. Außerdem führen wir Family MCs ein, eine Subklasse von pMCs mit endlich vielen induzierten MCs. Diese beschreiben wichtige Fragen aus der quantitativen Analyse von Software-Produktlinien, und dem Sketching von probabilistischen Programmen.

Auf der algorithmischen Seite (1) präsentieren und analysieren wir verbesserte Ansätze sowie die Kombination solcher. Die Analyse inspiriert (2) drei neue Methoden, welche auf konvexer Optimierung und bedeutenden Ideen aus Inductive Synthesis und Abstraction-Refinement aufbauen. All diese Ansätze nutzen Neuerungen in der Analyse von MDPs aus. Die neuen Methoden verbessern den aktuellen Stand der Technik beträchtlich und sind in der Lage, Hunderte Parameter und Millionen Zustände zu verarbeiten. Alle vorgestellten Ansätze wurden implementiert in Open-Source Tools.

Acknowledgements

First, I would like to thank Joost-Pieter Katoen. Joost-Pieter, you granted me lots of freedom to explore the research area, to find my way and to make my own mistakes. Even more important, you always gave the necessary guidance after such mistakes. Your advice has not only shaped my research, but also my understanding of professional and academic life. I would like to thank Alessandro Abate and Javier Esparza for acting as external examiners and their valuable feedback. Furthermore, I am thankful for Wil van der Aalst and Gerhard Woeginger for acting as committee members. Finally, I would like to thank Nils, Christian, Guillermo, and Tobias for spotting some mistakes in the draft of this thesis, and Angelika and Matthias for their feedback on the *Zusammenfassung*.

Naturally, writing a PhD thesis has been a longer project. After my first year as a student, Erika Ábrahám offered me a position as a student assistant in the theory of hybrid systems group. Erika, you gave excellent advice and your support allowed me to enter my graduate studies with some experiences in research. Working with the theory of hybrid systems group was loads of fun. I would like to thank Florian, Gereon, Johanna, Nils, Ulrich, Stefan and Xin for the warm welcome, the extensive mentoring, and for introducing me to the existential theory of the reals. Eventually, I found out that Joost-Pieter's group was doing actual rocket science (the flyer had a rocket on it!) and it was clear that I wanted to join the Compass project. Viet-Yen and Harold, I enjoyed working for you a lot, and I also want to thank my fellow *minions* Jens and Dimitri. One of the first assignments was regarding the work-flow of fault tree analysis....

Once I started looking for a master thesis, I discussed the possibility to go abroad with Joost-Pieter. He suggested a great group in a truly exotic place, and so I visited Mariëlle Stoelinga in the FMT group in Twente. Mariëlle, thanks for introducing me to all the details that fault trees offer and all the discussions that we had about them. I would like to thank Dennis and Enno for even more discussions on fault trees, and Arend for teaching me a lot, not only about graph grammars. I would also like to thank the FMT group for being such a hospitable group, and all the excellent events were very enjoyable. An honourable mention goes to Tom for his part in all the fun I had.

After finishing my master thesis, I joined the MOVES group. This group has not only been an incredibly kind and fun environment, but also very inspiring. I would like to thank Birgit, Bahare, Benni, Chris, Christina, Christoph, Elke, Falak, Federico, Friedrich, Harold, Jip, Joshua, Kevin, Lutz, Marcin, Matthias, Mojgan, Mingshuai, Nils, Philipp, Shahid, Souymodip, Steven, Tim, Tim and Thomas for contributing to the excellent atmosphere and for all the help that you provided over the years. Some colleagues have been extremely important for the achievements in this thesis: Nils, research with you is so much fun and has been a constant motivation to cooperate. I am also very grateful for everything you taught me regarding day-to-day life in academia.

Chris, you were an incredibly inspiring office mate. While you might underestimate the vast amount you taught me, I do not. Thank you so much. Matthias, we have been working together for almost ten years now, and it has been such a pleasure. Your eye for preciseness and the attention you spend on every aspect of research are admirable. Tim, you always manage to calmly express intricate ideas in a way that they seem simple, and even working in the middle of the night was fun. Benni and Christoph, we journeyed through our PhD (while you were always slightly ahead) and these years and beers together really dragged me through the frustrating bits of academia.

Over the years, I had the luck to have some very smart and nice student assistants: Thomas, Tom, Lutz, Tobias, and Alex, thanks for your efforts. I was also more than lucky to supervise a variety of students, and I hope that I wasn't the only one who learned a lot during these times. I would like to especially thank the students that finished their thesis under my (co-)supervision: Tom, Tim, Ronja, Michael, Dustin, Sebastian, Sven, Jip, and Kevin, I was always looking forward to what you discovered, and only very rarely disappointed.

Special thanks go to all my other co-authors, and with some, I had the joy of writing various papers. Ufuk, you have put forward some fascinating ideas and pushed our research far beyond probabilistic model checking. Murat, your enthusiasm and different perspective made cooperating so much fun. Milan, I enjoyed the long and in-depth discussions we have about sketching of probabilistic programs and hope to continue this line of work. I would like to also thank those not mentioned explicitly. You all contributed to my understanding of science.

Luckily, part of academic life involves the exchange with other groups. I would like to thank Tom for the two weeks travelling through China. I would like to thank Bernd Becker, Krishnendu Chatterjee, Nathanaël Fijalkow, Arnd Hartmanns, Mykel Kochenderfer, Mickael Randour, Jean-Francois Raskin, Sanjit Seshia, Ralf Wimmer, Lijun Zhang for hosting me and the various discussions. Furthermore, I would like to thank the various guests that visited us at the chair, and all organisers of the various conferences and summer schools that I visited for their efforts.

Throughout my studies in Aachen, and afterwards, I have enjoyed the company of many wonderful people. In particular, I would like to thank Björn, Janis, Jens and Max, for all the computer science discussions, but even more for all the fun activities that are only remotely connected to computer science. Björn, of all people I am not married to, you definitively suffered most from my PhD life, and there is not enough Hoegaarden in the world to thank you. It turns out that there are more alien life forms on this planet, often only remotely aware of the joys of computer science and academia. Interacting with these social human beings is an aspect of life I often have neglected, for which I want to sincerely apologise. In particular, I would like to thank Duco, Maarten, Oscar, Marieke, Sophie and Wouter for being excellent friends, even when I am not. There is a lot to say, but here and now I will only stress that you are all so impressive friends. Oscar's motivational speeches and Wouter's interest in MDPs have

affected me more than you might have expected. I would like to thank Jan, Mien, Jenny, Martijn, Ellen, Martijn, Ankie, Rob and Steffi for the warm welcome in their family, for their interest in whatever I was doing, and for understanding that sometimes, work is more urgent than family (without model checking, even computers may starve). My parents (5 out of 5 stars) have continuously supported me in all aspects of my life, and I am deeply grateful for the way they have raised me. Finally, Irma, thank you so much for all the love and support. You are truly amazing, but acknowledgements are not the best place to express what you mean to me.

Contents

1 Introduction 1
1.1 Motivation . . . 1
1.1.1 Markov Models . . . 1
1.1.2 Parameter Synthesis . . . 8
1.2 Contribution and Related Work . . . 14
1.3 Synopsis and Reading Guide . . . 16
1.4 Origins . . . 20

I Background 23

2 Solving Polynomial (In)equalities 25
Foundations on real algebra, satisfiability and optimisation

2.1 Polynomials . . . 25
2.2 Constraints, Formulae, Optimisation problems . . . 27
2.3 Satisfiability in the Existential Theory of the Reals . . . 31

3 Parameter-Free Models 35
Preliminaries on Probabilistic Model Checking

3.1 Markov Models . . . 35
3.1.1 Models . . . 35
3.1.2 Paths and graphs . . . 37
3.1.3 Strategies . . . 39
3.2 Measures and Properties . . . 41
3.2.1 Measures . . . 42
3.2.2 Properties . . . 45
3.2.3 Models with nondeterminism . . . 47
3.3 Analysing Markov Models . . . 50
3.3.1 Qualitative analysis . . . 50
3.3.2 Quantitative analysis . . . 52
3.4 Stochastic Games . . . 54
3.5 Modelling Languages . . . 56

II Understanding Parameter Synthesis 59

4 Parametric Probabilities in MDPs 61
Modelling uncertainty about the model

4.1 Parametric Markov Models 61
4.1.1 Fundamentals 64
4.1.2 Measures and properties on parametric models 67
4.1.3 Sets of valuations 70
4.2 Problem Statements 74
4.3 Encoding into Existential Theory of the Reals 78
4.3.1 Encoding valuations as regions 79
4.3.2 Encoding pMC feasibility 82
4.3.3 Encoding pMDP feasibility 93
4.3.4 Encoding via the solution function 100

5 Aspects of Parameter Synthesis 105
Fundamental traits and computational complexity

5.1 Classes of Parametric Models 105
5.1.1 Subclasses of parametric models 105
5.1.2 Subclasses of valuations 118
5.2 Solution Functions and Accepting Sets 121
5.2.1 The solution function 121
5.2.2 From polynomials to solution functions 127
5.2.3 Accepting sets for pMCs 133
5.3 Somewhere Optimal Strategies 135
5.4 Computational Complexity 139
5.4.1 Qualitative analysis 140
5.4.2 Quantitative reachability 146
5.5 Robust Strategies 157

6 Markov Chain Families 159
Modelling discrete choices

6.1 Family Markov Chains 159
6.1.1 Dedicated model for finite sets of MCs 162
6.1.2 Relation to pMCs 166
6.1.3 Problem statements 169
6.1.4 Aspects of family MCs 171
6.2 Sketching Probabilistic Programs 172
6.2.1 What are sketches? 173
6.2.2 Families versus sketches 175

7 Employing Parameter Synthesis 177
Solving other problems with parameter synthesis

7.1 Partially Observable MDPs . . . 177
7.1.1 What are partially observable MDPs? . . . 180
7.1.2 Relation to parameter synthesis . . . 181
7.1.3 Adding memory . . . 184
7.1.4 Outlook . . . 188
7.1.5 Empirical evaluation . . . 188
7.2 Concurrent Stochastic Reachability Games . . . 192
7.2.1 What are concurrent stochastic reachability games? . . . 193
7.2.2 Relation to parameter synthesis . . . 195
7.3 Sketching and Software Product Lines . . . 197
7.3.1 Illustrative challenges . . . 198
7.3.2 User-centric improvements . . . 201

III Inside Parameter Synthesis 205

8 Basic Methods for Parameter Synthesis 207
Essential techniques for synthesis

8.1 Computing the Solution Function . . . 207
8.1.1 Gaussian elimination . . . 208
8.1.2 State elimination . . . 213
8.1.3 Set-based transition elimination . . . 217
8.1.4 Empirical evaluation . . . 219
8.2 Feasibility and validity checking via ETR solvers . . . 222
8.3 Sampling the Parameter Space . . . 227
8.4 Parameter Space Partitioning . . . 230
8.4.1 Naive refinement loop . . . 231
8.4.2 Finding region candidates . . . 234
8.4.3 Requirements on verification back-ends . . . 238
8.4.4 Empirical evaluation . . . 238
8.5 Bird's Eye View . . . 240

9 Convex Solvers for Feasibility 243
Iteratively solving convex over-approximations

9.1 QCQP Reformulation of Feasibility Synthesis . . . 243
9.2 Efficiently Solving the QCQP . . . 247
9.2.1 A CCP approach to the feasibility problem . . . 247
9.2.2 Integration with model checking . . . 253

9.3 Empirical Evaluation 254
9.3.1 Set-up 254
9.3.2 Evaluation 255

10 Inductive Synthesis for Families 259
Pruning the parameter space by counterexamples

10.1 CEGIS for Markov Chain Families 259
10.1.1 Conflicts and the synthesiser 262
10.1.2 Counterexample-based verifier 263
10.2 CEGIS for Probabilistic Programming Sketches 266
10.2.1 Program-level synthesiser 267
10.2.2 Program-level verifier 271
10.2.3 Program-level CEs for CEGIS 273
10.2.4 Support for flawed programs 276
10.3 Empirical Evaluation 277
10.3.1 Set-up 277
10.3.2 Evaluation 278

11 Abstraction-Refinement for Families 283
Family verification with MDP model checking

11.1 Abstraction 283
11.1.1 All-in-one MDP 284
11.1.2 Abstraction by 'forgetting' 286
11.2 Refinement 289
11.2.1 Refinement loops 291
11.2.2 Informed splitting 292
11.3 Empirical Evaluation 294
11.3.1 Set-up 294
11.3.2 Evaluation 296

12 Parameter Lifting 301
Abstraction-based region verification

12.1 Parameter Lifting for Markov chains 301
12.1.1 Idea 302
12.1.2 Relaxation 304
12.1.3 Lifting 305
12.2 Parameter lifting for MDPs 308
12.2.1 Idea 309
12.2.2 Lifting 309

12.3 Empirical Evaluation 313
12.3.1 Set-up 313
12.3.2 Evaluation 314

IV Parameter Synthesis Today and in the Future 317

13 Tool Support 319

14 Conclusion and Future Work 323

V Appendices 327

A Notation 329

B Complexity 331
B.1 Basics 331
B.2 ETR Completeness 332

Declaration of Authorship 337

Bibliography 339
References 339
Co-authored Publications 363

Index 367

1 Introduction

1.1 Motivation

We study the analysis of sets of Markov models that are described by (variants) of so-called *parametric* Markov models. This work is motivated by the widespread use of Markov models and their successful analysis. We will argue that often, it is more adequate to analyse a set of Markov models rather than a single one, and that parameter synthesis is a fundamental problem underneath the analysis of sets of Markov models.

1.1.1 Markov Models

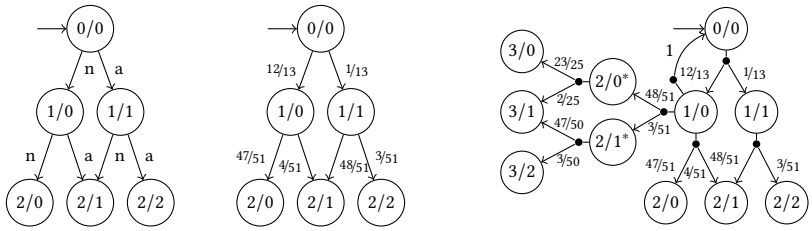
1.1.1.1 Introducing Markov models

Models | When designing a system, models play a crucial role. It is hard (and unnecessary) to give a complete definition of what constitutes a model, so we will stick to the following intuition: "A model of a physical system is a description of certain aspects of the system that is intended to yield insight into properties of the system" [LS16]. To analyse such models, they should be both accurate and unambiguous [BK08].

Models may be created as part of a design process to describe use-cases or interfaces between (sub)systems, as part of a documentation effort, potentially to fulfil requirements by a safety standard, or as a step towards the automated analysis of a system [SST18]. In general, "Modelling is the process of gaining a deeper understanding of a system through imitation. Models imitate the system and reflect properties of the system. Models specify what a system does" [LS16]. This process is not easy: "Modelling systems (...) at the right abstraction level is rather intricate and is really an art" [BK08].

Models come in a plethora of types, varying in mathematical rigour, abstraction, and expressive power. Examples include case study diagrams in the *Unified Modelling Language* [RJB99], *fault trees* [VS02], *Petri nets* [Rei85], and *Kripke structures*, see below.

Transition-based models | Kripke structures [Kri63], or their extension to labelled transition systems, are a simple graph-based formalism to capture, e.g., the behaviour of discrete hard- and software systems. The vertices of the graph describe *states*, the edges in the graph describe *transitions*. A transition describes the possible evolution of the system from one state to another.



(a) Kripke structure (b) Markov chain (c) Markov decision process

Figure 1.1: Models for drawing playing cards

Example 1.1 We consider a deck of 52 playing cards with four aces, from which two cards are drawn. We count the number of aces. The Kripke structure in Figure 1.1a (six states, six transitions) is an adequate model. Initially, we have drawn zero cards, including zero aces. We label the transition with a(ce) or n(o ace), and reach a fresh state accordingly.

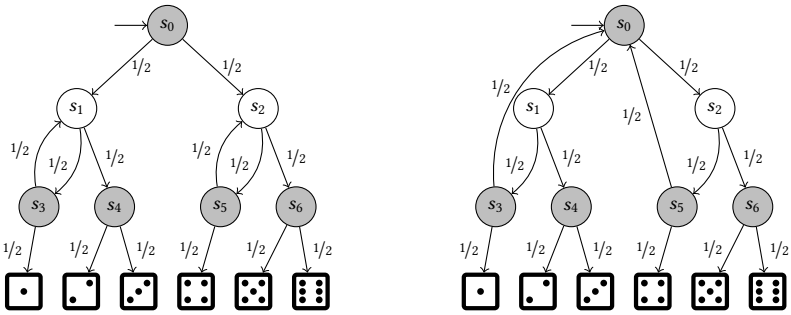
Realistic models often consist of many states and transitions, and are typically described by more accessible/compact modelling languages.

In the example above, the system accurately describes how drawing cards might evolve, but fails to capture that some outcomes are more likely than others. This likelihood is essential when describing systems: One might win all lotteries in a year, but this outcome is significantly less relevant than not winning any lottery. To alleviate this shortcoming of Kripke structures, we annotate transitions with probabilities.

Markov models | Markov models are stochastic state models that exhibit the Markov property: Given any current state, the probability distribution describing the next state is independent of previous states. In this thesis, we use Markov models to refer to *discrete-time Markov chains* (MCs) [Chu67; KS76; Häg02] and *Markov decision processes* (MDPs) [How71; Put94; Kal11]. MCs may be best thought of as a Kripke structure annotated with probabilities such that for each state, the probabilities of the outgoing transitions sum to one, describing a distribution over the successor states.

Example 1.2 We adapt the Kripke structure from Figure 1.1a to an MC by assuming that the deck is perfectly shuffled. Consequently, the probability in the first round to get an ace is $4/52 = 1/13$, and the probabilities change after the first round depending on the result.

MDPs extend MCs with a notion of decision: Instead of a unique distribution over the



(a) Knuth-Yao die (b) Alternative die

Figure 1.2: Modelling a die with fair coin flips.

states, MDPs feature a set of possible distributions at each state. Therefore, one may interpret MDPs as combination of MCs and Kripke structures.

Example 1.3 To illustrate MDPs, we change the rules of the card drawing: if the first card is not an ace, we may choose one of three actions: We continue as before and draw one further card, we start again, or we may draw two further cards at once. Each of the actions is accompanied by a different set of transitions and/or probabilities, as depicted in Figure 1.1c. Thus, in MDPs, some player chooses actions as in Kripke structures, and the outcome of the action is then determined probabilistically as in MCs.

For the remainder of this introduction, we use a prominent example, the Knuth-Yao die [KY76], which uses repeated coin flips to model a six-sided die.

Example 1.4 Figure 1.2a shows the Knuth-Yao die, an MC with 13 states. We start in the initial state, and flip (white and grey) fair coins. If we flip heads, we take the leftmost transition, otherwise, we take the rightmost transition, until we reach one of the states ⚀ – ⚅.

Markov models are state-based models describing stochastic processes.

1.1.1.2 Analysing Markov models

The examples above use MCs to *describe* the behaviour of a system. We want to assess this described behaviour by comparing it with some specification.

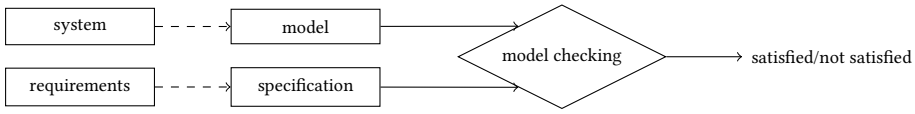


Figure 1.3: Model checking

Properties | We describe the expected behaviour of the systems using *temporal properties*. We exemplify some of these properties on our examples.

> **Example 1.5** For the MC in Figure 1.1b, we may be interested in determining the probability that exactly one ace is drawn. Therefore, we sum the probabilities over all paths that lead to the state 2/1, and along each path, multiply the probabilities. The result is that we get one ace with probability $12 \cdot 4/13 \cdot 51 + 1 \cdot 48/13 \cdot 51 = 2 \cdot 48/13 \cdot 51$. In a more Boolean manner, we may define a property stating that the probability to get exactly one ace is at least $1/7$. As $96/663 > 1/7$, the model satisfies the property. In the MDP, we may ask whether there exist some strategy, i.e., a sequence of action choices, such that the probability to get one ace is above $1/5$ (yes), or whether independently of the strategy, the probability to get one ace is above $1/5$ (no). Whereas in the cards example, these results may also be obtained with high-school combinatorics, the Knuth-Yao die is somewhat more complex through the presence of loops. A natural (satisfied) property is, e.g., that state ⚁ should be reached with probability at least (and at most) $1/6$. Another property is that in expectation, we require less than four coin flips. This model satisfies the property. Another protocol that successfully simulates a die, in Figure 1.2b, does not.

The properties above are instances of temporal properties of MCs [HJ94]. They, e.g., compare the probability to reach a set of target states with some constant, or the expected steps until some condition is met with some constant. Together, a set of properties form a *specification*.

Model checking | Assuming that the model is an adequate representation of the system, we may use the model to check whether the system satisfies the given specification. In particular, we want to automate such a check, as reasoning about systems exhibiting a notion of chance is intricate, famously witnessed by the Monty Hall problem[1]—explained in Example 6.2. Such an automatic check is schematically represented by the scheme in Figure 1.3, and that scheme occurs as *model checking* [EC80; Sif82; VW86; CES09; BK08; CHVB18] in the formal methods community: "Model checking is an automated technique that, given a finite-state model of a system and a formal property,

[1] The one with goats and cars, see also `https://en.wikipedia.org/wiki/Monty_Hall_problem`.

systematically checks whether this property holds for that model" [BK08], and if these models are Markov models, it is called *probabilistic model checking*. Model checking is considered a big success story of formal methods, see [AHV15], which mentions probabilistic model checking as a "promising new direction."

Probabilistic model checking | Probabilistic model checking [CY88; FKNP11; Kat16; BAFK18] is an analysis technique for Markov models with two prominent characteristics: *rigour* and *complete exploration*.

Model checking comes with a particular mathematical rigour: the outcome is either that the property holds, or that it does not. There is no room for a result stating that the property likely holds. While this rigour is sometimes unnecessary, and may harm the performance of the model checker, it is an excellent starting point in understanding the problem of parameter synthesis, and addresses the need for dependable outcomes in various domains. Furthermore, the rigour ensures that the outcome is as good as the model, and may be understood without knowledge of the underlying algorithm.

Regarding exploration, model checking assumes a complete (and thus finite) transition model: "Model checking is a verification technique that explores all possible system states in a brute-force manner" [BK08]. This inspection of all system states is often problematic in the light of the state-space explosion, i.e., it prevents its application on various large scale systems. This may be unnecessary: For stochastic systems, one may often decide a property without seeing all system states—that is much harder in a qualitative setting. Nevertheless, for parameter synthesis, these scalability problems are not a primary concern: after all, the analysis is typically much more expensive than merely constructing the model. However, supporting only finite models does restrict the types of parameter synthesis—more about that later.

Probabilistic model checkers have matured significantly over the last decade. State-of-the-art tools include PRISM [KNP11], Modest [HH14], and storm [9].

> Markov models are well-understood and may be efficiently analysed using probabilistic model checking, a black-box technique that precisely verifies whether a model satisfies a given temporal property.

1.1.1.3 Markov models are everywhere

While Markov models give rise to a beautiful mathematical theory, e.g., in [Kal11], the more practical oriented research in this thesis is motivated by their widespread adoption as an adequate model to understand and design systems that matter. The case study collections of PRISM[1] and QVBS [HKPQ+19][2] contain overviews of Markov

[1] https://www.prismmodelchecker.org/casestudies/
[2] http://qcomp.org/benchmarks/

models that have been collected by the verification community, and include benchmarks from various areas. In the planning community, the probabilistic planning competition [YLWA05][1] contains another set of interesting case studies.

Below, we give a very incomplete overview of application areas. Application areas do not necessarily model the problems directly as Markov model, but mostly use Markov models as the underlying formalism for analysis purposes. We only include areas in which (some) of the relevant properties are temporal properties as outlined above. We group the application areas based on the typical fidelity of the model, in particular, of the fidelity of the probabilities in the model.

Communicating systems | Distributed systems are omnipresent, and have to communicate over lossy channels. The quality of communication channels may be modelled by, e.g., the Gilbert-Elliott model, a two-state MC in which the probability for packet loss depends on the channel's state [MB89]. This channel is then analysed in conjunction with a communication protocol. Communication protocols that have been investigated range from IPv4 ZeroConf [BSHV03; KNPS06], and alternating bit protocols [DJJL01], to more complex (fragments of) IEEE 1394 "FireWire" [SV99; KNS03] and token passing protocols such as EchoRing [4]. We may be interested in the expected throughput or the probability that a packet is delayed more than k rounds. Mostly, the models are precise, and their probabilities are an adequate approximation of an intricate physical process.

Randomised algorithms and security | Randomisation is an essential approach to break the symmetry in systems, or to their diminish worst-case behaviour. Often, the model including its probabilities adequately represents a system. Consider the Carrier Sense Multiple Access/Collision Detection in Ethernet [802a]. When two stations simultaneously attempt to send a packet (giving rise to a collision), an exponential back-off mechanism is used to dodge the collision. In the k-th attempt, a delay out of 2^k possibilities is uniformly drawn. Properties of interest include the expected number of failed attempts or the probability that a packet is repeatedly delayed.

Further examples of randomised algorithms modelled by MCs include: coin-flips in self-stabilising protocols such as the Herman protocol [Her90; KNP12], the randomised choice of selecting actions in neighbour discovery in wireless sensor networks [MB01], the frequency hopping in the device inquiry phase of the Bluetooth protocol [DKNP06], digital signing [EGL85] and non-repudiation [LMT05], and privacy in the Crowds protocol [RR98; NS06], underlying the TOR private surfing software.

Robotics, reliability, ecology and more | Markov models are also created to document and study various technical and non-technical systems. Like in communication

[1] `https://ipc2018-probabilistic.bitbucket.io`

channels, the probabilities are often unknown and typically an abstraction of a far more intricate physical process, and the topology may be a coarse abstraction of the real system. Some areas in which Markov models are used are: In robotics, where probabilities are used to model, e.g., noisy sensor readings or imperfect actuators [TBF05], in all kinds of planning, e.g., for aircraft collision avoidance [EG14], for Mars robot missions [BJMR05], or for scheduling in an energy efficient way [BDDK+14]. In reliability engineering [TB17]—where distributions are used to model the wear of components—to assess the reliability of, e.g., a car driveline [16] or railway stations [VWKN19], or to assess fault-tolerant hardware [Neu56; NPKS05; HBFK+18], in performance modelling [BHHK10], e.g. to investigate climate control systems [HCA17], in adaptive management (ecology), where they are used to model the spread of diseases [VKRJ+18], invasive species [MWLR+07], or wildfires [KT97], and in various branches of (systems) biology to model interactions of cells or molecules [KNP08; KMLB+18]. Finally, MDPs may be used in business process modelling [MEO19].

Markov models in software engineering | With the tighter integration of computing systems with their environment, software interacts with systems described by Markov models mentioned above: Software cannot be studied independently of these interactions, and a tighter integration of probabilistic analysis and software engineering is advocated [Ros16; Hen13]. Much work focusses on the verification of dynamic reconfigurations [CGKM12; FGT12; FGL15; FTG16] and the adaption of software for performance reasons [SCFR17].

Reinforcement learning | (Model-based) reinforcement learning (RL, [SB98]) is a machine learning technique that allows agents to learn how to behave in environments such that the agent maximises some measure, similar to the measures used for the temporal properties. The formalism underlying RL are MDPs. On an abstract level, RL takes similar inputs as probabilistic model checking, but the goals and assumptions are different. The area of safe RL [GF15] aims to add rigour guarantees to these techniques, and is therefore even closer to probabilistic model checking.

Markov models are used to assess the behaviour of the environment that our society lives in, the dependability of systems it increasingly depends upon, and the safety of systems interacting with humans. Their analysis is advocated in the software engineering community, and they underly one of the most prominent techniques in machine learning. Together this might result in an increase in machines making decisions based on the analysis of Markov models.

1.1.2 Parameter Synthesis

We have seen that Markov models and their analysis are widespread. However, any model-based analysis is only as good as the model. It is often hard to obtain precise models that adequately capture a system in different environments, a system under design, an existing system, or a range of slight variations of some system. For instance, packet loss, failure rate, sensor noise, etc. are not fully defined, but are *parametric*. The word parameter[1] has distinct meanings throughout various (sub)disciplines of science in general and computer science in particular. In the scope of this thesis, parameters are meant as a variable aspect of a (Markov) model, which may be set to a range of values, such that fixed parameter values yield a concrete (Markov) model.

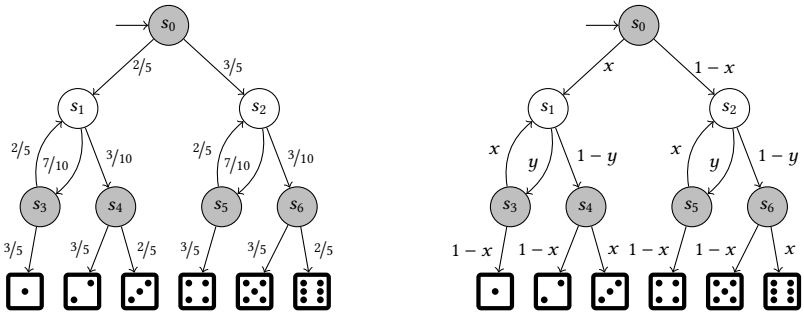
Parameter synthesis problems are a set of analysis problems that do not take a single model, but rather a parametric model which describes a set of models that share some common structure. As such, they can help in understanding and precisely analysing systems with imprecise models. This thesis considers the analysis of *parametric* Markov models where probability distributions are functions over system parameters. In particular, parametric MCs and parametric MDPs refer to models in which the probabilities may be described by symbols, as first considered in [Daw04; LMT07; HHZ11a].

1.1.2.1 Parameter synthesis in a nutshell

We concretise some important aspects from parameter synthesis. Many challenges and approaches are omitted for conciseness and presented in the main part of the thesis.

In Example 1.5, we have seen reachability probabilities for some concrete models. These resulting probabilities are correct for exactly the presented MCs. When constructing these models, however, we made a number or choices that influence the model: We assumed that the cards are perfectly shuffled, i.e., that the probability of getting an ace card depends only on the ratio between aces and non-aces in the deck. Likewise, we assumed that the coins are fair. When we analyse a Markov model, we would like to account for possible variations: If the coins are *almost* fair, how well do we simulate a die? To answer this question, we have to consider not only the MC for the fair coins, but also MCs for various degrees of biased coins.

[1] According to the Oxford University Press dictionary: "A numerical or other measurable factor forming one of a set that defines a system or sets the conditions of its operation."



(a) biased coins

(b) parametric MC

Figure 1.4: Parameterised Knuth-Yao die

Example 1.6 In Figure 1.4a, the MC for the Knuth-Yao die with biased coins is shown. Contrary to the MC with fair coins, state ⚀ is no longer reached with probability at least $3/20$. We are now interested in the effect of different degrees of bias: Therefore, we parameterise the bias. Figure 1.4b depicts a parametric version of the Knuth-Yao die from Example 1.4 with parameters $X = \{x, y\}$: where x is the probability of outcome *heads* in grey states and y the same for white states.

Replacing the parameters in an pMDP with a valuation of them yields a concrete MDP, which we call an instantiation. The set of all possible parameter valuations is called the parameter space. Sometimes, we are only interested in valuations from a particular set of valuations. Such sets are called *regions*.

Example 1.7 The sample valuation $\mathsf{val} := \{x \mapsto 2/5, y \mapsto 7/10\}$ results in the MC from Figure 1.4a, while Figure 1.2a corresponds to $\mathsf{val}' := \{x, y \mapsto 1/2\}$. The parameter space is $\{\mathsf{val} \mid 0 \leq \mathsf{val}(x), \mathsf{val}(y) \leq 1\}$, and almost-fair coins may be described by the region $\{\mathsf{val} \mid 9/20 < \mathsf{val}(x), \mathsf{val}(y) < 11/20\}$.

Now, rather than reporting that the probability is precisely $1/6$ for some concrete model, we want to make a statement about a set of models. That is, we are interested in valuations of the parameters such that the corresponding instantiations satisfy a property. We call such valuations accepting and all others rejecting. The two most prominent questions are now:

- *Feasibility synthesis*: Is there an accepting valuation (from a region)?
- *Region verification*: Are all valuations (from a region) accepting?

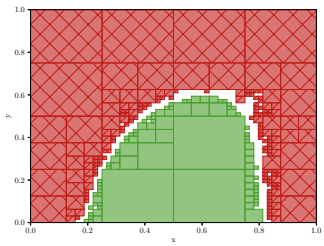


Figure 1.5: Parameter space partitioning

Example 1.8 Consider the pMC $\mathcal{D}$ and a property φ stating that ⚀ is reached with probability at least $3/20$. This property is feasible on the pMC, for instance by using fair coins, as seen before. Let $R = \{\mathsf{val}\colon X \to \mathbb{R} \mid 1/10 \leq \mathsf{val}(x) \leq 9/10 \text{ and } 3/4 \leq \mathsf{val}(y) \leq 5/6\}$ be a region, and the property $\varphi' := \neg\varphi$ constrains the probability to reach ⚀ to be at most $3/20$. The verification problem is to determine whether all valuations in R accept φ', which is equivalent to all valuations in R rejecting φ. As all valuations in R induce a probability to reach ⚀ below $3/20$, the verification problem evaluates to `true`. Thus, R *accepts* φ' and rejects φ.

Various extensions exist, including: *Model repair*: How to minimally tweak a valuation to obtain an accepting valuation? *Closed-form computation*: Find a mapping from valuations to the reachability probability in the corresponding MC. *Parameter space partitioning*: What is (an approximation of) the set of all accepting valuations?

Example 1.9 Reconsider the pMC and property from Example 1.6. The solution function

$$f_\varphi(x, y) = \frac{x \cdot (1 - y) \cdot (1 - x)}{1 - x \cdot y}$$

describes the probability to eventually reach ⚀, assuming that x and y are strictly between zero and one. In Figure 1.5, we approximate the partitioning into accepting and rejecting regions. The green (dotted) area—the union of some smaller rectangular accepting regions—indicates a set of parameter values for which φ is satisfied, whereas the red (hatched) area indicates a set of rejecting regions for φ. The white area indicates regions for which we do not (yet) have a conclusive answer.

These questions (in various combinations) have been previously considered in [HHZ11a; HHZ11b; FTG16; HBK17]. Finally, these parameter synthesis questions are

closely related to model repair [BGKR+11; CHHK+13; PÁJT+15; CK18]. The questions are intrinsically hard as *parameters can take infinitely many different values that also can depend on each other.* Furthermore, certain parameter values may eliminate some transitions in the model, i.e., they may affect the topology of the instantiation. Parameter synthesis may be answered manually or ad-hoc. In this thesis, we focus on automated techniques that apply to a wide range of Markov models.

> In parameter synthesis, we replace concrete transition probabilities in Markov models with symbols, and relate different substitution values for these symbols to the satisfaction of properties.

1.1.2.2 Relevance and adequacy of parameter synthesis

In closing, we discuss how the problems outlined above correspond to the more real-world scenarios sketched earlier. Recall that the parameter-free models that we obtain—i.e, the topology and the probabilities—are often approximations of the reality, and probabilities are often parameters that may be easily exchanged in the model.

Synthesising probabilities | Leaving probabilities unspecified may have various reasons: The system may be in its design phase and therefore some aspects may be unknown, or the model may be based on imprecise data about an existing system, or the system may not have been employed in the environment that we want to assess the system performance in. Thus, we know that some transition may happen, but do not have a precise value for its probability. We then want to determine for which fixed probabilities systems behave either optimally or sufficiently, i.e., which parameter values lead to maximal reliability, maximal performance, minimal energy consumption, minimal unavailability, and so on. To illustrate this further, let us be more concrete:

In various communication protocols on top of lossy channels, the loss rate for packets is often a-priori unknown. It depends on the environment that the system is deployed in—which is prone to changes. Thus, we want to assess the protocol performance for a range of channel qualities. Using our novel techniques, we may find a range of channel qualities within which a complex protocol behaves according to its specification [17]. In fault-tolerant hardware, the failure rates of the hardware component may be only approximated and depending on temperature, interference and other hard-to-model physical processes. Similarly, the performance of error detection schemes is affected. Using our novel techniques, we may find ranges of failure rates that are admissible when requiring the overall system to meet a specified reliability [17; HBFK+18].

In the models above, parameters occur as no adequate, precise values are known, and the system should meet the specification for a range of values. The examples

contain parameters that are said to be *uncontrollable*. Below, we give some concrete examples of systems with *controllable* parameters:

Consider the exponential backoff. An interesting question is whether a uniform distribution is optimal, where optimality refers to the minimal expected time until all packets have been sent. A bias for small delays seems beneficial but raises the collision probability. Using our novel techniques, within a minute we synthesise a different distribution, which induces a smaller expected time compared to the uniform distribution [11]. Likewise, consider the coin flip that determines whether a token should be passed, used in the self-stabilising Herman protocol. To optimise the expected time until stabilisation, the coin flip should be biased—and using our novel techniques, this bias may be synthesised faster than before [AVBK+17]. In a similar vein is also the synthesis of probabilities for adaptive software [CCGK+18; CGJP+16].

In the systems above, the values are left unspecified as they may be selected freely by the system designer such that we optimise their performance or safety. In such a setting, parameter synthesis may also help to reduce the amount of data required to obtain an adequate model [PWHA17]. Likewise, in early stages of reliable system design, the concrete failure rate of components [Cou09] is left unspecified (because not yet known), but due to the choice of a concrete component from a particular manufacturer, parameter synthesis in this stage may be a guidance [MMAG14].

Synthesising structure | Whereas in the scenarios above all parametric transitions are present in the interesting instantiations, the choice of parameter values may affect the topology. We see two reasons:

Transitions should fire with zero probability. Parameters may encode choices, e.g., in robotics: a particular choice should be made with probability x. If such a choice is suboptimal, the synthesiser should take an action with probability zero. Likewise, a parameter might encode a choice of actions that cannot or should not be randomised—randomised actions may harm reproducibility or may be ethically questionable [RVWD13].

Secondly, parameters may encode topology changes. If we parameterise a system in the choice of particular components, the behaviour of these components may not be described by an MC with the same topology, e.g., if a parameter encodes the presence or absence of a redundant component [RANL+15], then the topology of the MC changes. A related example in which we want to analyse topology changes is given when designing randomised algorithms. As an example, reconsider the two alternative die simulations with fair coin flips, as discussed in Example 1.5. More examples in this spirit originate from problems related to parameter synthesis.

Related problems | In this thesis, we show how the classical formulation of parameter synthesis is closely related to other problems. We give a teaser.

Versions of parameter synthesis problems are equivalent to searching for memoryless strategies in partially observable MDPs (POMDPs) [Åst65; KLC98; RN10]. Partially observable MDPs are a prominent extension to MDPs that addresses the lack of full introspection of a system: in many systems, the current state is only partially observable. Such systems occur in domains ranging from robotics over planning to reliability engineering. The analysis of POMDPs is notoriously hard: even basic problems, like the existence of a strategy such that a target state is reached with probability at least one half, are undecidable [MHC03]. Nontheless, we "cannot avoid POMDPs, however, because the real world is one" [RN10], and thus, much effort is put in analysing various subproblems. For computational tractability, strategies are often restricted to finite memory, and in particular the problem that coincides with feasibility is the existence of a memoryless strategy for POMDPs such that a target state is reached with probability at least one half. Actions in the POMDP are then reflected by parameters in the pMC. In similar spirit, one may make a connection to concurrent stochastic reachability games [AHK07], an elementary model for e.g., distributed event systems [AHM00].

Software product lines (SPLs) are sets of systems that contain varying features or components. To assess their performance in terms of energy consumption or reliability, they are subject to probabilistic model checking [GS13; VK13; RANL+15; CDKB18; VBLL18]. Parameter synthesis—especially with discrete parameters—is a fundamental approach that aims to exploit the similarities of the various systems within a product line.

Parameter synthesis is related to robust strategy synthesis in MDPs, i.e., the synthesis of strategies that perform well on various MDPs. Both forms of synthesis start with the impreciseness of the model, but robust strategy synthesis focusses on obtaining the strategy rather than on the influence of the parameters, and thus, the quantification order is swapped. However, parameter synthesis for pMCs and the evaluation problem in robust strategy synthesis coincide. Robust strategies have been widely studied in the field of operations research [NG05; WKR13] and controls [ZS06; WTM12; DFT16], recently drawn some attention in the verification community [HHHL+17; ABCK+18] and are considered in the scope of reinforcement learning [MMX12; LXM16; TPCZ18].

Parameter synthesis allows modelling more systems adequately by not requiring fixed probabilities. Parameter synthesis may also help to optimise reliability, energy consumption, and other measures by synthesising what probabilities system designers should use in randomised protocols. Finally, parameter synthesis is a fundamental problem behind various tasks in areas ranging from artificial intelligence to (software) engineering.

1.2 Contribution and Related Work

The foremost goal of this thesis is to provide a consistent and thorough introduction into techniques for parameter synthesis in MCs and MDPs. Therefore, we set out on an exciting endeavour answering the following questions:

- *What are parametric MCs and parametric MDPs? What is parameter synthesis in the context of these models?* We give a **coherent and comprehensive overview** of the formalism and the various problems that have been dubbed parameter synthesis. We give a precise embedding of feasibility synthesis into the existential theory of the reals (ETR) that formalises parameter synthesis further.

- *What is difficult in parameter synthesis? How difficult is it?* We show that an encoding above is computationally optimal in the sense that for every ETR formula, there is a polynomial-time translation into a feasibility problem. Thus, in terms of complexity, **parameter synthesis is equally hard as solving the existential theory of the reals**—it is NP-hard and in PSPACE. Furthermore, for pMCs, it is shown that the problem is (only) exponential in the number of parameters. This main result is enriched with various subclasses of parametric Markov models and results about them.

- *What does a discrete variant of pMCs look like? Can we use ideas such as sketching to describe them?* We introduce family MCs, a dedicated variant of pMCs that are amenable to **parameter synthesis of parameters with a discrete range and varying induced topology** of the concrete instantiations. We connect this model with a probabilistic sketching language and to parameter synthesis in pMCs.

- *How is parameter synthesis related to other research on probabilistic systems?* There are **close ties** to prominent problems in POMDPs, concurrent stochastic games, and the stochastic analysis of SPLs. In particular, we obtain **new theoretical complexity results** for finding memoryless strategies in POMDPs, and—using parameter synthesis—**improve the state-of-the-art in tool support** for POMDPs and SPLs.

- *What are the basic methods to tackle the parameter synthesis problems?* We present previously known, but **improved generalisations from model checking of parameter-free Markov models to their parametric variants**. We review their strengths and weaknesses and share some insights based on their reimplementation.

- *What are principled approaches that perform empirically well?* We propose **new methods for parameter synthesis** based on convex optimisation, abstraction-refinement and counterexample-guided inductive synthesis, and show that they **improve the state-of-the-art** considerably.

- *What is next?* We discuss the state-of-affairs in tool support and outline avenues for improvement, both on a practical and a more fundamental level.

Related work

We discuss related work directly tied to various chapters in the appropriate places. However, some research directions are directly linked to parameter synthesis in Markov models, but definitively outside the scope of this thesis. For these directions, we give some pointers.

Infinite state | While finite state systems are adequate for a large range of systems, such analysis techniques have some shortcomings when a parameterised set of systems is to be considered e.g., a distributed process with n interacting processes [PZ86]. Here, the number of processes is parameterised, and without an upper bound, the parameterised system is modelled by an infinite state system. An elementary problem is that the analysis of such processes quickly becomes undecidable [AK86], even for Kripke structures. Therefore, many subproblems and special techniques are considered—in particular for non-probabilistic systems: e.g., regular model checking [KMMP+01; AJNd02; BHRV12] representing system states as regular expressions, and counter abstraction together with the use of Petri nets [GS92; FS01; EFM99; PXZ02; ERW19]—the essential idea is to abstract from the identity of the processes. The topic has been covered by various surveys [AJNS04; BJKK+15; AST18]. Particular effort has been made towards the analysis of population protocols, which are used as a model for sensor networks or for chemical reactions [AADF+06; AAER07; NB15; EGLM17].

For probabilistic systems, there is less work. Some early ideas trace back to [PZ86]. There is a dedicated and automated analysis for probabilistic aspects of population models [BEK18]. Alternatively, one may consider the parameters as inputs to imperative probabilistic programs and analyse them with via an iterative reduction to finite-state models [APZ03; EGK12; LR16; ZMT18], or via, e.g., weakest precondition calculi [MM05; KKMO18], super-martingales [CS13] (although this may be hard to automate). Another approach is to rely on analysing infinite state MDPs [HHWZ09; SAM17], or by shifting the focus to the analysis of a more expressive system, such as probabilistic push-down systems [BEKK13], which may be parameterised in the maximal recursion depth.

Continuous time | Continuous-time Markov models have a (negative) exponentially distributed sojourn time, rather than a step-based notion of time. Parametric continuous-time Markov chains (with time-bounded reachability) were first considered by [HKM08]. A method with similarities to parameter lifting has been proposed in [BCDS13]. The method was improved in [CDKP14] and implemented in PRISM-PSY [CPPB+16]. A combination with sampling-based algorithms to find good parameter instantiations is explored in [CCGK+18]. Parameter synthesis with statistical guarantees

has been explored in [BMS16; BS18]. In [GST19], finding good parameter instantiations is considered by identifying subsets of parameters which have a strictly positive or negative influence on the property at hand.

Interval Markov chains | Instead of parametric transitions, interval MCs or MDPs feature intervals at their transitions [JL91; GLD00; KU02; Sku06]. These models allow for uncertainties, but do not allow for parameter dependencies. Therefore, the verification of interval Markov models [SVA06; CSH08; CHK13; HHHL+17] is often more performant, but overly pessimistic. Similarly, it cannot be used to synthesise, e.g., actions that—if taken—must be globally taken. Interval models are often used as an abstraction technique, see e.g., [FLW06; PT16; LWDA18]. Intervals may be equipped with a distribution that encodes the likelihood of being substituted with a particular value, see e.g., [PLSS13], where convex optimisation is utilised.

The lack of dependencies in interval MCs has motivated the introduction of constraint MCs [CDLL+11; DLLP+12] by showing that the parallel composition of two interval MCs may not be adequately captured by interval MCs. Constraint MCs allow additional constraints between the different transitions: the single transitions are labelled with what we call parameters, and the constraints span a set of admissible valuations of these parameters, something that we call a region. Unlike our work, the focus of these papers is on modelling, and these paper focus on (deciding) all kind of relations and closure operations. In similar spirit as our work—and therefore properly referenced in more appropriate places—[Cho17] allows for constraints on interval MCs and discusses the computational complexity.

In [DLP16; BDFL+18], these models are extended to so-called parametric interval MCs, where interval bounds themselves are parametric. Parametric interval MCs extend pMCs. Typical verification approaches are adaptions of a characterisation of reachability in pMCs using the existential theory of the reals.

Underspecified models | In another direction, [AR16] considers models in which the states are not yet fully identified, i.e., where we have no clear knowledge of which states are considered target states. Finally, [RS14] considers MDPs which are to be analysed in multiple environments. A similar idea is underneath hidden-model MDPs [CCMN+12].

1.3 Synopsis and Reading Guide

Chapters start with a short *summary* outlining the structure of that chapter and a paragraph with *background* that points to notions from previous chapters to make the chapter more accessible. Furthermore, a paragraph *origins* reflects which (co-authored) literature is at the heart of the chapter's contents. The next section contains a comprehensive overview of origins of the complete thesis. Below, I outline the

structure of the thesis, and then give some advice to help the reader answer their most pressing questions.

Structure and synopsis

Part I: Background

These chapters contain background material that many readers will be already familiar with. Their main purpose is to fix some notation.

Chapter 2: Solving Polynomial (In)equalities | We briefly discuss notation for formulating and solving formulae from the existential theory of the reals.

Chapter 3: Parameter-Free Models | We briefly recap fundamentals of the standard, parameter-free versions of the Markov models that we use throughout this thesis. Furthermore, we give a brief rehash of specifications, fundamentals of probabilistic model checking, and how models may be described.

Part II: Understanding Parameter Synthesis

This part is dedicated to the *problem* beyond parameter synthesis in Markov models, in particular Markov chains (MCs) and Markov decision processes (MDPs). We formally define the parametric models, various variants of parameter synthesis, analyse elementary properties of these problems, including but not limited to their computational complexity, and show that these problems may be seen as the fundamental problem underlying various—arguably more concrete—problems.

Chapter 4: Parametric Probabilities in MDPs | We formalise parametric MDPs and various of their properties. We furthermore define the synthesis problems considered throughout the thesis, and finally describe feasibility synthesis by formulating step-by-step an equivalent first-order formula from the existential fragment of the theory of the real numbers.

Chapter 5: Aspects of Parameter Synthesis | We explore fundamental aspects of parametric MDPs. In particular, we establish continuity arguments, and discuss the computational complexity of decision problems underlying the synthesis tasks. While the nature of the results is mostly theoretical, some insights shaped the algorithms developed in later chapters.

Chapter 6: Markov Chain Families | Family MCs are an alternative parameterised MC model: They describe finitely many MCs that vary their topology, but do share some sort of structure. We formalise the model and show its connection to parametric MCs. Furthermore, we show how we may consider family MCs as a fundamental variant of semantics for probabilistic program sketches.

Chapter 7: Employing Parameter Synthesis | We explore the relationship to other, well-studied synthesis problems. In particular, we consider partially observable MDPs, concurrent stochastic reachability problems, the analysis of stochastic aspects in product lines, and synthesis of expressions in probabilistic programs in general. Compared to earlier chapters, the focus is more practical, while still mostly formal.

Part III: Inside Parameter Synthesis

This part is dedicated to *solutions* towards parameter synthesis in Markov models. The chapters contain empirical results to illustrate their strengths and weaknesses.

Chapter 8: Basic Methods for Parameter Synthesis | We discuss more-or-less famous approaches to parameter synthesis: in particular, we compute closed forms, use the encoding into the existential theory of the reals and sample parameter values. We combine them into an approximate parameter space partitioning scheme. Finally, we discuss their strengths and weaknesses and outline the approaches that are discussed in the remaining chapters.

Chapter 9: Convex Solvers for Feasibility | We study how to employ convex optimisation solvers towards synthesis in parametric MDPs, and improve the scheme by integrating probabilistic model checking.

Chapter 10: Inductive Synthesis for Families | We study an adaption of the counterexample-guided inductive synthesis scheme to family MCs. The scheme heavily relies on the efficient computation of counterexamples for MCs. In the second part, we lift this approach to the program level.

Chapter 11: Abstraction-Refinement for Families | We consider an abstraction-refinement scheme for family MCs. It employs the power of model checking parameter-free models. The scheme targets the universal variants of parameter synthesis, i.e., it is well-suited to prove the absence of satisfying parameter values.

Chapter 12: Parameter Lifting | This chapter features a similar abstraction as in Chapter 11, but now targets parametric MCs and MDPs.

Part IV: Parameter Synthesis Today and in the Future

This part summarises the current state of affairs in tool support and empirical performance of the approaches. It also discusses areas for improvement. I hope and/or expect that this part might be outdated within a few years.

Chapter 13: Tool Support | We outline tool support for parameter synthesis in STORM, PROPHESY, and DYNASTY, and sketch some implementation decisions.

Chapter 14: Conclusion and Future Work | In this last chapter, we briefly review the state-of-the-art and discuss open questions from a more personal perspective.

What to read?

If you, dear reader, have ample time and want to contribute to parameter synthesis, you are more than welcome to read everything. In all other cases, the following questions/statements may reflect your interests and guide you to the parts that you might find most relevant.

- *What are the key problems considered by parameter synthesis in Markov models?* Please read Section 4.1 to see the formal definitions for all key ingredients, and Section 4.2 for definitions of the most important problem statements. Chapter 8 contains some basic analysis approaches, and in particular Section 8.5 gives an overview.
- *I want to understand the complexity of parameter synthesis.* Please read Chapters 4 and 5 completely.
- *I want to synthesise small strategies for POMDPs/CSRGs.* Please start with Section 4.1, then continue in Chapter 7. You then might want to continue reading about scalable approaches to feasibility synthesis.
- *I want to analyse software product lines./I want to sketch probabilistic programs.* Please start with definitions in Chapter 6, and some more practical insights in Section 7.3. Then, you might want to consider analysis techniques for fMCs/sketches.
- *I want to understand and compute solution functions (rational functions).* Please start with Chapter 4 to get a basic understanding, and continue Section 8.1 to learn about the algorithms. To learn more about solution functions, consider reading Section 5.2.
- *What is a complete method for parameter synthesis?* SMT solvers that handle the ETR encoding for feasibility and validity. The encoding is explained in Section 4.3, the practicalities in Section 8.2.
- *What are scalable approaches to feasibility?* Sampling-based approaches (Section 8.3) and the use of convex optimisation (Chapter 9) are most promising. CEGIS (Chapter 10) is only applicable to fMCs/sketches.

- *What are scalable methods to validity?* Currently only parameter lifting (Chapter 12) and its analogue for fMCs/sketches (Chapter 11) scale best. Rarely, SMT-based techniques are competitive.
- *What are scalable approaches for the analysis of fMCs/sketches?* You may use techniques based on inductive synthesis (Chapter 10) or abstraction-refinement (Chapter 11).

1.4 Origins

The thesis partly represents and sometimes resembles earlier work co-authored by me. Needless to say, it builds upon an extensive body of research in which I took no part.

In accordance with the regulations regarding doctoral studies at the RWTH Aachen University Faculty of Mathematics, Computer Science and Natural Sciences, I include a statement regarding my contributions to (only) these papers. All papers are (naturally) the outcome their authors' efforts. These efforts include vision, asking questions, long discussions[1], frustration, literature studies, implementation efforts, empirical evaluation, and endless polishing of its presentation. I have found no good measures for various aspects of doing research: In my opinion, all aspects are crucial, and crucially, I am grateful for all contributions of my co-authors. In particular, I enjoyed my time as a PhD student as we always considered science to be a team effort.

I summarise only my own recollection of what I (think that I) contributed. The presentation below is necessarily subjective, but my recollections have been backed by at least multiple (if not all) co-authors per paper. The order is roughly chronological, but I deviate to group papers by topic.

- '*PROPhESY: A PRObabilistic ParamEter SYnthesis Tool*', by C. Dehnert, S. Junges, N. Jansen, F. Corzilius, M. Volk, H. Bruintjes, J.-P. Katoen and E. Ábrahám, *CAV*, volume 9206, LNCS, Springer, 2015, pages 214–231 – This paper is the result by early efforts to make parameter synthesis more accessible. In particular, I did not like that the solution function was considered the final outcome of parameter synthesis. It marks the integration of parametric models into the model checker STORM (by me). I led the implementation of the front-end and synthesiser, including connecting STORM and an SMT-solver. Various efficiency improvements in STORM were realised without my help. The writing process was a group effort.

- '*Parameter Synthesis for Markov Models: Faster Than Ever*', by T. Quatmann, C. Dehnert, N. Jansen, S. Junges and J.-P. Katoen, *ATVA*, volume 9938, LNCS, Springer, 2016, pages 50–67 – With the idea to split the parameter space in subregions, and the observation that an exact encoding was not suitable, we together set out to encode a sound

[1] not to mention the amount of coffee consumed.

approximation. My co-authors discovered that this encoding could be interpreted as an MDP model checking query, which opened the doors for an efficient implementation and to support for parametric MDPs. I contributed throughout the whole project, but only had a minor role in the implementation.

- '*Parameter Synthesis for Markov Models*', by S. Junges, E. Ábrahám, C. Hensel, N. Jansen, J.-P. Katoen, T. Quatmann and M. Volk, *CoRR* abs/1903.07993 (2019), under review – After presenting and discussing the ideas from the two papers above for years, we wrote an extensive treatment that further formalises and clarifies those ideas. We re-implemented many of these and other ideas in the tool PROPhESY. I co-led the development of PROPhESY and its STORM bindings, and led the writing process.

- '*Parametric Markov Chains: PCTL Complexity and Fraction-free Gaussian Elimination*', by C. Baier, C. Hensel, L. Hutschenreiter, S. Junges, J.-P. Katoen and J. Klein, *Inf. Comp.* (2020), to appear – This paper is an extended version of [HBK17]. Almost all ideas were developed in [HBK17], and therefore, my contributions were mostly on presentation, discussions on experimental evaluation and formulation.

- '*On the Complexity of Reachability in Parametric Markov Decision Processes*', by T. Winkler, S. Junges, G. A. Pérez and J.-P. Katoen, *CONCUR*, volume 140, LIPIcs, Schloss Dagstuhl - LZI, 2019, 14:1–14:17 – This paper started with various notes and references on traits of parametric MDPs that I collected throughout the years, which my co-authors significantly extended. These efforts led to a body of initial results, and further discussions and literature study led to various additional results. In particular, I established the ETR-completeness results. The writing process was a group effort.

- '*Finite-State Controllers of POMDPs using Parameter Synthesis*', by S. Junges, N. Jansen, R. Wimmer, T. Quatmann, L. Winterer, J.-P. Katoen and B. Becker, *UAI*, AUAI Press, 2018, pages 519–529 – After working on partially observable MDPs in [10], I explored the connection to parameter synthesis. Various aspects were shaped by long and fruitful discussions. I got some help in preparing tool support for the empirical evaluation. The writing process was a group effort.

- '*Synthesis in pMDPs: A Tale of 1001 Parameters*', by M. Cubuktepe, N. Jansen, S. Junges, J.-P. Katoen and U. Topcu, *ATVA*, volume 11138, LNCS, Springer, 2018, pages 160–176 – Sparked by early successful attempts to apply convex optimisation techniques to parameter synthesis in [8], my co-authors experimented with employing a convex-concave procedure. I participated in many discussions to clarify the merits and limitations of the approach, and integrated the idea with model-checking based routines. Furthermore, based on an early prototype, I reimplemented the approach to be applicable on a range of benchmarks, and led the empirical evaluation. The writing process was a group effort.

- '*Counterexample-Driven Synthesis for Probabilistic Program Sketches*', by M. Ceska, C. Hensel, S. Junges and J.-P. Katoen, *FM*, volume 11800, LNCS, Springer, 2019, pages 101–120 – My co-authors sparked the idea to use counterexample guided inductive synthesis to speed up parameter synthesis. After initial discussions, we decided to focus on a discrete variant of the problem. We had many long discussions that were intertwined with my attempts to construct a prototype and my empirical evaluation. Some parts of the improvements on the implementation of counterexample generation were done by my co-authors. The writing process was a collective effort.

- '*Shepherding Hordes of Markov Chains*', by M. Ceska, N. Jansen, S. Junges and J.-P. Katoen, *TACAS*, volume 11428, LNCS, Springer, 2019, pages 172–190 – Inspired by some of the discussions on the paper above, I developed a prototype based on abstraction refinement. My co-authors helped greatly in shaping and improving the theory, and took care of most of the writing process.

- '*Model Repair Revamped*', by M. Ceska, C. Hensel, N. Jansen, S. Junges and J.-P. Katoen, *From Reactive Systems to Cyber-Physical Systems*, volume 11500, LNCS, Springer, 2019, pages 107–125 – This paper discusses the progress from the two papers above on a higher level, and reflects discussions by the authors.

I omit statements about publications whose results are not presented in this thesis. I included a full list of publications that I co-authored on Page 363.

Part I

Background

2 Solving Polynomial (In)equalities

Foundations on real algebra, satisfiability and optimisation

Summary | Polynomial equation systems can model a large variety of systems, ranging from tax to planes. Solving such systems has been a core pursuit of mathematicians throughout history. As polynomial equation systems have a vital role in this thesis, we very briefly collect the necessary fundamentals and mostly use this chapter to fix notations.

Origins | This section does not contain any novel material. We reused part of the presentation from [17; 19]. As different fields use incompatible vocabulary, we performed some contortions to provide consistent notation. Some helpful textbooks are [BPR06; BV04; BHMW09].

Background | Some standard notation is concisely recaptured in Appendix A, and some complexity results are given in Appendix B.1.

2.1 Polynomials

Throughout this thesis, let $\boldsymbol{x}$ denote an (ordered) finite set of *real-valued variables*[1] over $\mathbb{R}$. Let (globally) k denote the number of variables $|\boldsymbol{x}|$. variable

Polynomials as objects | Polynomials are algebraic objects, obtained by the multiplication and addition of variables. The set of polynomials together with these operations constitute a ring.

Definition 2.1 (Polynomial, rational function) For a finite set $\boldsymbol{x} = \{x_1, \ldots, x_k\}$, a *monomial* m is the product of powers of variables: monomial

$$m := x_1^{e_1} \cdot \ldots \cdot x_k^{e_k} \text{ with } e_i \in \mathbb{N}.$$

[1] In examples, we use variables x, y, ... for better readability.

polynomial
term

Let $\mathsf{Mon}[\boldsymbol{x}]$ denote the set of monomials over $\boldsymbol{x}$. A *polynomial* f (over $\boldsymbol{x}$) with t *terms* is a weighted finite sum of monomials with rational weights:

$$f \coloneqq \sum_{j=1}^{t} a_j \cdot m_j \text{ with } a_j \in \mathbb{Q} \setminus \{0\},\ m_j \in \mathsf{Mon}[\boldsymbol{x}].$$

rational function

Let $\mathbb{Q}[\boldsymbol{x}]$ be the set of polynomials over $\boldsymbol{x}$. A *rational function* $h \coloneqq f/g$ over $\boldsymbol{x}$ is a fraction of polynomials $f, g \in \mathbb{Q}[\boldsymbol{x}]$ with $g \neq 0$. Let $\mathbb{Q}(\boldsymbol{x})$ be the set of rational functions over $\boldsymbol{x}$.

Every rational function f/g may be considered as a polynomial (by setting $g = 1$).

Let m be a monomial with exponents $e_1, \ldots, e_k$. The set $\mathsf{occurs}(m)$ contains all variables occurring in m, i.e., $\mathsf{occurs}(m) = \{x_i \mid e_i > 0\}$. Let f be a polynomial with monomials $m_1, \ldots, m_t$ in the terms of f. We overload the notation and use $\mathsf{occurs}(f)$ to denote the set that contains all variables in polynomial f, i.e., $\mathsf{occurs}(f) \coloneqq \bigcup_{j=1}^{t} \mathsf{occurs}(m_j)$. Likewise, we overload the notation for rational functions and use $\mathsf{occurs}(h)$ to denote the set that contains all variables in rational function $h = f/g$, i.e., $\mathsf{occurs}(h) \coloneqq \mathsf{occurs}(f) \cup \mathsf{occurs}(g)$. A rational function h with $\mathsf{occurs}(h) \leq 1$ is called *univariate*, and *multivariate* otherwise.

·univariate

·linear

A monomial is *linear* if $\sum_{i=1}^{k} e_i \leq 1$, *multi-linear* if $e_i \leq 1$ for all $1 \leq i \leq k$, and *quadratic* if $\sum_{i=1}^{k} e_i \leq 2$. Furthermore, there is a unique *constant* monomial where $e_i = 0$ for all $1 \leq i \leq k$. A polynomial f is (multi-)linear (quadratic), if all monomials occurring in f are (multi-)linear (quadratic), respectively. A quadratic and multilinear polynomial is *bilinear*. A linear polynomial is also called affine[1].

·bilinear

Example 2.1 Let x, y denote variables. Then $x^2 + 2$ is a univariate quadratic polynomial. The polynomial $xy + x^2$ is multivariate quadratic, but not multilinear due to the monomial x^2. The polynomial $xy + 3y$ is multilinear and the polynomial $2x + y$ is linear. Finally, x/y^2 is a rational function.

Polynomials as functions | Polynomials may define functions. Therefore, *variable assignments* or *valuations* replace parameters by constant values in rational functions.

valuation

·assignment

Definition 2.2 (Variable assignments) A *variable assignment* of variables $\boldsymbol{x}$ is a function $\mathsf{val}\colon \boldsymbol{x} \to \mathbb{R}$. The set of all valuations for variables $\boldsymbol{x}$ is $\mathsf{Val}_{\boldsymbol{x}}$.

We omit the set of variables if it is clear from the context.

[1] Sometimes, polynomials with a constant monomial are considered not linear. We do not use this stricter notion of linear polynomials.

We may abbreviate the valuation val, $\mathsf{val}(x_i) = a_i \in \mathbb{R}$ by the k-dimensional vector $(a_1, \ldots, a_k) \in \mathbb{R}^k$. Likewise, for a polynomial $f \in \mathbb{Q}(X)$ and a valuation $\mathsf{val}\colon \boldsymbol{x} \to \mathbb{R}$ we write $f[\mathsf{val}]$ for the value $f(\mathsf{val}(x_1), \ldots, \mathsf{val}(x_k))$, i.e. the result of replacing each $x \in \boldsymbol{x}$ in f by $\mathsf{val}(x)$, with subsequent application of $+$ and $\cdot$. For rational function $h = \frac{f}{g}$, let $h[\mathsf{val}] := \frac{f[\mathsf{val}]}{g[\mathsf{val}]} \in \mathbb{R}$ if $g[\mathsf{val}] \neq 0$, and $h[\mathsf{val}] := \bot$ otherwise. A *root* of a polynomial f is a valuation val such that $f[\mathsf{val}] = 0$. The set roots denotes all roots of f, i.e., $\mathsf{roots}(f) := \{\mathsf{val} \in \mathsf{Val} \mid f[\mathsf{val}] = 0\}$. ·root

We may project valuations to a subset of the variables: Let $\mathsf{val} \in \mathsf{Val}_{\boldsymbol{x}}$ and $\boldsymbol{x}' \subseteq \boldsymbol{x}$. Then $\mathsf{val}_{\downarrow \boldsymbol{x}'}\colon \boldsymbol{x}' \to \mathbb{R}$ such that $\mathsf{val}_{\downarrow \boldsymbol{x}'}(x) = \mathsf{val}(x)$ for all $x \in \boldsymbol{x}'$.

Example 2.2 Let $f := xy + 3y$ and $\mathsf{val} := \{x \mapsto 5, y \mapsto 1\}$. Then $f[\mathsf{val}] = 5 \cdot 1 + 3 \cdot 1 = 8$.

Derivatives and convex polynomials | We may rewrite each polynomial $g \in \mathbb{Q}[\boldsymbol{x}]$ w.r.t. a main variable y as

$$g_y := \sum_{i=0}^{n} f_i \cdot y^i$$

with $f_i \in \mathbb{Q}[\boldsymbol{x}']$ and $\boldsymbol{x}' := \boldsymbol{x} \setminus \{y\}$. The partial derivative of g_y w.r.t. the variable y is

$$\frac{\partial g_y}{\partial y} := \sum_{i=0}^{n} i \cdot f_i \cdot y^{i-1}.$$

The *gradient* $\nabla(g)$ of a polynomial g is column vector of dimension k with its i-th entry being the partial derivative of g w.r.t. the i-th variable. The *Hessian* $\mathbf{H}(g)$ of a polynomial g is then a $k \times k$ matrix with polynomial entries such that the entry $\mathbf{H}(g)_{i,j}$ corresponds to taking the partial derivative w.r.t. the variables x_i and x_j, i.e., $\mathbf{H}(g)_{i,j} = \frac{\partial^2 g}{\partial x_i \partial x_j}$. A polynomial g is *convex*, if its Hessian is positive semi-definite. ·gradient ·Hessian ·convex

2.2 Constraints, Formulae, Optimisation problems

Constraints | We use constraints to describe sets of valuations.

Definition 2.3 (Constraint (syntax)) Let $f \in \mathbb{Q}[\boldsymbol{x}]$ and $\sim \in \{<, \leq, =, \geq, >\}$. The (in)equality $f \sim 0$ with $f \in \mathbb{Q}[\boldsymbol{x}]$ is a *(polynomial) constraint*. The set $\Gamma[\boldsymbol{x}]$ contains all constraints over $\boldsymbol{x}$. constraint

We allow $f \sim f'$ with $f, f' \in \mathbb{Q}[\boldsymbol{x}]$ as syntactic sugar. They may be equivalently reformulated as $f - f' \sim 0$. A constraint $f \sim 0$ is an *equation* if $\sim$ is equality. A constraint is linear (quadratic) if f is linear (quadratic), respectively.

Definition 2.4 (Constraint (semantics)) The models relation for constraints, $\models \subseteq \mathsf{Val}_{\boldsymbol{x}} \times \Gamma[\boldsymbol{x}]$, relates valuations and formulae such that:

$$\mathsf{val} \models f \sim 0 \quad \text{iff} \quad f[\mathsf{val}] \sim 0 \text{ for } f \sim 0 \in \Gamma[\boldsymbol{x}].$$

satisfying valuations

The set $\mathsf{sat}(f \sim 0) \subseteq \mathsf{Val}$ of *satisfying valuations* contains all models, i.e.,

$$\mathsf{sat}(f \sim 0) := \{\mathsf{val} \in \mathsf{Val} \mid \mathsf{val} \models f \sim 0\}.$$

Remark 2.1 (Notation of constraints) We face some subtle ambiguity. The *identity* $f = 1$ states that f is the constant polynomial (with a single term) being 1. The *constraint* $f = 1$ is satisfied by any valuation such that $f[\mathsf{val}] = 1[\mathsf{val}]$. Thus, the polynomial $f := x + 1$ (with variable x) satisfies the constraint with valuation $\{x \mapsto 0\}$, but does not satisfy the equivalence $f = 1$. While the meaning is mostly clear from the context, we choose to use the following alternative notation:

relation	$=$	$\leq$	$\geq$	$>$	$<$
notation in constraints	$\doteq$	$\leqslant$	$\geqslant$	$\gtrdot$	$\lessdot$

In particular, this convention ensures that $=$ always indicates the identity relation.

Example 2.3 $1 - x \cdot y \gtrdot 0$ and $x^2 \lessdot 0$ are constraints. For $\mathsf{val} := \{x, y \mapsto 1/2\}$, it holds that $\mathsf{val} \in \mathsf{sat}(1 - x \cdot y \gtrdot 0)$, and for $\mathsf{val}' := \{x, y \mapsto 0\}$, it holds that $\mathsf{val}' \notin \mathsf{sat}(x^2 \lessdot 0)$.

formula

Formulae | A *formula* ψ over a set of constraints is recursively defined: Each constraint is a formula and the Boolean combination of formulae is also a formula.

Definition 2.5 (Formula (syntax)) A (real algebraic, quantifier-free) *formula* ψ over variables $\boldsymbol{x}$ is given by the grammar:

$$\psi := \gamma \mid \neg\psi \mid \psi \wedge \psi \qquad \text{with } \gamma \in \Gamma[\boldsymbol{x}].$$

We let $\Psi[\boldsymbol{x}]$ be the set of all formulae over $\boldsymbol{x}$.

We may add the remaining Boolean connectives, e.g., disjunction $\psi \vee \psi$ and implication $\psi \rightarrow \psi$, and the Boolean constants $\texttt{true} := 0 \doteq 0$ and $\texttt{false}$ as syntactic sugar. Furthermore, we write $g \not\doteq g'$ to denote the *formula* $g \lessdot g' \vee g \gtrdot g'$.

Example 2.4 Let x, y be variables. Examples for formulae are: $\neg\left(x^2 \doteq 0\right)$, $(1 - x \cdot y \gtrdot 0) \vee \left(x^2 \lessdot 0\right)$, and $x^2 + y^2 - 1 \leqslant 0 \wedge x + y - 1 \leqslant 0$.

We often consider formulae with Boolean variables, i.e., variables that are enforced to be 0 or 1. A set of variables may be enforced to be Boolean by adding a conjunction with the formula $\bigwedge_{b \in B}(b \doteq 0 \vee b \doteq 1)$. We refrain from doing this explicitly and simply clarify which variables are Boolean. Similarly, variables with finitely many integer values may be encoded. For a Boolean variable, we may abbreviate $b \doteq 1$ within formulae by b, and $b \doteq 0$ by $\bar{b}$ and call these constraints *literals*. A formula is *Boolean*, if all variables are Boolean and it consists of Boolean connectives and literals. A Boolean formula is also called *propositional*.

literals

·propositional

Example 2.5 Let x, y, z be variables. We may use the propositional formula $(x \vee \bar{y}) \wedge z$ to denote

$$\big((x \doteq 1 \vee y \doteq 0) \wedge z \doteq 1\big) \wedge (x \doteq 0 \vee x \doteq 1) \wedge (y \doteq 0 \vee y \doteq 1) \wedge (z \doteq 0 \vee z \doteq 1).$$

Formulae are implicitly existentially quantified, their semantics are the straightforward extension of the semantics of constraints.

Definition 2.6 (Formula (semantics)) The models relation for formulae relates valuations and formulae: $\models \subseteq \mathsf{Val} \times \Psi[\boldsymbol{x}]$ with

$$\begin{aligned}
\mathsf{val} &\models f \sim 0 && \text{iff} \quad f[\mathsf{val}] \sim 0 \text{ for } f \sim 0 \in \Gamma[\boldsymbol{x}],\\
\mathsf{val} &\models \neg\psi && \text{iff} \quad \mathsf{val} \not\models \psi,\\
\mathsf{val} &\models \psi \wedge \psi' && \text{iff} \quad \mathsf{val} \models \psi, \mathsf{val} \models \psi'.
\end{aligned}$$

The set $\mathsf{sat}(\psi) \subseteq \mathsf{Val}$ of *satisfying valuations* or *solutions* contains all models, i.e.,

satisfying valuation

$$\mathsf{sat}(\psi) := \{\mathsf{val} \in \mathsf{Val} \mid \mathsf{val} \models \psi\}.$$

Two formulae ψ, ψ' are called *equivalent*, if $\mathsf{sat}(\psi) = \mathsf{sat}(\psi')$. Two formulae are called *equisatisfiable* if $\mathsf{sat}(\psi) = \emptyset$ iff $\mathsf{sat}(\psi') = \emptyset$.

·equivalent

·equisatisfiable

Statements of the form $f \sim f'$ with $f, f' \in \mathbb{Q}(\boldsymbol{x})$ are not formulae, handling them requires treatment of $f[\mathsf{val}] = \bot$. In this thesis, such statements are processed using ideas from Section 4.3.1.2.

Definition 2.7 (Constraint system) A *constraint system* $C \subseteq \Gamma[\boldsymbol{x}]$ is a short-hand for the formula $\psi_C := \bigwedge_{c \in C} c$. We overload $\mathsf{sat}(C) := \mathsf{sat}(\psi_C)$.

constraint system

Constraints systems with only equalities are equation systems. If all constraints are linear, then these are linear systems.

Example 2.6 We denote systems of constraints as follows.

Constraints 2.1 (Example Constraints) Let x, y be real-valued variables.

$$x^2 + y^2 \leqslant 1,$$
$$x + y \leqslant 1.$$

The constraints in Constraints 2.1 are equivalent with the formula

$$x^2 + y^2 - 1 \leqslant 0 \wedge x + y - 1 \leqslant 0.$$

Remark 2.2 Throughout this thesis, we are more liberal and denote constraint systems where single entries are formulae, i.e., connected by Boolean connectives.

Sets of valuations | Semi-algebraic sets describe sets of solutions of a formula.

semi-algebraic set

Definition 2.8 (Semi-algebraic set) A *semi-algebraic set* R over $\boldsymbol{x}$ is a set of valuations that satisfy some formula $\psi(R) \in \Psi[\boldsymbol{x}]$, i.e.,

$$R \;=\; \mathsf{sat}(\psi(R)).$$

We call $\psi(R)$ the *representation* of R.

Example 2.7 Let the semi-algebraic set R over $X = \{x, y\}$ be described by

$$\psi(R) := x^2 + y^2 - 1 \leqslant 0 \wedge\; x + y - 1 \leqslant 0.$$

Thus, $R = \{\, \mathsf{val} \mid (x^2{+}y^2{-}1)[\mathsf{val}] \leq 0 \wedge (x{+}y{-}1)[\mathsf{val}] \leq 0 \,\}$. The set R contains the instantiation $\mathsf{val} := (2/5, 3/5)$ as $(2/5)^2 + (3/5)^2 - 1 \leq 0$ and $2/5 + 3/5 - 1 \leq 0$. The instantiation $\mathsf{val}' := (1/2, 3/5) \notin R$ as $1/2 + 3/5 - 1 > 0$. Semi-algebraic sets do not have to describe a connected space; e.g., consider the set R' described by $\psi(R') := -x^2 + 1 < 0$ is $R' = (-\infty, -1] \cup [1, +\infty)$.

A semi-algebraic set may be partitioned into a finite number of connected sets (cells), and—informally—(the boundaries of) each cell may be described by a finite set of polynomials. The size $\|R\|$ of a semi-algebraic set R is given by the Lebesgue measure. All semi-algebraic sets are Lebesgue measurable. A semi-algebraic subset represented by a linear constraint system is convex and called a *convex polytope*.

convex polytope

Optimisation | (Polynomial) optimisation problems[1] with variables $\boldsymbol{x}$ are tuples (obj, Constr), with *objective* $\mathsf{obj} \in \mathbb{Q}[\boldsymbol{x}]$ and *conditions* (or constraints) $\mathsf{Constr} \subseteq \Gamma[\boldsymbol{x}]$. A solution to an optimisation problem is a valuation $\mathsf{val} \in \mathsf{sat}(\mathsf{Constr})$ s.t. $\forall \mathsf{val}' \in \mathsf{sat}$,

[1] We do not write *programs* to avoid confusion with programs describing models, cf. Section 3.5.

$\mathsf{obj}[\mathsf{val}] \leq \mathsf{obj}[\mathsf{val}']$. A *linear program* (*LP*) is an optimisation problem where the objective and the conditions are affine. LP

Example 2.8 In Constraints 2.2 we formulate a nonlinear optimisation problem. It is nonlinear, as one of the conditions are nonlinear.

Constraints 2.2 (Optimisation problem example) Let x, y be variables.

$$\begin{aligned}
&\text{minimise } x + y \text{ such that}\\
&x \leqslant 1,\\
&y \leqslant 1,\\
&x \cdot y \geqslant 1/4.
\end{aligned}$$

The unique solution val to this optimisation problem is defined by $\mathsf{val}(x) = \mathsf{val}(y) = 1/2$. In Constraints 2.3 we formulate a linear program:

Constraints 2.3 (LP example) Let x, y be variables.

$$\begin{aligned}
&\text{minimise } x + y \text{ such that}\\
&x \leqslant 1,\\
&y \leqslant 1,\\
&x + y \geqslant 1.
\end{aligned}$$

Observe that the LP has multiple solutions (all valuations s.t. $\mathsf{val}(x) + \mathsf{val}(y) = 1$).

Another branch of optimisation problems are so called *MaxSat* problems [BHMW09]. MaxSat
A MaxSat instance consists of a propositional formula ψ. The solution to a MaxSat problem is a satisfying valuation with a maximal number of variables assigned to `true`.

2.3 Satisfiability in the Existential Theory of the Reals

Computational complexity results | We collect the most important results about the computational complexity of satisfiability of ETR formulae.

Definition 2.9 (SAT) Given a propositional $\psi \in \Psi[\boldsymbol{x}]$, the satisfiability problem (SAT) asks whether there exists a valuation in $\mathsf{sat}(\psi)$.

For propositional formulae, a standard form is the conjunctive normal form (CNF). A CNF ψ is a conjunction of disjunctions of literals, i.e.,

$$\psi = \bigwedge_i \bigvee_j l_{i,j} \qquad \text{literal } l_{i,j} \in \{b, \bar{b} \mid \text{Boolean variable } b \in \boldsymbol{x}\}.$$

A formula is in 3CNF, if in every disjunction, there are at most three literals. Every propositional formula may be transformed in an equisatisfiable 3CNF by the Tseitin-transformation.

Definition 2.10 (3SAT) Given a propositional formula φ in 3CNF, the satisfiability problem (3SAT) asks whether there exists a valuation in $\mathsf{sat}(\psi)$.

Proposition 2.1 [Coo71] Deciding 3SAT is NP-complete.

Definition 2.11 (ETR-SAT) Given $\psi \in \Psi[\boldsymbol{x}]$, the satisfiability problem in the existential theory of the reals (ETR-SAT) asks whether $\exists \mathsf{val} \in \mathsf{sat}(\psi)$.

ETR denotes the complexity class [SS17] of problems with a polynomial-time many-one reduction to deciding ETR-SAT.

Proposition 2.2 ([Can88]) NP $\subseteq$ ETR $\subseteq$ PSPACE.

A careful analysis of its complexity is given in [Ren92]. Interestingly, the Boolean structure of an ETR formula may be encoded into a polynomial. Then, solving an ETR formula is equally hard as determining whether a polynomial has a root.

Definition 2.12 (ETR-FEAS) Given $f \in \mathbb{Q}[\boldsymbol{x}]$, the feasibility problem asks whether f has a real root, i.e., exists a valuation $\mathsf{val} \in \mathsf{roots}(f)$.

Proposition 2.3 ([SS17]) The problem ETR-FEAS is ETR-complete.

There are two crucial subclasses of ETR that allow for efficient solving. The first fixes the number of variables that may occur in the formulae.

Definition 2.13 (Fixed-Var-ETR) Fix m. Given $\psi \in \Psi[\boldsymbol{x}]$ with $|\boldsymbol{x}| \leq m$, the satisfiability problem in the existential theory of the reals (ETR-SAT$[m]$) asks whether $\exists \mathsf{val} \in \mathsf{sat}(\psi)$.

Proposition 2.4 ([BPR06]) For every m, ETR-SAT$[m] \subseteq$ P.

The second restriction considers the conjunction of linear polynomials.

Definition 2.14 Given a linear constraint system C, the LP satisfiability problem (LP-SAT) asks whether C has a solution, i.e., exists a valuation $\mathsf{val} \in \mathsf{sat}(C)$.

Proposition 2.5 ([Kha79]) LP-SAT $\subseteq$ P.

Practical solving | For the SAT problem, SAT-solvers, often based on the DPLL-algorithm, have been successful in showing remarkable efficiency in terms of empirical run times on a wide range of benchmarks [BHMW09]. SAT-solvers have been extended to deal with various extensions to propositional logic, these extensions are commonly

referred to as SAT modulo theories (SMT) [BHMW09]. In this context, ETR is commonly referred to nonlinear real arithmetic (NRA) and the linear fragment is called linear real arithmetic (LRA). We discuss methods and tools for solving ETR in the context of parameter synthesis in Section 8.2. For linear constraint systems and LPs, the polynomial-time algorithm is based on the so-called ellipsoid method (generalised to interior point methods), but empirically, the simplex algorithm—with exponential worst-case performance—performs often better.

3 Parameter-Free Models

Preliminaries on Probabilistic Model Checking

Summary | We consider Markov chains and Markov decision processes. We concisely present and formally define essential concepts ranging from paths to measures and reachability properties to introduce the notation used in this thesis. Various examples aim to keep the material accessible, but we prefer concise over meaningful examples.

Origins | This chapter does not contain novel material. Its contents are mostly based on [BK08].

Background | Some standard notation is concisely listed in Appendix A.

3.1 Markov Models

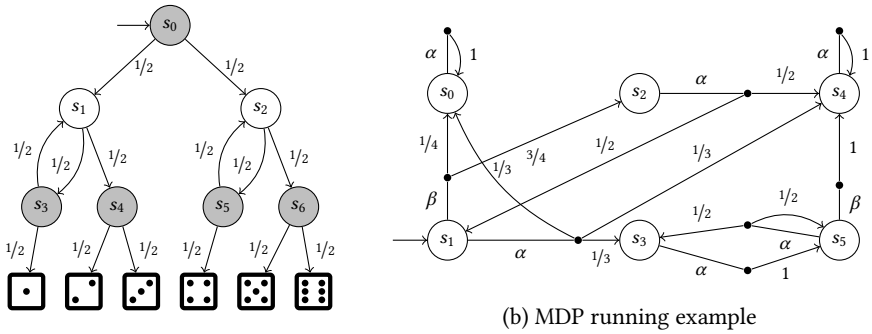
Markov models are stochastic state models that exhibit the Markov property: Given any current state, the probability distribution describing the next state is independent of previous states. In this thesis, we use Markov models to refer to *discrete-time Markov chains* (MCs) [Chu67; KS76; Häg02] and *Markov decision processes* (MDPs) [How71; Put94; Kal11]. We use the notation from the verification community[1]—in particular, from [BK08]. We refer to the introduction for motivation and restrict ourselves to the technicalities in this chapter.

3.1.1 Models

Definition 3.1 (Markov decision process) A *Markov decision process* (*MDP*) is a tuple $M \coloneqq (S, \iota, Act, P, \mathsf{rew})$ where S is a set of *states*, $\iota \in S$ is an *initial state*, Act is a finite set of *actions*, $P \colon S \times Act \nrightarrow Distr(S)$ is a partial *transition probability* function, and rew are *state rewards* $\mathsf{rew} \colon S \to \mathbb{R}_{\geq 0}$. MDP rewards

We denote $P(s, \alpha)(s')$ with $P(s, \alpha, s')$. Let $\mathbb{A}\mathrm{ct}(s) \coloneqq \{\alpha \in Act \mid P(s, \alpha) \neq \bot\}$ denote the

[1] In particular, we are not interested in so-called discounted rewards and therefore omit a discount factor.



(a) MC running example

(b) MDP running example

Figure 3.1: Running examples

·available actions
self-loop

available actions in state s. W.l.o.g. we assume that $|\mathbb{A}\mathrm{ct}(s)| \geq 1$ for all $s \in S$. Furthermore, we refer to a transition $P(s, \alpha, s) = 1$ as a *self-loop*.

Example 3.1 We depict an MDP in Figure 3.1b. The MDP has six states, the initial state is s_1. In the initial state, there are two available actions α and β, each with their own probabilistic outcome, e.g. $P(s_1, \beta) = \{s_0 \mapsto 1/4, s_2 \mapsto 3/4\}$. In state s_2, there is only one available action. We have not depicted any rewards.

MC

Definition 3.2 (Markov Chain) A (*discrete-time*) *Markov chain* (*MC*) is an MDP such that $|\mathbb{A}\mathrm{ct}(s)| = 1$ for all states $s \in S$. We may denote an MC as tuple $D := (S, \iota, P, \mathsf{rew})$ with S, ι, rew as in Definition 3.1 and *transition probability* function $P\colon S \to \mathit{Distr}(S)$.

For Markov chains, we denote $P(s)(s')$ with $P(s, s')$.

Example 3.2 In Figure 3.1a, we depict the Knuth-Yao die, an MC that we have seen before in Example 1.4: we model the outcome of a die with successive coin flips. The MC consists of 13 states: the lower six states are marked by their semantics: the outcome of a die. We omit the self-loops in these states. All other states are equipped with a uniform distribution over two states.

In both MCs and MDPs, we sometimes omit the specification of rewards and assume reward assigns zero to all states. MDPs are often referred to as *nondeterministic* models, as the action choice in the states is unspecified. We refer to states with more than one available action as *nondeterministic states*.

nondeterminism

·nondeterministic

3.1.2 Paths and graphs

3.1.2.1 Paths

We fix an MDP $M := (S, \iota, Act, P, \mathsf{rew})$. A *path* of an MDP M is an (in)finite sequence $\pi := s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} \cdots$, where $s_i \in S$, $\alpha_i \in Act(s_i)$, and $P(s_i, \alpha_i, s_{i+1}) \neq 0$ for all $i \in \mathbb{N}$. The set Π^M of paths in M is the union of finite paths Π^M_{fin} and infinite paths Π^M_{∞}. The notions of paths carry over to MCs (actions are omitted). path

For finite $\pi = s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} \ldots s_n$, we furthermore define the *length* $|\pi| := n + 1$ of π. For infinite paths, we set $|\pi| := \infty$. A prefix of path π is a finite path π' such that the sequence of states and actions in π' is mimicked by π. A prefix is a *proper prefix*, if $|\pi'| < |\pi|$. For any path $\pi = s_0 \xrightarrow{\alpha_0} s_1 \ldots$ we set $\pi@i := s_i$. A path π *visits* a state s, if there is some $i \in \mathbb{N}$ such that $\pi@i = s$. Specifically, $\mathsf{first}(\pi) := s_0$ is the *first state* of π, and for finite π, $\mathsf{last}(\pi) := \pi@|\pi|$ is the *last state* of π. ·length ·visits ·first state ·last state

3

Example 3.3 An example of a finite path in the MDP from Figure 3.1b is:

$$\pi = s_1 \xrightarrow{\beta} s_2 \xrightarrow{\alpha} s_1 \xrightarrow{\alpha} s_0 \xrightarrow{\alpha} s_0.$$

The path has length 4, visits states s_0, s_1, s_2 and state s_0 is the last state. The path may be extended to an infinite path by taking the loop in s_0 indefinitely.

3.1.2.2 Path sets

We mostly consider paths between or from fixed states. Let $S' \subseteq S$ be a subset of the states and $s \in S$ a state. The set

$$\Pi^M(S') := \{\pi \in \Pi^M \mid \mathsf{first}(\pi) \in S'\}$$

contains all paths starting in some $s \in S'$. We simplify the notation of the set $\Pi^M(\{s\})$ to $\Pi^M(s)$. Analogously, $\Pi^M_{\mathsf{fin}}(s)$, $\Pi^M_{\infty}(s)$, $\Pi^M_{\mathsf{fin}}(S')$, $\Pi^M_{\infty}(S')$ define the (in)finite paths starting in some $s' \in S'$, respectively. The set

$$\Pi^M(\Diamond S') := \{\pi \in \Pi^M_{\mathsf{fin}} \mid \mathsf{last}(\pi) \in S' \wedge \forall i < |\pi| : \pi@i \notin S'\}$$

contains all finite(!) paths that end in S' with no proper prefix visiting a state $s \in S$. Again, we simplify notation of $\Pi^M(\Diamond\{s\})$ to $\Pi^M(\Diamond s)$. Similarly, for any *horizon* $h \in \mathbb{N}$ the set

$$\Pi^M(\Diamond^{\leq h} S') := \{\pi \in \Pi^M_{\mathsf{fin}} \mid \mathsf{last}(\pi) \in S' \wedge |\pi| \leq h \wedge \forall i < |\pi| : \pi@i \notin S'\}$$

contains all paths of length at most h that end in S' with no proper prefix visiting a state $s \in S$. For subsets $S', T \subseteq S$, the sets

$$\Pi^M(S', \Diamond T) := \Pi^M(S') \cap \Pi^M(\Diamond T), \quad \text{and}$$
$$\Pi^M(S', \Diamond^{\leq h} T) := \Pi^M(S') \cap \Pi^M(\Diamond^{\leq h} T)$$

denotes the paths of states starting in S' and reaching T (in h steps). We may simplify notation for singleton sets as above.

Example 3.4 Reconsider the MDP from Figure 3.1b: The paths in $\Pi^M(\Diamond\{s_0\})$ are of the form

$$s_1\big(\xrightarrow{\beta} s_2 \xrightarrow{\alpha} s_1\big)^* \xrightarrow{\alpha} s_0 \quad \text{or} \quad s_1\big(\xrightarrow{\beta} s_2 \xrightarrow{\alpha} s_1\big)^* \xrightarrow{\beta} s_0,$$

where the * denotes the Kleene-star for arbitrary repetition. The path from Example 3.3 is not in $\Pi^M(\Diamond\{s_0\})$, as a proper prefix of the path already visits s_0.

3.1.2.3 Underlying graphs

MDPs may be considered as annotated graphs and this perspective helps in describing many operations on MDPs.

underlying digraph

Definition 3.3 For an MDP $M = (S, \iota, P, \mathsf{rew})$, the *underlying digraph* of M is $G(M) := (S, E(M))$ with

$$E(M) := \{(s, s') \mid \exists \alpha \in \mathbb{A}\mathrm{ct}(s) : P(s, \alpha, s') > 0\}.$$

This definition then immediately allows to lift various definitions from graphs to MDPs. For completeness, we define the relevant notions below.

successors

The set $\mathsf{succ}(s, \alpha)$ of *successors* of state/action pair s, α is

$$\mathsf{succ}(s, \alpha) := \mathit{supp}(P(s, \alpha)),$$

and the set of successors of a states S' is

$$\mathsf{succ}(S') := \{s' \in S \mid \exists s \in S' \, \exists \alpha \in \mathbb{A}\mathrm{ct}(s) : s' \in \mathsf{succ}(s, \alpha)\}.$$

reachable states

In particular, we write $\mathsf{succ}(s)$ to denote $\mathsf{succ}(\{s\})$. The *reachable states* $\mathsf{reach}(S)$ are the transitive closure of the successor relation, i.e.,

$$\mathsf{reach}(S') := \{s \in S \mid \Pi^M(S', \Diamond\{s\}) \neq \emptyset\}.$$

We denote $\mathsf{reach}(\{s\})$ with $\mathsf{reach}(s)$ and $\mathsf{reach}(\{\iota\})$ with $\mathsf{reach}(M)$. Likewise, we may define predecessors of some states. The set $\mathsf{prec}(S')$ of *predecessors* of states S' is

predecessors

$$\mathsf{prec}(S') := \{s' \in S \mid \exists \alpha \in \mathbb{A}\mathrm{ct}(s') : \mathsf{succ}(s', \alpha) \cap S' \neq \emptyset\},$$

and we write $\mathsf{prec}(s)$ for $\mathsf{prec}(\{s\})$. A set $S' \subseteq S$ of states is *strongly connected*, if for all states $s, s' \in S'$, there exists a path from s to s', i.e., strongly connected

$$S' \text{ is a SCC} \quad \text{iff} \quad \Pi^M(s, \Diamond s') \neq \emptyset.$$

A *strongly connected component* (*SCC*) of M is a subset S' of states such that no true subset of S' is strongly connected. An SCC is *trivial* if the SCC is a singleton. A *bottom SCC* (*BSCC*) is an SCC S' such that from S', no states outside of S' may be reached, i.e., $\Pi^M(S', \Diamond S \setminus S') = \emptyset$. The set $\mathsf{BSCC}_M := \bigcup\{S' \subseteq S \mid S' \text{ is a BSCC}\}$ denotes all states in some bottom SCC. A state s is called *absorbing*, if $P(s, \alpha, s) = 1$ for all $\alpha \in \mathbb{A}\mathrm{ct}(s)$. Absorbing states constitute trivial SCCs. SCC BSCC ·absorbing

Example 3.5 Reconsider the MDP from Figure 3.1b: We have $\mathsf{succ}(s_1, \beta) = \{s_0, s_2\}$, and $\mathsf{succ}(s_1) = \{s_0, s_2, s_3, s_4\}$. Furthermore, $\mathsf{reach}(M) := \mathsf{reach}(s_1) = \{s_0, \ldots, s_5\}$, $\mathsf{reach}(s_3) = \{s_3, s_4, s_5\}$, and finally $\mathsf{prec}(s_4) = \{s_1, s_2, s_5\}$. The SCCs are $\{s_0\}, \{s_1, s_2\}, \{s_3, s_5\}, \{s_4\}$. The BSCCs are $\{s_0\}$ and $\{s_4\}$. In Figure 3.1a, the state s_0 is not part of any SCC.

In these notions, we do not reflect the combination of actions and probabilistic outcomes. Various notions on graphs under MDPs, most prominently, *maximal end components*, do take the differences between actions and probabilistic outcomes into account. These notions, however, are not directly relevant for this thesis.

3.1.3 Strategies

Action choices have to be resolved before the probability of paths may be determined. Actions are resolved using *strategies*.

Definition 3.4 (Strategy) A *strategy* for an MDP $M = (S, \iota, Act, P, \mathsf{rew})$ is a (measurable) function strategy

$$\sigma \colon \Pi^M_{\mathsf{fin}} \to Distr(Act) \quad \text{such that } supp(\sigma(\pi)) \subseteq \mathbb{A}\mathrm{ct}(\mathsf{last}(\pi)) \text{ for all } \pi \in \Pi^M_{\mathsf{fin}}.$$

- A strategy is *memoryless*, if ·memoryless

$$\mathsf{last}(\pi) = \mathsf{last}(\pi') \text{ implies } \sigma(\pi) = \sigma(\pi') \quad \text{for all } \pi, \pi' \in \Pi^M_{\mathsf{fin}}.$$

- A strategy is *deterministic*, if ·deterministic

$$|supp(\sigma(\pi))| = 1 \quad \text{for all } \pi \in \Pi^M_{\mathsf{fin}}.$$

- A strategy is *deterministic memoryless*, if it is deterministic and memoryless.

The set of all strategies of M is $\mathfrak{S}^M$, the set of all memoryless strategies is $\mathfrak{S}^M_m$, and the set of all deterministic memoryless strategies is Σ^M.

Remark 3.1 The literature also refers to strategies as *schedulers* or *policies*. We consider *finite-memory* and *observation-based* strategies in Section 7.1.

We may use the function signature $\sigma\colon \Pi^M_{\mathsf{fin}} \to \mathit{Act}$ for deterministic strategies, and $\sigma\colon S \to \mathit{Distr}(\mathit{Act})$ for memoryless strategies.

Example 3.6 Reconsider the MDP from Figure 3.1b. When denoting strategies, it suffices to specify only paths where the last state is nondeterministic. An example of a strategy is defined by

$$\pi \mapsto \begin{cases} \{\alpha \mapsto 1/2, \beta \mapsto 1/2\} & \text{if } \mathsf{last}(\pi) = s_1 \text{ and } |\pi| < 10, \\ \{\alpha \mapsto 1, \beta \mapsto 0\} & \text{if } \mathsf{last}(\pi) = s_1 \text{ and } |\pi| > 10, \\ \{\alpha \mapsto 1/3, \beta \mapsto 2/3\} & \text{if } \mathsf{last}(\pi) = s_5. \end{cases}$$

This strategy is neither memoryless nor deterministic. The strategy

$$\sigma_2 = \{s_1 \mapsto \{\alpha \mapsto 1, \beta \mapsto 0\}, s_5 \mapsto \{\alpha \mapsto 1/3, \beta \mapsto 2/3\}\}$$

is memoryless, but not deterministic. The strategy

$$\sigma_3 = \{s_1 \mapsto \beta, s_5 \mapsto \alpha\}$$

is memoryless deterministic.

induced MC

Definition 3.5 (Induced Markov chain) Let $M = (S, \iota, \mathit{Act}, P, \mathsf{rew})$ be an MDP and $\sigma \in \mathfrak{S}^M$ a strategy. The *induced MC* of M and σ is given by $M[\sigma] := (\Pi^M_{\mathsf{fin}}, \iota, P[\sigma], \mathsf{rew}')$ where

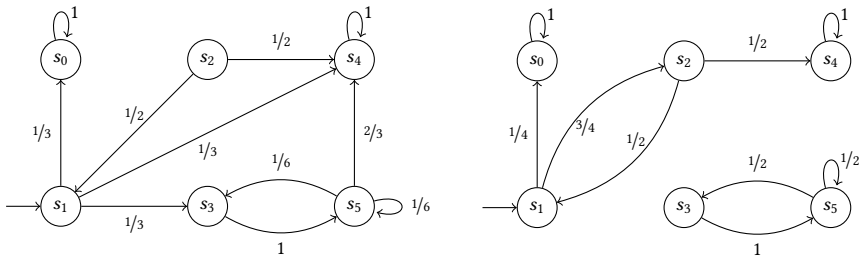
$$P[\sigma](\pi, \pi') := \begin{cases} P(\mathsf{last}(\pi), \alpha, s') \cdot \sigma(\pi)(\alpha) & \text{if } \pi' = \pi \xrightarrow{\alpha} s', \\ 0 & \text{otherwise,} \end{cases}$$

and $\mathsf{rew}'(\pi) = \mathsf{rew}(\mathsf{last}(\pi))$.

Remark 3.2 Let $M = (S, \iota, \mathit{Act}, P, \mathsf{rew})$ be an MDP. For memoryless strategies σ, the MC $M[\sigma]$ may be identified with the finite MC $M[\sigma]' := (S, \iota, P[\sigma]', \mathsf{rew})$ where

$$P[\sigma](s, s') := \sum_{\alpha \in \mathbb{A}\mathrm{ct}(s)} P(s, \alpha, s') \cdot \sigma(s)(\alpha).$$

Formally, $M[\sigma]$ is *probabilistic bisimilar* (cf. [BK08, Sect. 10.4.2]) to $M[\sigma]'$. In particular, all reachability probabilities are preserved. This equivalence justifies using $M[\sigma]'$ as redefinition of $M[\sigma]$ for memoryless strategies.



(a) MC $M[\sigma_2]$ (b) MC $M[\sigma_3]$

Figure 3.2: Induced MCs

Example 3.7 As this thesis mostly uses memoryless strategies, we only give examples for induced MCs of the memoryless strategies from Example 3.6. We depict the induced MCs in Figure 3.2.

3.2 Measures and Properties

We fix an MC $D = (S, \iota, P, \mathsf{rew})$ with *target states* $T \subseteq S$, and we are interested in asking questions like: target states

- "What is the probability to reach a specified set of *target states*?"
- "What is the probability to reach the target states *within h steps*?"
- "What is the *expected number of steps* to reach the target states?"
- "What is the *expected reward collected* until we reach the target states?"

Remark 3.3 We are never interested in what happens once we have reached a target state. This observation motivates the following restriction. W.l.o.g. target states are absorbing and have no reward attached, i.e., $\mathsf{rew}(s) = 0$ for all $s \in T$. In MCs where target states $t \in T$ are not absorbing, we simply make them absorbing by removing all available actions and adding an action $\alpha \in Act$ with $P(t, \alpha, t) = 1$. Target states are then trivial BSCCs.

From any state in a BSCC which does not contain a target state, the target states cannot be reached. This set $B_T \subseteq S$ are the *bad states* for reaching T in D, i.e., bad states

$$B_{T,D} \coloneqq \mathsf{BSCC}_D \setminus T.$$

We typically omit the subscripts and simply refer to the bad states.

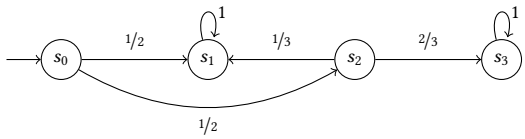


Figure 3.3: A simple MC

3.2.1 Measures

We define measures for reachability probabilities and for expected rewards.

3.2.1.1 Probabilities

·mass

A probability measure $Pr^D \colon \Pi^D_{\mathsf{fin}} \to [0,1]$ for finite paths $\pi = s_0 s_1 \dots s_n$ is given by the product of transition probabilities, referred to as the *mass* of the path:

$$Pr^D(\pi) := \prod_{i=0}^{n-1} P(s_i, s_{i+1})$$

The *unique probability measure* for infinite paths $Pr^D(\Pi^D_\infty)$ is defined by the usual cylinder set construction, see [BK08] for details.

reachability probability

Definition 3.6 (Reachability probability) Let D be an MC with target states T. We define the *reachability probability* $Pr_D(s \models \Diamond T)$ for reaching T from state s:

$$Pr_D(s \models \Diamond T) := \sum_{\pi \in \Pi^D(s, \Diamond T)} Pr^D(\pi),$$

·bounded

and the *bounded reachability probability* $Pr_D(s \models \Diamond^{\leq h} T)$ for reaching T from state s within $h \in \mathbb{N}$ steps:

$$Pr_D(s \models \Diamond^{\leq h} T) := \sum_{\pi \in \Pi^D(s, \Diamond^{\leq h} T)} Pr^D(\pi).$$

horizon

We refer to h as the *horizon*. We define the (bounded) reachability probability $Pr_D(\Diamond T)$ (resp. $Pr_D(\Diamond^{\leq h} T)$) for reaching T in D (in h steps) as reachability probability from the initial state, i.e.,

$$Pr_D(\Diamond T) := Pr_D(\iota \models \Diamond T) \quad \text{and} \quad Pr_D(\Diamond^{\leq h} T) := Pr_D(\iota \models \Diamond^{\leq h} T).$$

·has

Let $s \in S$ with $\kappa := Pr_D(s \models \Diamond T)$. We say that a state *has probability* κ to the target T.

Example 3.8 In the MC Figure 3.3, there are two paths to state s_1. They have probability $1/2$ and $1/6$ respectively, so the total probability to reach s_1 is $2/3$. In the MC Figure 3.2b, the probability to reach s_4 is

$$3/4 \cdot 1/2 + 3/4 \cdot 1/2 \cdot 3/4 \cdot 1/2 + 3/4 \cdot 1/2 \cdot 3/4 \cdot 1/2 \cdot 3/4 \cdot 1/2 + \ldots,$$

which simplifies to a geometric series:

$$\sum_{i=1}^{\infty} (3/8)^i = \frac{3/8}{1 - 3/8} = 3/5.$$

If we restrict the horizon to $h := 4$, then the result is $3/8 + 9/64$.
In the MC from Figure 3.2a, the probability to reach s_4 is the sum of

- $1/3$ (going there directly)
- $1/3 \cdot 1 \cdot 2/3$ (visiting s_5 once)
- $1/3 \cdot 1 \cdot (1/6 + 1/6) \cdot 2/3$ (visiting s_5 twice)
- ...

This infinite sum converges to $2/3$.

Remark 3.4 In line with the remainder of this thesis, we focus on unbounded reachability probabilities. Bounded reachability in MCs (and MDPs) may be reduced to unbounded reachability in a h-times larger MC by a technique called *unrolling* [AHK03]. For performance reasons, it is sometimes better to avoid this unrolling and present dedicated approaches.

Following Remark 3.3, the next result shows a strong duality of target and bad states.

Proposition 3.1 ([BK08, Thm. 10.27]) Let D be a finite MC, then for any state s:

$$Pr_D(s \models \Diamond \mathsf{BSCC}_D) = 1, \quad \text{and consequently } Pr_D(s \models \Diamond T) + Pr_D(s \models \Diamond B_{T,D}) = 1.$$

Example 3.9 Reconsider the probability to reach state s_4 in Figure 3.2a. The probability to reach state s_0 is trivially $1/3$. The only BSCCs are s_0 and s_4. We obtain $1 = 1/3 + Pr(\Diamond\{s_4\})$, and thus, the probability to reach s_4 is $2/3$.

Finally, the following lemma shows that instead of just making the target state absorbing, we may also declare a complete BSCC that contains a target state absorbing.

Lemma 3.2 Let $D = (S, \iota, P, \mathsf{rew})$ be a finite MC with BSCC S', then for all $s, s' \in S'$:

$$Pr_D(s \models \Diamond\{s'\}) = 1.$$

3.2.1.2 Rewards

We define the reward collected along a finite path π as the sum over the rewards at the visited states before reaching T:

$$\mathsf{colrew}(\pi) := \sum_{i=0}^{|\pi|-1} \mathsf{rew}(\pi@i)$$

expected reward

Definition 3.7 (Expected reward) Let D be an MC with target states T. Furthermore, let state $s \in S$. We define the *expected reward* $Er_D(s \models \Diamond T)$ as

$$Er_D(s \models \Diamond T) := \begin{cases} \sum_{\pi \in \Pi^D(s, \Diamond T)} Pr^D(\pi) \cdot \mathsf{colrew}(\pi) & \text{if } Pr_D(s \models \Diamond T) = 1, \\ +\infty & \text{otherwise.} \end{cases}$$

A special treatment of cases where $Pr_D(s \models \Diamond T) \neq 1$ is common in the literature (and necessary when considering more general cases), with typical choices being to set $Er_D(s \models \Diamond T)$ to 0 or to ∞. We deliberately choose to set the expected reward to $+\infty$ to avoid numerous case splits.

expected steps

Remark 3.5 (Expected steps) Let D be an MC with target states T. For rewards rew s.t. $\mathsf{rew}(s) = \mathsf{Ind}(s \in S \setminus T)$, we call the expected reward the *expected steps*. In this thesis, expected steps are mostly used for their compactness and intuitive meaning.

Example 3.10 In the MC Figure 3.3, let $\mathsf{rew}(s) = \{s_1, s_3 \mapsto 0, s_0 \mapsto 1, s_2 \mapsto 2\}$. The expected reward until reaching $\{s_1, s_3\}$ is $1 \cdot 1/2 + 3 \cdot 1/2 = 2$. The expected steps until reaching $\{s_1, s_3\}$ is $1/2 + 2 \cdot 1/2 = 3/2$. Consider the MC in Figure 3.2, but now with $\mathsf{rew}(s) = 2 \cdot \mathsf{Ind}(s = s_2)$. The expected rewards until reaching $\{s_0, s_4\}$ is

$$3/4 \cdot \left(2 \cdot 5/8 + 4 \cdot 25/64 + 6 \cdot 125/512 + \dots\right)$$

which we obtain by summing over all paths visiting s_2 once, twice, etc., and multiplying their mass by two times the visits to s_2. The sum simplifies to

$$3/4 \sum_{i=1}^{\infty} 2 \cdot i \cdot 5/8^i = 10/3.$$

The next proposition shows that computing reachability probabilities may be considered a special case of computing expected rewards.

Proposition 3.3 Let $D = (S, \iota, P, \mathsf{rew})$ be a finite MC with a set of target states T and for all $s \in S$, $\mathsf{rew}(s) = \mathsf{Ind}(s \in T)$. It holds that

$$Pr_D(\Diamond T) = Er_D(\Diamond \mathsf{BSCC}_D).$$

Remark 3.6 While this remark would justify focussing on expected rewards altogether, we typically consider probabilities as it simplifies the treatment considerably.

The following equations are obtained by unrolling the definition over paths:

Proposition 3.4 Let D be a finite MC with target states T and $Pr_D(\Diamond T) = 1$. Then:

$$Er_D(s \models \Diamond T) = \begin{cases} 0 & \text{if } s \in T, \\ \mathsf{rew}(s) + \sum_{s' \in S} P(s, \alpha, s') \cdot Er_D(s' \models \Diamond T) & \text{otherwise.} \end{cases}$$

Following Proposition 3.3, an analogue holds for reachability probabilities.

Example 3.11 Reconsider Example 3.10, in particular the MC in Figure 3.3 with $\mathsf{rew}(s) = \{s_1, s_3 \mapsto 0, s_0 \mapsto 1, s_2 \mapsto 2\}$. The expected reward until reaching $\{s_1, s_3\}$ in state s_0 is characterised by

$$1 + 1/2 \cdot Er_D(s_1 \models \Diamond T) + 1/2 \cdot Er_D(s_2 \models \Diamond T).$$

The probability until reaching $\{s_1\}$ is characterised by

$$1/2 \cdot Pr_D(s_1 \models \Diamond T) + 1/2 \cdot Pr_D(s_2 \models \Diamond T).$$

These linear equations are essential in computing measures efficiently, see Section 3.3.

3.2.1.3 Other measures

Reachability and expected rewards are fundamental traits of Markov chains, but definitely not the only measures. Many interesting problems are best described by means of measures like reward-bounded reachability, conditional reachability, bounded expected rewards, discounted rewards, mean-payoffs, and many more. However, none of these measures is discussed in detail within the scope of this thesis.

3.2.2 Properties

Properties evaluate MC by comparing some measure to a *threshold* $\lambda \in \mathbb{Q}_{\geq 0}$ using a comparison relation $\bowtie \in \{<, \leq, >, \geq\}$. We thus turn from the quantitative view to a Boolean view. For each property φ, we define the *satisfaction relation* $\models$:

threshold

satisfaction relation

Definition 3.8 (Properties) Let D be an MC with target states T, λ a threshold and $\bowtie$ a comparison relation. We define the following properties:

- *unbounded reachability* property $\mathbb{P}_{\bowtie\lambda}(\Diamond T)$:

  $$D \models \mathbb{P}_{\bowtie\lambda}(\Diamond T) \quad \text{iff} \quad Pr_D(\Diamond T) \bowtie \lambda.$$

·unbounded reachability

·bounded reachability

- *bounded reachability* property $\mathbb{P}_{\bowtie\lambda}(\Diamond^{\leq h}T)$:

$$D \models \mathbb{P}_{\bowtie\lambda}(\Diamond^{\leq h}T) \quad \text{iff} \quad Pr_D(\Diamond^{\leq h}T) \bowtie \lambda.$$

·expected reward

- *expected reward* property $\mathbb{E}_{\bowtie\lambda}(\Diamond T)$:

$$D \models \mathbb{E}_{\bowtie\lambda}(\Diamond T) \quad \text{iff} \quad Er_D(\Diamond T) \bowtie \lambda.$$

·complement

Furthermore, we define the *complement property* $\overline{\varphi}$ for a property φ by using the complementary comparison relation. Thus,

$$\overline{\bowtie} := \begin{cases} < & \text{if } \bowtie = \geq, \\ > & \text{if } \bowtie = \leq, \\ \leq & \text{if } \bowtie = >, \\ \geq & \text{if } \bowtie = <, \end{cases} \qquad \text{and } \overline{\varphi} := \begin{cases} \mathbb{P}_{\overline{\bowtie}\lambda}(\Diamond T) & \text{if } \varphi = \mathbb{P}_{\bowtie\lambda}(\Diamond T), \\ \mathbb{P}_{\overline{\bowtie}\lambda}(\Diamond^{\leq h}T) & \text{if } \varphi = \mathbb{P}_{\bowtie\lambda}(\Diamond^{\leq h}T), \\ \mathbb{E}_{\overline{\bowtie}\lambda}(\Diamond T) & \text{if } \varphi = \mathbb{E}_{\bowtie\lambda}(\Diamond T). \end{cases}$$

Proposition 3.5 An MC satisfies either φ or $\overline{\varphi}$. Thus, $D \models \varphi$ iff $D \not\models \overline{\varphi}$.

·upper bounded
·safety
·lower bounded
·liveness

- A property is *upper bounded*, if $\bowtie \in \{<, \leq\}$. Upper bounded properties are called *safety* properties.
- A property is *lower bounded*, if $\bowtie \in \{>, \geq\}$. Lower bounded properties are called *liveness* properties.

specification

A *specification* for a system consists of a set of properties Φ. An MC D satisfies the specification, written $D \models \Phi$, if $D \models \varphi$ for all $\varphi \in \Phi$.

Example 3.12 Reconsider the two MCs in Figure 3.2, both with target state $T = \{s_4\}$. Recall that the probabilities to reach the target are $2/3$ and $3/5$, respectively. We may want to reach the target state with probability at most $13/20$, i.e., $\varphi := \mathbb{P}_{\leq 13/20}(\Diamond T)$. We have that $M[\sigma_2] \models \varphi$ and $M[\sigma_3] \not\models \varphi$.

While measures and models are closely related, we loosen this relationship when turning to properties. Therefore, we have to define target states without a fixed model. Formally, we label states in MCs (and MDPs) with some atomic propositions, i.e. $L\colon S \longrightarrow 2^{AP}$, and then describe the target states as being the set of states that are labelled with a specific label $\mathsf{target} \in AP$, i.e., $T = \{s \in S \mid \mathsf{target} \in L(s)\}$. For details, see [BK08]. For our purposes, however, we relax the notation and denote target states T as subset of every considered MC.

Example 3.13 In Example 3.12, we used the same property for different sets of target states. Thus, strictly, we used T as a label rather than a set of states.

Remark 3.7 We sometimes depict target states together with the model by a double line as border, similar to, e.g., accepting states in deterministic finite automata.

While we only formally introduce reachability properties, these notions may be extended to richer logics like arbitrary PCTL [HJ94], PCTL* [ASB95], or ω-regular properties [Var99]. For these logics, reachability is considered an elementary property.

3.2.3 Models with nondeterminism

Measures | For models with nondeterminism, such as MDPs, the measures cannot be uniquely defined, as they depend on how the nondeterminism is resolved. Instead we use the measures on the induced MCs.

Definition 3.9 (Induced measures) Let M be an MDP with rewards, $\sigma \in \mathfrak{S}^M$ a strategy and h a horizon.

$$Pr_M^\sigma(s \models \Diamond T) := Pr_{M[\sigma]}(s \models \Diamond T),$$
$$Pr_M^\sigma(s \models \Diamond^{\leq h} T) := Pr_{M[\sigma]}(s \models \Diamond^{\leq h} T),$$
$$Er_M^\sigma(s \models \Diamond T) := Er_{M[\sigma]}(s \models \Diamond T).$$

Thus, for each strategy we may consider the induced measure. Typically, we are interested in the range of measures that may be obtained by picking a strategy. The following proposition ensures that a maximal and minimal measure exist and that these may be obtained by considering only memoryless deterministic strategies.

Proposition 3.6 ([BK08, Lem. 10.102ff]) Let M be a finite MDP and $s \in S$.

$$\inf_{\sigma \in \mathfrak{S}^M} Pr_M^\sigma(s \models \Diamond T) = \min_{\sigma \in \Sigma^M} Pr_M^\sigma(s \models \Diamond T),$$
$$\sup_{\sigma \in \mathfrak{S}^M} Pr_M^\sigma(s \models \Diamond T) = \max_{\sigma \in \Sigma^M} Pr_M^\sigma(s \models \Diamond T).$$

The proposition justifies the notation

$$Pr_M^{\min}(s \models \Diamond T) := \min_{\sigma \in \Sigma^M} Pr_M^\sigma(s \models \Diamond T), \text{ and } Pr_M^{\max}(s \models \Diamond T) := \max_{\sigma \in \Sigma^M} Pr_M^\sigma(s \models \Diamond T).$$

Example 3.14 For the MDP in Figure 3.1b with target state $T := \{s_4\}$, we have that

$$Pr_M^{\min}(\Diamond T) = 1/3 \text{ using } \sigma = \{s_1, s_5 \mapsto \alpha\},$$

and

$$Pr_M^{\max}(\Diamond T) = 2/3 \text{ using } \sigma = \{s_1 \mapsto \alpha, s_5 \mapsto \beta\}.$$

For the supremum on expected rewards, similar results require to ensure that the expected rewards are finite. In particular, it depends on the strategy whether the target states are reached with probability one.

Proposition 3.7 Let M be a finite MDP and $s \in S$.

$$\begin{aligned}
&\inf_{\sigma \in \mathfrak{S}^M} Er_M^\sigma(s \models \Diamond T) \\
&\quad = \begin{cases} \min_{\sigma \in \Sigma^M} Er_M^\sigma(s \models \Diamond T) & \text{if there exists } \sigma \in \Sigma^M \text{ s.t. } Pr_M^\sigma(s \models \Diamond T) = 1, \\ +\infty & \text{otherwise.} \end{cases} \\
&\sup_{\sigma \in \mathfrak{S}^M} Er_M^\sigma(s \models \Diamond T) \\
&\quad = \begin{cases} \max_{\sigma \in \Sigma^M} Er_M^\sigma(s \models \Diamond T) & \text{if for all } \sigma \in \Sigma^M\text{: } Pr_M^\sigma(s \models \Diamond T) = 1, \\ +\infty & \text{otherwise.} \end{cases}
\end{aligned}$$

We stress that there always is a memoryless deterministic strategy that obtains this optimum. Moreover, there is no unique optimum strategy. Let M be a finite MDP such that $\mathrm{opt}_{\sigma \in \Sigma^M} Er_M^\sigma(\Diamond T) < \infty$. We say that M has *finite optimal reward*. More precisely, we say M has finite minimal, or finite maximal reward respectively.

finite optimal reward

The following proposition clarifies that a strategy is optimal for a complete reachable fragment of the MDP, not just for a particular state.

Proposition 3.8 Let M be a finite MDP with finite optimal reward, and let $\sigma \in \{\arg\mathrm{opt}_{\sigma' \in \Sigma^M} Er_M^{\sigma'}(\Diamond T)\}$. Then:

$$\sigma \in \{\arg\mathrm{opt}_{\sigma' \in \Sigma^M} Er_M^{\sigma'}(s \models \Diamond T)\} \quad \text{for all } s \in \mathsf{reach}(S).$$

Following Proposition 3.3, the analogue holds for reachability probabilities.

·optimal

We call a strategy σ as in the proposition above *optimal* (w.r.t. a measure and target states T). If the MDP has no finite optimal reward, then we call any strategy which induces infinite reward optimal. To clarify whether we minimise or maximise, we call the strategies *minimal* or *maximal*, respectively.

Example 3.15 Consider the MDP from Figure 3.1b with target states $T := \{s_0, s_4\}$ and $\mathsf{rew} = \{s_1 \mapsto 1, s_2 \mapsto 2\}$. The MDP does not have finite maximal reward, as there is a strategy that prevents reaching state s_0 or s_4 (by having a loop in states s_3, s_5). However, we do have finite minimal reward.

Optimal strategies have the following intuitive characterisation using *Bellman-equalities*, which generalise the equation for rewards in MCs in Proposition 3.4.

Proposition 3.9 (Bellman's Characterisation) Let M be a finite MDP with finite optimal reward and σ an optimal strategy (w.r.t. expected rewards and T). Then:

$$Er_M^\sigma(s \models \Diamond T) = \begin{cases} 0 & \text{if } s \in T, \\ \mathsf{rew}(s) + \mathrm{opt}_{\alpha \in Act} \sum_{s' \in S} P(s, \alpha, s') \cdot Er_M^\sigma(s' \models \Diamond T) & \text{otherwise.} \end{cases}$$

Following Proposition 3.3, the analogue holds for reachability probabilities.

Example 3.16 Consider the MDP from Figure 3.1b with target states $T := \{s_0, s_4, s_5\}$ and $\mathsf{rew} = \{s_1 \mapsto 1, s_2 \mapsto 2\}$, we obtain as characterising equation for the maximal reward from s_1:

$$1 + \max\big(\frac{1}{3}Er_M(s_0 \models \Diamond T) + \frac{1}{3}Er_M(s_3 \models \Diamond T) + \frac{1}{3}Er_M(s_4 \models \Diamond T), \\ \frac{1}{4}Er_M(s_0 \models \Diamond T) + \frac{3}{4}Er_M(s_2 \models \Diamond T)\big).$$

In particular, the equations yield a dynamic programming formulation, *and* show that we may find an optimal strategy by optimising the actions at each state independently.

Properties | Finally, properties may be formulated for MDPs. In that case, we need a quantifier over the strategies.

Definition 3.10 (Satisfaction-relation for MDPs) Let M be an MDP and $\sigma \in \mathfrak{S}^M$. Let φ be a property as in Definition 3.8 on page 45.

$$M, \sigma \models \varphi \quad \text{iff} \quad M[\sigma] \models \varphi.$$

Furthermore:

$$M \exists\!\models \varphi \quad \text{iff} \quad \text{exists } \sigma \in \mathfrak{S}^M \text{ such that } M[\sigma] \models \varphi, \quad \text{and}$$
$$M \forall\!\models \varphi \quad \text{iff} \quad \text{for all } \sigma \in \mathfrak{S}^M \text{ it holds that } M[\sigma] \models \varphi.$$

While in this definition we quantify over all strategies, the following proposition reminds us that it suffices to consider only the deterministic memoryless strategies.

Proposition 3.10 Let M be a finite MDP, $\sigma_{\max}$ ($\sigma_{\min}$) a maximal (minimal) strategy for expected rewards to T. Let $\varphi := \mathbb{E}_{\leq\lambda}(\Diamond T)$.

$$M \exists\!\models \varphi \quad \text{iff} \quad Er_M^{\sigma_{\min}}(\Diamond T) \leq \lambda, \text{ and}$$
$$M \forall\!\models \varphi \quad \text{iff} \quad Er_M^{\sigma_{\max}}(\Diamond T) \leq \lambda.$$

Analogues hold for all comparison relations and for reachability probabilities.

Example 3.17 Using Example 3.14, we have that

$$M \exists\models \mathbb{P}_{\leq 1/2}(\Diamond T), \text{ but } M \forall\not\models \mathbb{P}_{\leq 1/2}(\Diamond T), \text{ also } M \exists\not\models \mathbb{P}_{< 1/4}(\Diamond T), M \forall\models \mathbb{P}_{\leq 7/10}(\Diamond T).$$

Equivalence | Throughout this thesis, we do locally alter and reconstruct various MDPs. We use a lazy notion of *equivalence* on MDPs to formalise that these transformations are sound.

·equivalence

Definition 3.11 (Equivalence) Let M, M' be MDPs with state sets S, S' and target states $T \subseteq S$, $T' \subseteq S'$. We say that $M \simeq_{T,T'} M'$, if

- for every strategy $\sigma \in \Sigma^M$, there exists a strategy $\sigma' \in \Sigma^{M'}$ such that:
$$Pr_M^{\sigma}(\Diamond T) = Pr_M^{\sigma'}(\Diamond T') \quad \text{and} \quad Er_M^{\sigma}(\Diamond T) = Er_M^{\sigma'}(\Diamond T').$$
- for every strategy $\sigma' \in \Sigma^{M'}$, there exists a strategy $\sigma \in \Sigma^M$ such that:
$$Pr_M^{\sigma}(\Diamond T) = Pr_M^{\sigma'}(\Diamond T') \quad \text{and} \quad Er_M^{\sigma}(\Diamond T) = Er_M^{\sigma'}(\Diamond T').$$

We omit the subscript if the target states are clear from the context.

Remark 3.8 We do not compute such equivalences, but use them to formalise the soundness of operations that merge or split states. Notions like (weak) probabilistic bisimilarity are super-seeded by this notion of equivalence.

3.3 Analysing Markov Models

This section recaps how to compute the measures on MCs and MDPs introduced before. We only consider finite Markov models.

3.3.1 Qualitative analysis

Qualitative analysis refers to finding states reaching the target with probability 0 or 1.

3.3.1.1 Markov chains

For any MC D with target states T, the state space may be partitioned into

$$S = S_{=0}^T \uplus S_?^T \uplus S_{=1}^T, \text{ with}$$

zero-states

- *zero-states* $S_{=0}^T$ are states which have probability zero to reach the target states T:
$$S_{=0}^T = \{s \in S \mid Pr_D(s \models \Diamond T) = 0\},$$

- *one-states* $S^T_{=1}$ are states which have probability one to reach the target states T: one-states

$$S^T_{=1} = \{s \in S \mid Pr_D(s \models \Diamond T) = 1\},$$

- *quant-states* $S^T_{?}$ are all other states[1], i.e., states with a probability to reach the target strictly between zero and one: quant-states

$$S^T_{?} = \{s \in S \mid 0 < Pr_D(s \models \Diamond T) < 1\}.$$

The partitioning may be computed by simple graph-algorithms, see [BK08, Sect 10.1.2]. The following proposition gives an important characterisation of the zero-states and gives a characterisation for the one-states that uses the duality of target and bad states.

Proposition 3.11 Let D be a finite MC with target states T.

$$S^T_{=0} = \{s \in S \mid \Pi^D(s, \Diamond T) = \emptyset\}, \quad \text{and} \quad S^T_{=1} = S^{B_T}_{=0}.$$

Example 3.18 Reconsider the MC in Figure 3.2a with target state s_4. The state s_4, but also the states s_3 and s_5 are one-states. The state s_0 is a zero-state.

3.3.1.2 MDPs

- *exist-zero states* $S^T_{=\exists 0}$ are the set of states for which there exists a strategy σ such that the induced probability to reach the target states T is zero: exist-zero states

$$S^T_{=\exists 0} = \{s \in S \mid \text{exists } \sigma \text{ such that } Pr^\sigma_M(s \models \Diamond T) = 0\},$$

- *forall-zero states* $S^T_{=\forall 0}$ are the set of states for which for all strategies σ the induced probability to reach the target states T is zero: forall-zero states

$$S^T_{=\forall 0} = \{s \in S \mid \text{for all } \sigma : Pr^\sigma_M(s \models \Diamond T) = 0\},$$

- and similarly exist-one and forall-one states.

All these states may be computed using polynomial-time graph algorithms, which are discussed in detail in [BK08, p. 858ff].

Example 3.19 We consider the MDP in Figure 3.1b with target states $T = \{s_4\}$. Then s_4 is trivially a forall-one-state and s_0 is trivially a forall-zero state. Forall-zero and forall-one states are also exists-zero and exists-one states, respectively. Additionally, the states s_3 and s_5 are exists-one and exists-zero states.

[1] The notation stems from model checking procedures in which the zero-states and one-states are eliminated first. The remaining states then have an unknown probability, hence the question mark.

3.3.2 Quantitative analysis

We now turn our attention to the quantitative analysis, in which quantities such as probabilities or rewards have to be computed. It no longer suffices to consider the graph-structure.

3.3.2.1 Markov chains

To compute $Pr_D(s \models \Diamond T)$, the following characteristic linear equation system , based on an unrolling of the sum-over-paths definition, see Proposition 3.4.

Constraints 3.1 Let D be a finite MC. Consider variables $\{p_s \mid s \in S\}$:

$$\begin{aligned} p_s &\doteq 1 && \text{for all } s \in T, \\ p_s &\doteq 0 && \text{for all } s \in S^T_{=0}, \\ p_s &\doteq \sum_{s' \in S} P(s, s') \cdot p_{s'} && \text{for all } s \in S \setminus \left(T \cup S^T_{=0}\right). \end{aligned}$$

Proposition 3.12 Let D be a finite MC. Constraints 3.1 has a unique solution val. For every state s,

$$\mathsf{val}(p_s) = Pr_D(s \models \Diamond T).$$

Example 3.20 We consider the MC in Figure 3.2b with target state s_4. We omit unreachable states. We obtain the following equations using p_i to denote p_{s_i}.

$$p_0 \doteq 0, \quad p_4 \doteq 1, \quad p_1 \doteq 1/4 \cdot p_0 + 3/4 \cdot p_2, \quad p_2 = 1/2 \cdot p_1 + 1/2 \cdot p_4.$$

A similar constraint system may be constructed for expected rewards, again assuming that $Pr_D(s \models \Diamond T) = 1$:

Constraints 3.2 Consider variables $\{e_s \mid s \in S\}$:

$$\begin{aligned} e_s &\doteq 0 && \text{for all } s \in T, \\ e_s &\doteq \mathsf{rew}(s) + \sum_{s' \in S} P(s, s') \cdot e_{s'} && \text{for all } s \in S \setminus T. \end{aligned}$$

Example 3.21 We consider the MC in Figure 3.2b with target states $\{s_0, s_4\}$ and reward $\mathsf{rew}(s) := \mathsf{Ind}(s = s_2)$. We omit unreachable states. We obtain the following equations using e_i to denote e_{s_i}:

$$e_0 \doteq 0, \quad e_4 \doteq 0, \quad p_1 \doteq 1/4 \cdot e_0 + 3/4 \cdot e_2, \quad e_2 \doteq 1 + 1/2 \cdot e_1 + 1/2 \cdot e_4.$$

3.3.2.2 MDPs

We first consider the computation of probabilities under minimising strategies. Therefore, we generalise the linear equation system to a system of linear inequalities.

Constraints 3.3 Let M be a finite MDP. Consider variables $\{p_s \mid s \in S\}$:

$$\begin{aligned}
&\text{minimise } p_\iota \text{ s.t.} \\
&p_s \doteq 1 && \text{for all } s \in T, \\
&p_s \doteq 0 && \text{for all } s \in S^T_{=\exists 0}, \\
&p_s \geqslant \sum_{s' \in S} P(s, \alpha, s') \cdot p_{s'} && \text{for all } s \in S \setminus (S^T_{=\exists 0} \cup T), \alpha \in Act.
\end{aligned}$$

Proposition 3.13 Let M be a finite MDP. Constraints 3.3 have a unique solution val. For every state $s \in S$,

$$\mathsf{val}(s) = Pr^{\min}_M(s \models \Diamond T).$$

Example 3.22 We consider the MDP in Figure 3.1b with target states s_4, and denote p_{s_i} with p_i. We are interested in the minimal probability to reach the target. Recall also the exist-zero states from Example 3.19.

$$\begin{aligned}
&\text{minimise } p_1 \text{ s.t.} \\
&p_4 \doteq 1, \quad p_0 \doteq 0, \quad p_3 \doteq 0, \quad p_5 \doteq 0, \\
&p_1 \geqslant \frac{1}{3}p_0 + \frac{1}{3}p_3 + \frac{1}{3} \cdot p_4, \quad p_1 \geqslant \frac{1}{4}p_0 + \frac{3}{4}p_2, \quad p_2 \geqslant \frac{1}{2}p_1 + \frac{1}{2}p_4.
\end{aligned}$$

If we aim to decide $M \not\models \mathbb{P}_{\leq\lambda}(\Diamond T)$, then we may utilise the following constraints.

Constraints 3.4 Let M be a finite MDP with property $\mathbb{P}_{\leq\lambda}(\Diamond T)$: Consider variables $\{p_s \mid s \in S\}$:

$$\begin{aligned}
&p_\iota \leqslant \lambda \\
&p_s \doteq 1 && \text{for all } s \in T, \\
&p_s \doteq 0 && \text{for all } s \in S^T_{=\exists 0}, \\
&p_s \geqslant \sum_{s' \in S} P(s, \alpha, s') \cdot p_{s'} && \text{for all } s \in S \setminus (S^T_{=\exists 0} \cup T), \alpha \in Act.
\end{aligned}$$

Proposition 3.14 Let M be a finite MDP.

$$\text{Constraints 3.4 has a solution} \quad \text{iff} \quad M \forall\models \mathbb{P}_{\leq\lambda}(\Diamond T).$$

Probabilities under maximising strategies may be computed analogously, using the following optimisation problem.

Constraints 3.5 Let M be a finite MDP. Consider variables $\{p_s \mid s \in S\}$:

$$\begin{aligned}
&\text{maximise } p_\iota \quad \text{s.t.} \\
&p_s \doteq 1 && \text{for all } s \in T, \\
&p_s \doteq 0 && \text{for all } s \in S^T_{=\forall 0}, \\
&p_s \leqslant \sum_{s' \in S} P(s, \alpha, s') \cdot p_{s'} && \text{for all } s \in S \setminus (S^T_{=\forall 0} \cup T), \alpha \in Act.
\end{aligned}$$

Propositions 3.13 and 3.14 have analogues for maximal strategies and the remaining bound/quantification combinations. Moreover, there are straightforward analogues for the equation systems for expected rewards. All these are omitted here for conciseness.

Example 3.23 We consider the MDP M in Figure 3.1b with property $\varphi := \mathbb{P}_{\geq\lambda}(\Diamond\{s_4\})$. We are interested in determining whether $M \forall\models \varphi$ and recall the exist-one states from Example 3.19. We denote p_{s_i} with p_i and obtain:

$$\begin{aligned}
&p_1 \geq \lambda, \quad p_4 \doteq 1, \quad p_0 \doteq 0, \quad p_3 \doteq 1, \quad p_5 \doteq 1, \\
&p_1 \leqslant \frac{1}{3}p_0 + \frac{1}{3}p_3 + \frac{1}{3}p_4, \quad p_1 \leqslant \frac{1}{4}p_0 + \frac{3}{4}p_2, \quad p_2 \leqslant \frac{1}{2}p_1 + \frac{1}{2}p_4.
\end{aligned}$$

Empirically, it is often more beneficial to apply methods such as *value iteration* or *policy iteration* [BK08], even if these methods have an exponential worst-case runtime.

3.4 Stochastic Games

Stochastic games [Con90] may be seen as an extension to MDPs. We briefly sketch them as they are mentioned in some chapters—details are outside the scope of this thesis.

SG

Definition 3.12 (Stochastic game) An MDP with a state space $S = S_{\mathrm{I}} \uplus S_{\mathrm{II}}$ is a *stochastic game* (*SG*).

Thus, an SG is an MDP in which the states are partitioned, i.e., each state is assigned to either player I or to player II.

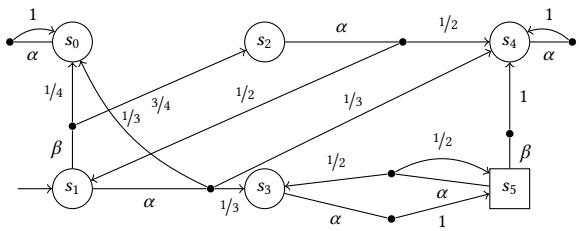


Figure 3.4: A stochastic game

Example 3.24 We depict an SG in Figure 3.4. Boxes reflect player II states.

A strategy for an SG may then be partitioned into a player-I and a player II strategy.

Definition 3.13 (Strategy) A *strategy* for an SG G is a tuple $(\sigma_{\mathrm{I}}, \sigma_{\mathrm{II}})$ with (measurable) functions strategy

$$\sigma_{\mathrm{I}} \colon \Pi^{\mathrm{I}}_{\mathsf{fin}} \to \mathit{Distr}(\mathit{Act}) \text{ such that } \mathit{supp}(\sigma_{\mathrm{I}}(\pi)) \subseteq \mathbb{A}\mathrm{ct}(\mathsf{last}(\pi)) \text{ for all } \pi \in \Pi^{\mathrm{I}}_{\mathsf{fin}},$$
$$\sigma_{\mathrm{II}} \colon \Pi^{\mathrm{II}}_{\mathsf{fin}} \to \mathit{Distr}(\mathit{Act}) \text{ such that } \mathit{supp}(\sigma_{\mathrm{II}}(\pi)) \subseteq \mathbb{A}\mathrm{ct}(\mathsf{last}(\pi)) \text{ for all } \pi \in \Pi^{\mathrm{II}}_{\mathsf{fin}},$$

where $\Pi^{\mathrm{I}}_{\mathsf{fin}} := \{\pi \in \Pi^{G}_{\mathsf{fin}} \mid \mathsf{last}(\pi) \in S_{\mathrm{I}}\}$ and $\Pi^{\mathrm{II}}_{\mathsf{fin}} := \{\pi \in \Pi^{G}_{\mathsf{fin}} \mid \mathsf{last}(\pi) \in S_{\mathrm{II}}\}$.

The notions of memoryless and deterministic carry over from MDP strategies to player i strategies. In particular, for some SG G, we denote deterministic memoryless strategies for player I and player II as Σ^{G}_{I} and Σ^{G}_{II}, respectively.

We lift the *satisfaction relation* for MDPs to SGs. satisfaction relation

Definition 3.14 (Satisfaction relation on SGs) For an SG G, property φ, and quantifiers Q_1, Q_2, we define $G \models^{Q_1, Q_2} \varphi$ as:

$$Q_1 \sigma_{\mathrm{I}} \in \Sigma^{G}_{\mathrm{I}}\ Q_2 \sigma_{\mathrm{II}} \in \Sigma^{G}_{\mathrm{II}} : \quad G[\sigma_{\mathrm{I}}, \sigma_{\mathrm{II}}] \models \varphi.$$

For these games, the order of players does not influence the outcome [Con90].

Example 3.25 Consider the SG in Figure 3.4 and a property for reaching state s_4 with probability at least $13/20$, i.e. $\varphi := \mathbb{P}_{\geq 13/20}(\Diamond T)$. We have that $G \models^{\exists,\exists} \varphi$ by $(\{s_1 \mapsto \alpha\}, \{s_5 \mapsto \beta\})$. However, we have $G \not\models^{\exists,\forall} \varphi$, as player II may take α in s_5 and prevents reaching s_4 from s_5. Likewise, $G \not\models^{\forall,\exists} \varphi$: if player I selects β in state s_1, then no matter what player II does, the probability to reach s_4 is $3/5$.

Tool support for analysing SGs is mostly based on value iteration.

3.5 Modelling Languages

MCs and MDPs are typically large, consisting of thousands, millions or even more states. Therefore, they are typically specified in a concise modelling language. While their semantics are beyond the scope of this thesis, we briefly introduce the PRISM-language as an example of such a language. Besides PRISM introduced below, *Modest* is a powerful language based on a process algebra that is well-suited to describe all sorts of protocols, *pgcl* is an imperative language which coin-flips as primitives, and *Jani* [7] is a powerful intermediate language to increase tool-support for a variety of further domain specific languages. Models in these languages are available from the quantitative verification benchmark set [HKPQ+19].

Modest

pgcl

Jani

PRISM | We briefly recap how the model-based concepts translate to language concepts in the *PRISM* guarded-command language. A PRISM program consists of one or more reactive modules that may interact with each other. We consider a single *module*. This simplification is not a restriction, every program may be flattened into this form, which severely impacts the performance of model checkers, but simplifies the explanation. A module has a set of bounded variables, the valuations of these variables span the state space of the underlying model. Transitions between states are described by guarded *commands* of the form:

PRISM

·module

·command

$$\texttt{guard} \quad \longrightarrow \quad p_1 : \texttt{update}_1 + \ldots\ldots + p_n : \texttt{update}_n$$

The *guard* is a Boolean expression over the module's variables. If the guard evaluates to `true` for some state (a valuation), then the module evolves into one of the n successor states by updating its variables. An *update* is chosen according to the probability distribution given by the expressions $p_1, \ldots, p_n$. In every state enabling the guard, the evaluation of $p_1, \ldots, p_n$ must sum up to one. The operation fixdl takes a program and adds commands that introduce self-loops for states without enabled guard: we say that the operation *fixes deadlocks*. Overlapping guards yield nondeterminism. When modelling MCs, they are disallowed[1]. Roughly, a *program* $\mathcal{L}$ thus is a tuple $(\mathsf{Var}, \mathsf{Cmd})$ of variables and commands. For a program $\mathcal{L}$, the *underlying MC* $[\![\, \mathcal{L} \,]\!]$ are $\mathcal{L}$'s semantics. We lift specifications: Program $\mathcal{L}$ satisfies a specification Φ, iff $[\![\, \mathcal{L} \,]\!] \models \Phi$, etc.

·guard

·update

·fixdl

program

underlying MC

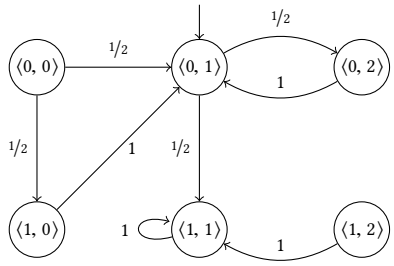
Example 3.26 In Figure 3.5a, we give a PRISM program for an MC. The program contains two parameters x and y, where x is either zero or one, and y between zero and two. In total, there are thus 6 different states. We denote states as tuples with the x- and y-value. We depict the MC (after fixing deadlocks) in Figure 3.5b. From state $\langle 0, 0 \rangle$, (only) the first guard is enabled and thus there are two transitions,

[1] Outside of this thesis, overlapping guards are sometimes permitted when specifying MCs

```
module rex
x : [0..1] init 0;
y : [0..2] init 1;

x = 0 & y < 2 -> 0.5: x'=1 + 0.5: y'=y+1;
y = 2 -> 1: y'=y-1;
x = 1 & y != 1 -> 1: x'=y & y'=x;
endmodule
```

(a) PRISM program



(b) Underlying model after fixing deadlocks.

Figure 3.5: PRISM program $\mathcal{L}$ with underlying model $[\![\, \mathsf{fixdl}(\mathcal{L}) \,]\!]$.

each with probability a half: one in which x becomes one and one in which y is increased by one. We see a similar pattern for state $\langle 0, 1\rangle$, which also enables the first command. There is no guard enabled in state $\langle 1, 1\rangle$, therefore the fixdl operation adds a command to the program that produces a self-loop.

Part II

Understanding Parameter Synthesis

4 Parametric Probabilities in MDPs

Modelling uncertainty about the model

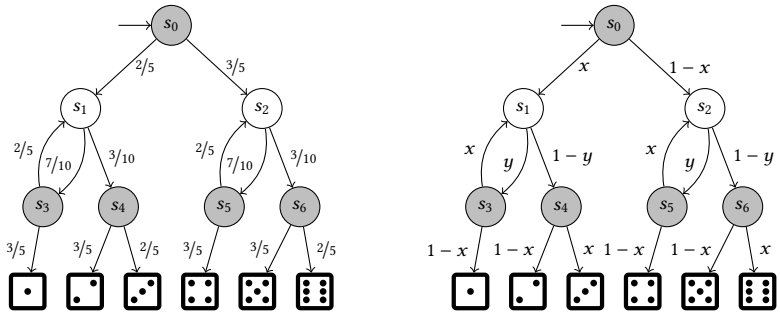
Summary | We introduce a parameterised version of MDPs (pMDPs), in which the probabilities are not fixed, but rather unknown or imprecise. We represent such uncertainties in the probabilities by (expressions over) parameters. This chapter formalises pMDPs and relates them to the parameter-free MDPs by instantiating the parameters with concrete valuations. We then naturally lift measures and properties from MDPs to pMDPs and valuations, and introduce a notion of regions to describe sets of valuations, e.g., sets of valuations complying with these properties. These ingredients allow us to formulate a selection of problem statements. Most fundamental is feasibility, i.e., the existence of parameter values such that the instantiated model satisfies the given properties. In the last part of this chapter, we show how this feasibility problem is encoded as a formula in the existential theory of the reals.

Origins | The presentation of this chapter is mostly based on [17]. The focus on feasibility is more in line with [19], which also included many of the ETR encodings.

Background | This chapter introduces the parametric version of Markov chains and MDPs, cf. Section 3.1, using rational functions and polynomials, cf. Section 2.1. The problem statements relate to (unbounded) reachability properties, as introduced in Section 3.2. As this chapter is foundational in its nature, some notions might be hard to fully grasp on a first read. Such notions are typically accompanied by a forward reference that will give more context. On the other hand, a more in-depth treatment is deferred to Chapter 5 whenever possible, keeping this first chapter accessible to first time readers.

4.1 Parametric Markov Models

We start fixing the formal model of parametric MDPs and parametric MCs. Therefore, we have a closer look at the parametric version of the Knuth-Yao die, as in Figure 1.4b.



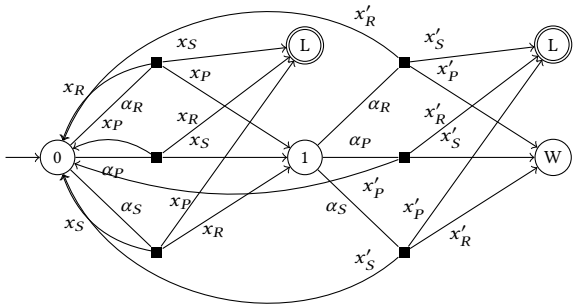
(a) with biased coins (b) with parametric transitions

Figure 4.1: Two variants of the Knuth-Yao die

Example 4.1 Recall from Example 3.2 that the Knuth-Yao algorithm [KY76] uses repeated coin flips to model a six-sided die. It uses a fair coin to obtain each possible outcome ('one', 'two', ..., 'six') with probability $1/6$. Figure 4.1a depicts an MC of a variant in which two *unfair* coins are flipped in an alternating fashion. Flipping the coins yields *heads* with probability $2/5$ (gray states) or $7/10$ (white states), respectively. Accordingly, the probability of *tails* is $3/5$ and $3/10$, respectively. The event of throwing a 'two' corresponds to reaching the state ⚁ in the MC. Assume now a specification consisting of the property requiring the probability to obtain 'two' to be larger than $3/20$. Knuth-Yao's original algorithm satisfies this property as using a fair coin results in $1/6$ as probability to reach ⚁. The biased model however, does not satisfy the property; in fact, ⚁ is reached with probability $1/10$. We may now ask ourselves: how unfair may these coins be and still satisfy the property? Vice versa, we can ask: Assuming that the probability to throw *heads* is between $2/5$ and $3/5$, does the property hold for all admissible probabilities of throwing *heads*?

Thus, when analysing parameterised MCs, we consider temporal properties on all, often uncountably many, instantiations. In the example above, we asked whether all 'almost fair' coins satisfy a temporal property (reaching ⚁).

Example 4.2 We consider a slightly modified variant of rock-paper-scissors, in which we play against Eve. Eve selects behind her back either Rock (R), Paper (P) or Scissors (S). Now, without knowing what she selected, we have to make a (randomised) decision. Then, we win the round as usual, in accordance with Figure 4.2b. We are interested in minimising the probability that we ever loose



(a) The parametric MDP for modified rock-paper-scissors.

		We play		
		Rock	Paper	Scissors
	Rock	draw	we win	we loose
Eve plays	Paper	we loose	draw	we win
	Scissors	we win	we loose	draw

(b) Winning a round of rock-paper-scissors

Figure 4.2: Playing (a slightly modified) rock-paper-scissors.

a round. Luckily, we thus either (1) win by playing (draws) forever, or (2) Eve gives up if we win twice in a row. We model this protocol by the pMDP in Figure 4.2a. Initially, Eve selects either (R), (P) or (S). Then, we select with some probability x_P *paper* , x_R *rock*, and x_S *scissors*. This probability is independent of the selection of Eve. If we loose, we go to a target state L (duplicated to avoid clutter). Otherwise, if we win, we move to the next round, and if we draw, we restart the game. In the second round, we may choose a different distribution over the actions. The distribution is represented by x'_P, x'_R, x'_S. After two wins, we reach the sink state W.
We want to know how we should play, i.e., how we should choose the actions to avoid reaching L. On a technical level, this questions corresponds to asking for the right values for x_R, x_P, x_S. In the worst-case, Eve has some transcendental powers and always counters optimally. What is the best strategy for us, i.e., what is the minimal probability of reaching the target state L. And how may we randomise if we merely want to ensure that we reach L with a probability less than 90%? Clearly, we have to randomise, otherwise Eve will easily counter our moves.

The example above is inspired by a connection to playing concurrent stochastic games that we explore in Section 7.2. In that domain, playing rock-paper-scissors is a

natural example, cf. [KNPS18]. Much alike, to the Knuth-Yao die example, we consider temporal properties on all instantiations. However, here we are primarily interested in finding some suitable instantiations among the uncountably many, rather than making a statement over all instantiations. Furthermore, we see that in the first example, the parameters were considered uncontrollable, whereas in the second example, we can control the parameter values. Chapter 7 contains further examples.

Next, we formalise parametric MDPs and their instantiations, the relation to properties, a notion of sets of instantiations, and then formalise problem statements.

4.1.1 Fundamentals

Parametric MDPs extend finite (classical, parameter-free) MDPs from Definition 3.1 with a set X of parameters and an adapted transition relation: Probabilities are no longer expressed by values from $[0, 1]$, but by rational functions over X[1].

pMDP

Definition 4.1 (pMDP) A *parametric Markov Decision Process* (*pMDP*) $\mathcal{M}$ is a tuple $(S, \iota, Act, X, \mathcal{P}, \mathsf{prew})$ with a finite set S of *states*, an initial state $\iota \in S$, a finite set Act of *actions*, a finite set X of *parameters*, a parametric probabilistic transition $\mathcal{P} \colon S \times Act \times S \to \mathbb{Q}(X)$, and a reward function $\mathsf{prew} \colon S \to \mathbb{Q}(X)$.

As pMDPs extend (finite) MDPs, the concepts of paths, graphs, and strategies as in Section 3.1.2 carry over naturally. Likewise, parametric MCs are obtained as an extension to MCs, and are a special case of pMDPs.

pMC

Definition 4.2 (pMC) A *parametric Markov chain* (*pMC*) $\mathcal{D}$ is a tuple $(S, \iota, X, \mathcal{P}, \mathsf{prew})$ with a finite set S of *states*, an initial state $\iota \in S$, a finite set X of *parameters*, a parametric probabilistic transition $\mathcal{P} \colon S \times S \to \mathbb{Q}(X)$, and a reward function $\mathsf{prew} \colon S \to \mathbb{Q}(X)$.

Just like for their parameter-free counterparts, we may omit the reward function whenever all states are awarded reward zero. Our definition of pMCs is based on[2] the definitions in [Daw04; LMT07] and the pMDP definition originates from [HHZ11a].

Example 4.3 Figure 4.1b depicts a parametric version of the biased Knuth-Yao die from Example 4.1. It has parameters $X = \{x, y\}$, where x is the probability of outcome *heads* in grey states and y the same for white states. The probability for *tails* is then $1-x$ and $1-y$, respectively. Figure 4.2a depicts the pMDP for our rock-paper-scissors variant. The parameters are $X = \{x_R, x_P, x_S, x'_R, x'_P, x'_S\}$.

Parameters are thus variables that may be substituted by concrete values. Not all

[1] For technical reasons, we exclude non-rational probabilities which are allowed in MDPs.
[2] We discuss related work centrally in Section 1.2.

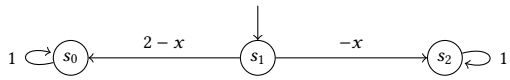


Figure 4.3: an unrealisable pMC

valuations are meaningful in the context of pMDPs. We are only interested in valuations for pMDPs that yield MDPs. Such *valuations* are called well-defined.

valuation

Definition 4.3 (Well-defined valuation) Let $\mathcal{M}$ be a pMDP with parameters X. A valuation val of X is *well-defined* for $\mathcal{M}$ if:

·well-defined

- probabilities are non-negative, i.e., $\mathcal{P}(s, \alpha, s')[\mathsf{val}] \geq 0$ for all $s, s' \in S, \alpha \in \mathbb{A}\mathrm{ct}(s)$.
- outgoing probabilities induce distributions, i.e., $\sum_{s' \in S} \mathcal{P}(s, \alpha, s')[\mathsf{val}] = 1$ for all $s \in S, \alpha \in \mathbb{A}\mathrm{ct}(s)$.
- rewards are non-negative, i.e., $\mathsf{prew}(s)[\mathsf{val}] \geq 0$ for all $s \in S$.

The set $\mathsf{Val}^{\mathrm{wd}}_{\mathcal{M}}$ consists of all well-defined valuations for $\mathcal{M}$.

We call valuations of X that are well-defined valuations for $\mathcal{M}$ *valuations for* $\mathcal{M}$.

Example 4.4 The well-defined valuations for the pMC in Figure 4.1b are

$$\mathsf{Val}^{\mathrm{wd}}_{\mathcal{D}} = \{\mathsf{val} \in \mathsf{Val} \mid 0 \leq \mathsf{val}(x), \mathsf{val}(y) \leq 1\}.$$

A valuation $\{x \mapsto 2, y \mapsto 0\}$ is not well-defined, as $1 - x[\mathsf{val}]$ is negative. By construction, the outgoing probabilities induce distributions in this example.
The well-defined valuations for the pMDP in Figure 4.2a are

$$\begin{aligned}\mathsf{Val}^{\mathrm{wd}}_{\mathcal{M}} = \{\mathsf{val} \in \mathsf{Val} \mid\ & \mathsf{val}(x_P) + \mathsf{val}(x_R) + \mathsf{val}(x_S) = 1,\\ & \mathsf{val}(x_P), \mathsf{val}(x_R), \mathsf{val}(x_S) \geq 0\\ & \mathsf{val}(x'_P) + \mathsf{val}(x'_R) + \mathsf{val}(x'_S) = 1,\\ & \mathsf{val}(x'_P), \mathsf{val}(x'_R), \mathsf{val}(x'_S) \geq 0\}.\end{aligned}$$

A valuation $\{x_R, x_P, x_S, x'_R, x'_P, x'_S \mapsto 2/3\}$ is not well-defined, as the sum of $\mathsf{val}(x_R), \mathsf{val}(x_P), \mathsf{val}(x_S)$ exceeds one.
Some pMDPs do not have any well-defined valuation. Consider, e.g., the pMC in Figure 4.3. Due to the transition to s_2, $-x$ must be non-negative, so x must be non-positive. Due to the transitions to s_0 and s_2, $2 - 2x$ must sum to 1 to induce a proper distribution. However, then x must be at least one, which contradicts that x must be non-positive.

·realisable

Definition 4.4 (Realisable) A pMDP $\mathcal{M}$ is *realisable*, if $\mathsf{Val}^{\mathrm{wd}}_{\mathcal{M}} \neq \emptyset$.

We generally assume that the sets of well-defined instantiations are not empty and unrealisable pMDPs are considered a modelling error. In Section 4.3.1, we see how to verify whether a pMDP is realisable. Realisable pMDPs are a template for parameter-free MDPs: we may replace parameters by concrete values to obtain MDPs. We call the process of replacing parameters instantiating a pMDP.

instantiation

Definition 4.5 (Instantiation) Let $\mathcal{M}$ be a pMDP and $\mathsf{val} \in \mathsf{Val}^{\mathrm{wd}}_{\mathcal{M}}$ a valuation. The *instantiation* of $\mathcal{M}$ with val is the MDP $\mathcal{M}[\mathsf{val}] := (S, \iota, Act, P, \mathsf{rew})$ with

$$P(s, \alpha, s') := \mathcal{P}(s, \alpha, s')[\mathsf{val}] \text{ and } \mathsf{rew}(s) := \mathsf{prew}(s)[\mathsf{val}] \quad \text{for all } s, s' \in S, \alpha \in \mathbb{A}\mathrm{ct}(s).$$

Example 4.5 Reconsider the pMC in Figure 4.1b. The valuation val with $\mathsf{val}(x) = 2/5$ and $\mathsf{val}(y) = 7/10$ is well-defined and yields the MC in Figure 4.1a.

generator

It is helpful and natural to consider a pMDP $\mathcal{M}$ as a *generator* for an, in general, uncountable set $\langle \mathcal{M} \rangle$ of instantiated MDPs.

Definition 4.6 (Generator) The generator of pMDP $\mathcal{M}$ is the set

$$\langle \mathcal{M} \rangle := \{\mathcal{M}[\mathsf{val}] \mid \mathsf{val} \in \mathsf{Val}^{\mathrm{wd}}_{\mathcal{M}}\}.$$

Let $R \subseteq \mathsf{Val}^{\mathrm{wd}}_{\mathcal{M}}$ be a set of valuations for $\mathcal{M}$. We define the set

$$\langle \mathcal{M} \mid R \rangle := \{\mathcal{M}[\mathsf{val}] \mid \mathsf{val} \in R\}.$$

Thus, $\langle \mathcal{M} \rangle = \langle \mathcal{M} \mid \mathsf{Val}^{\mathrm{wd}}_{\mathcal{M}} \rangle$. For this set, and sets yet to be introduced, we often omit everything but the superscript and write, e.g., $\langle \mathcal{M} \mid \mathrm{wd} \rangle$.

We may concretise pMDPs to MDPs by instantiating the parameters, and MDPs to MCs by replacing nondeterminism. One may also obtain pMCs from pMDPs and then concretise the pMCs to MCs. The result will be the same. Therefore, consider that the notion of strategies (Definition 3.4 on page 39) carries over from MDPs to pMDPs and thus induced MCs (Definition 3.5 and Remark 3.2) carry over to induced pMCs. Strategies from pMDPs can be applied to their instantiations, formally:

Proposition 4.1 Let $\mathcal{M}$ be a pMDP with some strategy $\sigma \in \mathfrak{S}^{\mathcal{M}}$. Then:

$$\sigma \in \mathfrak{S}^{M} \text{ for all MDPs } M \in \langle \mathcal{M} \rangle.$$

The idea here is that in each state, the actions are preserved by parameter instantiations.

Well-defined valuations for a pMDP are also well-defined for the induced pMC. The reverse does not hold in general. When applying a strategy to a pMDP, some transitions—and thus some restrictions on the well-defined valuations—disappear, but no new transitions—and thus no restrictions—are added. The most extreme example is

a pMDP in which some parameters do not occur after applying a strategy to the pMDP. Consequently:

Proposition 4.2 Let $\mathcal{M}$ be a pMDP with strategy $\sigma \in \mathfrak{S}^{\mathcal{M}}$. Then:

$$\mathsf{Val}^{\mathrm{wd}}_{\mathcal{M}} \subseteq \mathsf{Val}^{\mathrm{wd}}_{\mathcal{M}[\sigma]}.$$

Together, we deduce that the instantiation of parameters and strategies commute:

Proposition 4.3 Let $\mathcal{M}$ be a pMDP with some strategy $\sigma \in \mathfrak{S}^{\mathcal{M}}$ and a valuation $\mathsf{val} \in \mathsf{Val}^{\mathrm{wd}}_{\mathcal{M}}$. Then:

$$\mathcal{M}[\mathsf{val}][\sigma] = \mathcal{M}[\sigma][\mathsf{val}].$$

Before we continue, we define some convenient notation: We often refer to parameters occurring at any outgoing transition. Therefore, we overload the function $\mathsf{occurs}(\ldots)$: Let $s \in S$ be a state and $\alpha \in Act$. We let $\mathsf{occurs}(s, \alpha)$ denote all parameters occurring on transitions from state s on taking action α, or formally

occurs(. . .)

4

$$\mathsf{occurs}(s, \alpha) := \bigcup_{s' \in S} \mathsf{occurs}(\mathcal{P}(s, \alpha, s')),$$

and the parameters that occur via any action as $\mathsf{occurs}(s)$ as

$$\mathsf{occurs}(s) := \bigcup_{\alpha \in Act} \mathsf{occurs}(s, \alpha).$$

4.1.2 Measures and properties on parametric models

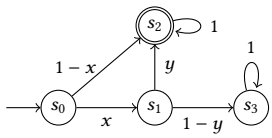
We lift measures to *solution functions* and the models relation for properties to *accepting sets*. For an advanced treatment of these objects, we refer to Section 5.2 on page 121.

Measures | We consider three measures: (unbounded) maximal/minimal reachability probability, and maximal/minimal expected reward. Furthermore, we consider their corresponding properties, as introduced for parameter-free models in Section 3.2. Here, we generalise these notions straightforwardly. Each well-defined parameter valuation yields an instantiation for which the measures are defined. We may thus immediately map valuations to measures, *Solution functions* capture this mapping.

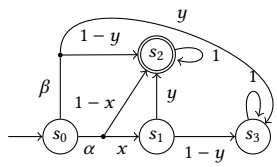
solution function

Definition 4.7 (Solution function) For a pMC $\mathcal{D}$ and a state s, let the *(probability) solution function* and the *(reward) solution function* be

$$\begin{aligned} &{}^{\mathbb{P}}\mathsf{sol}^{\mathcal{D}}_{s \to T} \colon \mathsf{Val}^{\mathrm{wd}}_{\mathcal{D}} \to [0, 1], && {}^{\mathbb{E}}\mathsf{sol}^{\mathcal{D}}_{s \to T} \colon \mathsf{Val}^{\mathrm{wd}}_{\mathcal{D}} \to \mathbb{R}^{\infty}_{\geq 0}, \\ &{}^{\mathbb{P}}\mathsf{sol}^{\mathcal{D}}_{s \to T}(\mathsf{val}) := Pr_{\mathcal{D}[\mathsf{val}]}(s \models \Diamond T), && {}^{\mathbb{E}}\mathsf{sol}^{\mathcal{D}}_{s \to T}(\mathsf{val}) := Er_{\mathcal{D}[\mathsf{val}]}(s \models \Diamond T). \end{aligned}$$



(a) A tiny pMC



(b) A tiny pMDP

Figure 4.4: Two small, acyclic models

For a pMDP $\mathcal{M}$, let

$$
\begin{aligned}
&{}^{\mathbb{P}}\mathsf{minsol}^{\mathcal{M}}_{s\to T} \colon \mathsf{Val}^{\mathrm{wd}}_{\mathcal{M}} \to [0,1],\\
&{}^{\mathbb{P}}\mathsf{minsol}^{\mathcal{M}}_{s\to T}\mathsf{val} := \min_{\sigma\in\Sigma^{\mathcal{M}}} {}^{\mathbb{P}}\mathsf{sol}^{\mathcal{M}[\sigma]}_{s\to T}(\mathsf{val}) = \min_{\sigma\in\Sigma^{\mathcal{M}}} Pr^{\sigma}_{\mathcal{M}[\mathsf{val}]}(s \models \Diamond T),
\end{aligned}
$$

and define ${}^{\mathbb{E}}\mathsf{minsol}^{\mathcal{M}}_{s\to T}$ analogously on rewards, and ${}^{\mathbb{P}}\mathsf{maxsol}^{\mathcal{M}}_{s\to T}$, ${}^{\mathbb{E}}\mathsf{maxsol}^{\mathcal{M}}_{s\to T}$ analogously as the maximum, respectively.

For conciseness we make the following simplifications. When the measure is clear from the context (or not relevant) we omit the left-superscript. We omit the initial state whenever possible and omit $\mathcal{M}$ and T whenever they are clear from the context.

Example 4.6 We first consider the pMC $\mathcal{D}$ in Figure 4.4a. There are two paths to the target state. The probability to reach the target is the sum over the probabilities over these two paths, and thus, the probability is $f := x \cdot y + 1 - x$. For any well-defined instantiation val, the probability to reach the target in $\mathcal{D}[\mathsf{val}]$ is $f[\mathsf{val}]$. Thus,

$$ {}^{\mathbb{P}}\mathsf{sol}^{\mathcal{D}}_{T} = x \cdot y + 1 - x. $$

Now, let $\mathsf{prew} := \{s_0 \mapsto 1, s_1 \mapsto x, s_2, s_3 \mapsto 0\}$. To determine the expected reward to $T' := \{s_2, s_3\}$ three paths are relevant. We obtain

$$ {}^{\mathbb{E}}\mathsf{sol}^{\mathcal{D}}_{T'} = (1-x) + xy \cdot (1+x) + x(1-y) \cdot (1+x) = 1 + x^2. $$

For the pMDP in Figure 4.4b, we have (after reformulation)

$$ {}^{\mathbb{P}}\mathsf{minsol}^{\mathcal{M}}_{T} = \min\{1-(1-y)x, 1-y\}. $$

We observe that for a combination of large values for y, action β minimises the probability to the target, whereas for smaller values of y, depending on the value for x, α minimises the probability.

Example 4.7 For the pMC in Figure 4.1b there are infinitely many paths to, e.g., $\square$. Using techniques from Section 8.1 the solution function is

$$^{\mathbb{P}}\mathsf{sol}_T^{\mathcal{D}} = \begin{cases} \frac{x\cdot(1-y)\cdot(1-x)}{1-xy} & \text{if } x \neq 1 \text{ and } y \neq 1, \\ 0 & \text{otherwise.} \end{cases}$$

Properties | In this paragraph, we consider some fixed property φ. We may partition well-defined valuations into two sets, based on the satisfaction of the property by an instantiated model. As for parameter-free models, we need to distinguish how we quantify over the nondeterminism.

Definition 4.8 (Accepting/Rejecting valuations) Let $\mathcal{M}$ be a pMDP, φ a property, and Q a quantifier. We define the set of *accepting* valuations ·accepting

$$\mathsf{acc}_{Q,\varphi}^{\mathcal{M}} := \{\mathsf{val} \in \mathsf{Val}_{\mathcal{M}}^{\mathrm{wd}} \mid \mathcal{M}[\mathsf{val}] \; Q\models \varphi\},$$

and analogously, the set of *rejecting* valuations ·rejecting

$$\mathsf{rej}_{Q,\varphi}^{\mathcal{M}} := \{\mathsf{val} \in \mathsf{Val}_{\mathcal{M}}^{\mathrm{wd}} \mid \mathcal{M}[\mathsf{val}] \; Q\not\models \varphi\}.$$

Consequently, we refer to $\langle \mathcal{M} \mid \mathsf{acc}_{Q,\varphi}\rangle$ as the *accepting instantiations*. For pMCs, universal and existential quantification coincide, and we omit the quantifier. A subset of *the* accepting valuations (instantiations) is called accepting, and analogously, a subset of the rejecting valuations is called rejecting. A subset that is neither accepting nor rejecting is called *inconsistent*. ·inconsistent

Example 4.8 We continue with Example 4.6. First, consider the pMC $\mathcal{D}$ in Figure 4.4a and observe that the quantifier over the strategies can be dropped. Let $\varphi := \mathbb{P}_{\geq 1/2}(\Diamond T)$. We obtain

$$\mathsf{acc}_{\varphi}^{\mathcal{D}} = \{\mathsf{val} \in \mathsf{Val}_{\mathcal{D}}^{\mathrm{wd}} \mid {}^{\mathbb{P}}\mathsf{sol}_T^{\mathcal{D}}[\mathsf{val}] \geq 1/2\},$$

and consequently,

$$\mathsf{rej}_{\varphi}^{\mathcal{D}} = \{\mathsf{val} \in \mathsf{Val}_{\mathcal{D}}^{\mathrm{wd}} \mid {}^{\mathbb{P}}\mathsf{sol}_T^{\mathcal{D}}[\mathsf{val}] < 1/2\}.$$

If $\mathsf{val}(x) \leq 1/2$, then from considering the transition $s_0 \xrightarrow{1-x} s_2$ we may conclude that val is accepting. Thus, the set

$$\{\mathsf{val} \in \mathsf{Val}_{\mathcal{D}}^{\mathrm{wd}} \mid \mathsf{val}(x) \leq \frac{1}{2}\}$$

is an accepting set.
Now, we consider the pMDP $\mathcal{M}$ in Figure 4.4b. Let $\varphi := \mathbb{P}_{\geq 1/2}(\Diamond T)$. For (p)MDPs, we have to fix the quantification over the strategies. We first consider universal quantification. An MDP then satisfies a lower-bounded property, if even a minimising strategy satisfies the property (cf. Proposition 3.10). Thus,

$$\begin{aligned}\mathsf{acc}^{\mathcal{M}}_{\forall,\varphi} &= \{\mathsf{val} \in \mathsf{Val}^{\mathrm{wd}}_{\mathcal{M}} \mid {}^{\mathbb{P}}\mathsf{minsol}^{\mathcal{D}}_{T}[\mathsf{val}] \geq 1/2\} \\ &= \{\mathsf{val} \in \mathsf{Val}^{\mathrm{wd}}_{\mathcal{M}} \mid 1 - \big(1 - \mathsf{val}(y)\big)\,\mathsf{val}(x) \geq 1/2 \text{ and } 1 - \mathsf{val}(y) \geq 1/2\}.\end{aligned}$$

Conversely, for existential quantification over the strategies, we obtain:

$$\begin{aligned}\mathsf{acc}^{\mathcal{M}}_{\exists,\varphi} &= \{\mathsf{val} \in \mathsf{Val}^{\mathrm{wd}}_{\mathcal{M}} \mid {}^{\mathbb{P}}\mathsf{maxsol}^{\mathcal{D}}_{T}[\mathsf{val}] \geq 1/2\} \\ &= \{\mathsf{val} \in \mathsf{Val}^{\mathrm{wd}}_{\mathcal{M}} \mid 1 - \big(1 - \mathsf{val}(y)\big)\,\mathsf{val}(x) \geq 1/2 \text{ or } 1 - \mathsf{val}(y) \geq 1/2\}.\end{aligned}$$

We will see and exploit the close (and natural) connection between solution functions and the accepting set on various occasions later.

4.1.3 Sets of valuations

Parametric models are only interesting in combination with multiple parameter valuations, and we naturally reason about sets of parameter valuations. For example, we could be interested in the shape of all valuations that accept a property. We might also be interested in all valuations that represent an (almost) fair coin flip. Lastly, we are interested in valuations such that the corresponding instantiations share some traits, e.g., have the same underlying graph. We aggregate sets of parameter valuations as *regions* and then consider sets of valuations with the same topology in more detail. We consider further classes of valuations in Section 5.1.2.

Regions | In the scope of this work, it suffices to consider *regions* as sets of valuations.

region

Definition 4.9 (Region) A *region* R is a semi-algebraic set over $\mathbb{R}^k$.

Consequently, any region R may be represented by an ETR formula ψ_R over (at least) the parameters in $\mathcal{M}$. Any region which is a subset of a region R is called a *subregion* of R. Typically, it is worthwhile to consider regions as geometric objects.

Example 4.9 Consider the Knuth-Yao die pMC in Figure 4.1b. The regions

$$\mathsf{sat}(0 \leqslant x \leqslant 1 \wedge 0 \leqslant y \leqslant 1), \text{and}$$
$$\{\mathsf{val} \in \mathsf{Val}^{\mathrm{wd}}_{\mathcal{D}} \mid \mathsf{val} \in \mathsf{sat}(0 \lessdot x \lessdot 1 \wedge 0 \lessdot y \lessdot 1)\},$$

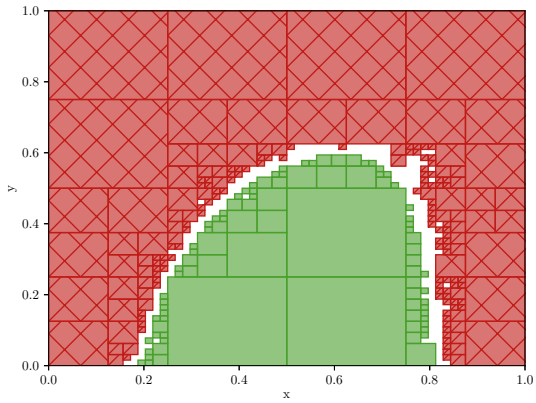


Figure 4.5: Parameter space partitioning into accepting (green), rejecting (red), and unknown (white) regions.

are two rectangular sets. The first set contains all well-defined valuations. Reconsider Example 4.8. The first accepting set is the region described as

$$\{\mathsf{val} \in \mathsf{Val}^{\mathrm{wd}}_{\mathcal{D}} \mid \mathsf{val} \in \mathsf{sat}\big({}^{\mathbb{P}}\mathsf{minsol}^{\mathcal{D}}_{T}[\mathsf{val}] \geqslant 1/2\big)\}.$$

In Figure 4.5, we depict various accepting (in red) and rejecting (in green) regions. Furthermore, the sets

$$\{\mathsf{val} \in \mathsf{Val}^{\mathrm{wd}}_{\mathcal{D}} \mid \mathsf{val} \in \mathsf{sat}(x^2 + y^2 \lessdot 1)\}, \text{ and } \{\mathsf{val} \in \mathsf{Val}^{\mathrm{wd}}_{\mathcal{D}} \mid \mathsf{val}(x) \in \{0, 1\}\},$$

are regions. The sets

$$\{\mathsf{val} \in \mathsf{Val}^{\mathrm{wd}}_{\mathcal{D}} \mid \exists i \in \mathbb{N} \text{ s.t. } \mathsf{val}(x) = \frac{1}{x^i}\}, \text{ and } \{\mathsf{val} \in \mathsf{Val}^{\mathrm{wd}}_{\mathcal{D}} \mid \mathsf{val}(x) \leq \frac{1}{\pi}\}$$

are not regions (the first set contains an infinite disjunction, the second contains a non-algebraic number).

In Section 4.3.1 we show that sets of well-defined valuations are always regions. Likewise accepting sets are always regions, as we show in Section 4.3. Thus, regions are well-suited to capture sets of valuations that we deem interesting. Regions are thus sufficiently expressive, but also ensure a string of nice properties. In particular, they have a finite representation, notions such as distance and measure may be inherited

from semi-algebraic sets, and regions are in some form well-behaved. Finally, regions are (naturally) closed under intersection, union, etc.

> **Remark 4.1** Although regions are often used to refer to connected sets, the usage of region in the literature on parameter synthesis for Markov models is more general.

Regions may be characterised by the type of their encoding. In particular, we are interested in two classes with (rather) simple encodings: affine and rectangular regions.

·affine

> **Definition 4.10 (Affine region)** A region is *affine* if it is represented by a linear constraint system.

Affine regions describe convex polytopes. We refer to the vertices (or angular points) of the polytope as the *region vertices*.

> **Definition 4.11 (Rectangular region)** A region R with representation
>
> $$\psi_R \;=\; \bigwedge_{i=1}^{k} -x_i + a_i \trianglelefteq_i^1 0, x_i + b_i \trianglelefteq_i^2 0$$
>
> with $a_i \leq b_i \in \mathbb{Q}$ and $\trianglelefteq_i \in \{<, \leq\}$ for $i = 1, \ldots, k$ is called *rectangular*. A rectangular region is *closed* if all inequalities $\trianglelefteq_i^{\cdot}$ in constraints in ψ_R are non-strict.

Rectangular regions are hyper-rectangles and a subclass of affine regions. A *closed* rectangular region R may be represented as $R = \bigtimes_{x \in X} [a_x, b_x]$ with parameter intervals $[a_x, b_x]$ described by the bounds a_x and b_x for all $x \in X$. Observe that for this notation, the order on X has to be clear. For a region R, we refer to the *bounds* of parameter x by $\mathcal{B}_R(x) = \{a_x, b_x\}$ and to the *interval* of parameter x by $\mathcal{I}_R(x) = [a_x, b_x]$. We may omit the subscript R, if it is clear from the context. For a rectangular region R, the size $\|R\|$ equals $\prod_{x \in X} b_x - a_x$.

> **Example 4.10** We continue with Example 4.9. The sets
>
> $$\{\mathsf{val} \in \mathsf{Val}_{\mathcal{D}}^{\mathrm{wd}} \mid \mathsf{val} \in \mathsf{sat}(x^2 + y^2 \lessdot 1)\}, \{\mathsf{val} \in \mathsf{Val}_{\mathcal{D}}^{\mathrm{wd}} \mid \mathsf{val}(x) \in \{0, 1\}\}$$
>
> are not affine (the first has nonlinear constraints, the second uses a disjunction). The sets
>
> $$\{\mathsf{val} \in \mathsf{Val}_{\mathcal{D}}^{\mathrm{wd}} \mid 0 < \mathsf{val}(x), \mathsf{val}(y) < 1\}, \{\mathsf{val} \in \mathsf{Val}_{\mathcal{D}}^{\mathrm{wd}} \mid \mathsf{val}(x) + \mathsf{val}(y) = 1\},$$
> $$\text{and } \{\mathsf{val} \in \mathsf{Val}_{\mathcal{D}}^{\mathrm{wd}} \mid 1/10 \leq \mathsf{val}(x) \leq 4/5 \text{ and } 2/5 \leq \mathsf{val}(y) \leq 7/10\}$$

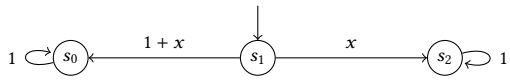


Figure 4.6: Realisable pMC without graph-preserving valuations

are all affine regions. The first and last are rectangular regions, the second is not. Only the last set is a closed rectangular set, which we also denote with $R = [1/10, 4/5] \times [2/5, 7/10]$ for parameter ordering x, y.

Graph-preserving valuations | Recall that $\langle \mathcal{M} \rangle$ considers the instantiations that are induced by well-defined valuations. Below, we consider a restriction on the valuations. In the analysis of parameter-free MDPs, it is often essential to exploit the topology of the MDP, e.g., when computing zero-states. In $\langle \mathcal{M} \rangle$, not all MDPs have the same topology. The goal below is to consider a restriction on the valuations such that all MDPs in $\langle \mathcal{M} \rangle$ have the same topology as $\mathcal{M}$. The topology changes, if a transition is present in the pMDP but not in its instantiation.

Definition 4.12 (Vanishing transitions) Let $\mathcal{M}$ be a pMDP with $\mathsf{val} \in \mathsf{Val}^{\mathrm{wd}}_{\mathcal{M}}$. We call a transition (s, α, s) *vanishing* under val if vanishing

$$\mathcal{P}(s, \alpha, s') \neq 0 \quad \text{and} \quad \mathcal{P}(s, \alpha, s')[\mathsf{val}] = 0.$$

The set $\mathsf{Vanish}_{\mathcal{M}}(\mathsf{val}) \subseteq S \times Act \times S$ contains all vanishing transitions.

A valuation preserves the topology (the graph) if no transitions vanish, formally:

Definition 4.13 (Graph-preserving valuations) Let $\mathcal{M}$ be a pMDP. A valuation $\mathsf{val} \in \mathsf{Val}^{\mathrm{wd}}_{\mathcal{M}}$ is *graph-preserving* if $\mathsf{Vanish}_{\mathcal{M}}(\mathsf{val}) = \emptyset$. The set $\mathsf{Val}^{\mathrm{gp}}_{\mathcal{M}}$ contains all graph-preserving valuations. ·graph-preserving

The generator for this class is then $\langle \mathcal{M} \mid \mathsf{Val}^{\mathrm{gp}}_{\mathcal{M}} \rangle$, alternatively denoted $\langle \mathcal{M} \mid \mathrm{gp} \rangle$.

Example 4.11 Let us again consider Figure 4.1b. If we set $\mathsf{val}(x) := 0$, then various transitions disappear, in particular the transitions from s_0 to s_1, and from s_3 to s_1. Thus, any valuation with $\mathsf{val}(x) = 0$ is not graph-preserving. Later in Example 5.21 we illustrate that instantiations which are not graph-preserving might require a case distinction in the solution function (e.g., to avoid division by zero). In Theorem 5.18 on page 122, we show that this observation is general. There exist realisable pMCs without graph-preserving instantiations, cf. Figure 4.6.

		quantification over strategies	
		$\exists$	$\forall$
quantification over valuations	$\exists$	angelic feasiblity	demonic feasibilty
	$\forall$	angelic validity	demonic validity

Table 4.1: Quantification over parameters (first) and strategies (second).

Regions may have particular characteristics that are shared by its valuations. For example, when all valuations are graph-preserving, it is a graph-preserving region. Slightly weaker, a region is graph-consistent, when all valuations induce the same topology (but not necessarily the topology of the corresponding pMDP).

Definition 4.14 (Graph-consistent region) A graph-consistent region R is a subregion of the well-defined valuations such that for all $\mathsf{val}, \mathsf{val}' \in R$:

$$\mathsf{Vanish}_{\mathcal{M}}(\mathsf{val}) = \mathsf{Vanish}_{\mathcal{M}}(\mathsf{val}').$$

The region is maximally graph-consistent, if no true superset of R is graph-consistent.

Example 4.12 Let us again consider Figure 4.1b. As we have seen previously, valuations with $\mathsf{val}(x)$ are not graph-preserving. However, the set

$$\{\mathsf{val} \in \mathsf{Val}^{\mathrm{wd}}_{\mathcal{D}} \mid \mathsf{val}(x) = 0 \wedge 0 < \mathsf{val}(y) < 1\}$$

is maximally graph-consistent.

Graph-consistent regions satisfy the same qualitative properties, see Proposition 4.7 on page 84, and the well-defined valuations may be decomposed into graph-consistent regions, see Corollary 5.14 on page 119.

4.2 Problem Statements

This section aims to centrally collect various variants of the problem statement that are addressed or mentioned throughout this thesis. The most essential are the following four problems: Given a pMDP $\mathcal{M}$, a region R, and a property φ:

- (**angelic feasibility**)
  is there an MDP $M \in \langle \mathcal{M} \mid R \rangle$ s.t. $M \mathrel{\exists\!\models} \varphi$, or equivalently, is $\mathsf{acc}^{\mathcal{M}}_{\exists,\varphi} \cap R \neq \emptyset$?
- (**demonic feasibility**)
  is there an MDP $M \in \langle \mathcal{M} \mid R \rangle$ s.t. $M \mathrel{\forall\!\models} \varphi$, or equivalently, is $\mathsf{acc}^{\mathcal{M}}_{\forall,\varphi} \cap R \neq \emptyset$?
- (**angelic validity**)
  for each MDP $M \in \langle \mathcal{M} \mid R \rangle$, does $M \mathrel{\exists\!\models} \varphi$ hold, or equivalently, is $R \subseteq \mathsf{acc}^{\mathcal{M}}_{\exists,\varphi}$?

- (**demonic validity**)
  for each MDP $M \in \langle \mathcal{M} \mid R \rangle$, does $M \mathrel{\forall\!\models} \varphi$, or equivalently, is $R \subseteq \mathsf{acc}^{\mathcal{M}}_{\forall,\varphi}$?

For pMCs, the angelic and demonic problems coincide, respectively. We remark that feasibility and validity problems are dual. Table 4.1 gives a conceptual overview of the combinations of quantifiers over the valuations and the strategies. Observe that we always quantify over the valuations first. Quantification over the strategies and then over the valuations is briefly discussed in Section 5.5.

Example 4.13 First consider the pMC for the Knuth-Yao die from Example 4.1, with φ describing the question whether the probability to reach ⚁ is at most $3/20$. Validity reflects the question whether irrespectively of how unfair the coin flips are (within bounds given by the region), this probability bound is met. Feasibility asks whether there even is one such parameter valuation. Thus, when considering $\neg\varphi$, feasibility asks for one counterexample to the validity formulated above.
Now reconsider the rock-paper-scissors example pMDP from Example 4.2. We let φ describe that the probability of us loosing is at most 90%. Demonic feasibility reflects the setting from the initial example, where we want to find distributions (technically: valuations describing distributions) such that, no matter how Eve plays, we can guarantee we loose at most 90% of the games. Demonic validity reflects the setting where we certify that the region describes a set of distributions (parameter valuations) such that, no matter which valuations from the region we use, and no matter how Eve plays, we can guarantee we loose at most 90% of the games. The valuations constitute a set of *permissive valuations*: using any of them is guaranteed to be good enough. Angelic validity is the dual to demonic feasibility: when considering $\neg\varphi$, it may be used to show that no matter how we play, Eve has a way of countering such that we cannot guarantee that we loose at most 90% of the games. Angelic feasibility reflects looking for a strategy such that we—if Eve is stupid—may hope that Eve looses more than half of the games.

permissive valuations

Validity problems may be considered verification problems, in which we want to certify that certain behaviour is unlikely irrespectively of the precise parameter values, see the examples above. Consequently, we often refer to validity as *region verification*. Feasibility problems are then typical instances of a synthesis problem[1] and we are mostly interested in obtaining the corresponding instantiation that satisfies φ. We discuss our approaches for feasibility and validity in Chapter 8.

region verification

We now extend the feasibility problems to *optimal feasibility* as we are often interested in preferably good or even optimal instantiations. For example, we want to optimise the reachability probability: Given a pMDP $\mathcal{M}$, a region R, and a target set T,

optimal feasibility

[1] We remark that arguably the feasibility problems in this context are simpler, as guessing parameter values correctly and checking a parameter-free model is sufficient.

- (**demonic supremum probability**)
  find $\sup_{\mathsf{val} \in R} \min_{\sigma \in \Sigma^{\mathcal{M}}} Pr^{\sigma}_{\mathcal{M}[\mathsf{val}]}(\Diamond T)$,
- (**demonic infimum probability**)
  find $\inf_{\mathsf{val} \in R} \max_{\sigma \in \Sigma^{\mathcal{M}}} Pr^{\sigma}_{\mathcal{M}[\mathsf{val}]}(\Diamond T)$.

or the analogue formulations with expected rewards, or (typically interesting to a lesser extent) their angelic counterparts. We stress that the maximum (or minimum) over such sets must not necessarily exist, see also Section 5.2. In fact, we mostly want to find instantiations that optimise the reachability probability. Therefore, we resort to an η-optimal synthesis problem. For conciseness, we only formulate the problem for maximal probabilities: Given a pMDP $\mathcal{M}$, a region R, a tolerance η, and a target set T,

- (**demonic η-maximal feasibility**)
  find $M \in \langle \mathcal{M} \mid R \rangle$ such that $Pr^{\sigma}_{M}(\Diamond T) > (1 - \eta) \cdot \sup_{\mathsf{val} \in R} \min_{\sigma \in \Sigma^{\mathcal{M}}} Pr^{\sigma}_{\mathcal{M}[\mathsf{val}]}(\Diamond T)$.

Indeed, this formulation allows us to extract the matching valuation / instantiation.

> **Example 4.14** In the Knuth-Yao die pMC from Example 4.1, finding the optimal parameter valuation corresponds to finding a combination of unfair coins that maximises the probability to reach ⚁, or analogously—but in this example much simpler— a combination of unfair coins that minimises this probability. In the rock-paper-scissors pMDP from Example 4.2, finding the optimal parameter valuation under demonic nondeterminism reflects exactly the optimisation of our worst-case performance, i.e., when Eve is playing optimally. Finding optimal parameter valuations under angelic nondeterminism reflects the (potentially uninteresting) scenario where Eve is trying to loose as well.

We may put additional constraints on the parameter valuations. Given a pMDP $\mathcal{M}$, a region R and property φ, and additionally a *parameter-cost function*[1]

parameter-cost function

$$\mathsf{cost}_{\mathcal{M}} \colon \mathsf{Val}^{\mathrm{wd}}_{\mathcal{M}} \to \mathbb{R}_{\geq 0},$$

and a budget κ. Whenever $\mathsf{cost}_{\mathcal{M}}$ is given by a (maximum/minimum over) rational function(s), this problem is a mere reformulation of the feasibility problems with a region defined by

$$R' := R \cap \{\mathsf{val} \mid \mathsf{cost}_{\mathcal{M}}(\mathsf{val}) \leq \kappa\} \text{ for some given } \kappa \in \mathbb{R}_{\geq 0}$$

It is more interesting to optimise the valuation among all accepting valuations. The quantitative variant is then to find the so-called *repair*:

- (**model repair**) [BGKR+11]
  find $\mathsf{val}^* := \arg\min_{\mathsf{val} \in \mathsf{acc}^{\mathcal{M}}_{\varphi}} \mathsf{cost}_{\mathcal{M}}(\mathsf{val})$.

[1] Equivalently defined as $\mathsf{cost}_{\mathcal{M}} \colon \langle \mathcal{M} \rangle \to \mathbb{R}_{\geq 0}$

Example 4.15 For the Knuth-Yao die from Example 4.1, model repair may be used to search for the minimal deviation from a fair coin such that a property as in Example 4.13 is violated. For the rock-paper-scissors pMDP from Example 4.2, model repair may find the maximal deviation from selecting a uniform distribution over the parameters such that the probability for Eve to loose is sufficiently large.

All these variants of optimal feasibility naturally contain two obligations: Finding a feasible valuation and showing that (almost) all other valuations are invalid. Thus, approaches for these optimisation problems may be inferred by naive or involved combinations of approaches for validity and feasibility.

Let us reconsider the model repair problem. There, the optimal repair may be determined independent of the actual pMDP once the accepting set is fixed, i.e., it is an optimisation problem over $\langle \mathcal{M} \mid \mathsf{acc}_\varphi \rangle$, or equivalently, over $\mathsf{acc}_\varphi^{\mathcal{M}}$. This optimisation is often straightforward when there is a concise and suitable representation for $\mathsf{acc}_\varphi^{\mathcal{M}}$.

The latter observation, among others, motivates the *complete partitioning* problem, i.e., given a pMDP, a region R and a property φ, complete partitioning

- (**complete partitioning**)
  find $R \cap \mathsf{acc}_\varphi^{\mathcal{M}}$ and $R \setminus \mathsf{acc}_\varphi^{\mathcal{M}}$ and represent them concisely.

and represent them concisely. Another strong motivation is that this constitutes a synthesis extension to the validity problem: Instead of checking whether a set of valuations is accepting, it finds the (largest) accepting set. Such sets are *robust* for perturbations: if the valuation is slightly moved, the valuation still satisfies the property. We refer to Example 4.8 for examples of these sets. One representation for $\mathsf{acc}_\varphi^{\mathcal{M}}$ is supported by (closed forms for) the solution function. However, the solution function is often complex and highly nonlinear. Therefore, we are interested in the *partial partitioning* problem, i.e., given a pMDP $\mathcal{M}$, a region R a property φ and an $\eta \in [0, 1]$: partial partitioning

- (**partial partitioning**)
  find $R_{\mathsf{acc}} \subseteq \mathsf{acc}_\varphi^{\mathcal{M}}, R_{\mathsf{rej}} \subseteq \mathsf{rej}_\varphi^{\mathcal{M}}$ such that
  $$\|R_?\| \leq \eta \cdot \|R\| \quad \text{with } R_? := R \setminus (R_{\mathsf{acc}} \cup R_{\mathsf{rej}}).$$

We refer to $R_?$ as an *unknown region*. We have dedicated Section 8.4 to computational approaches to this problem. unknown region

Example 4.16 For the Knuth-Yao die in Figure 4.1b, Figure 4.5 on page 71 shows the partitioning into accepting (green), rejecting (red), and unknown (white) regions for $\eta := 0.95$.

We emphasise that this list is not meant to be exhaustive, but rather motivates the problems that we consider in this thesis. Most prominently, we do not consider

sensitivity analysis, i.e., we do not study the derivative. Some work in this direction has been done in [CFRS14; SFCR16; 18]. In particular, it should clarify that feasibility is a fundamental problem underlying many more formulations.

We finish by briefly illustrating why already feasibility synthesis is hard: the parameter dependencies yield an intricate trade-off.

Example 4.17 Reconsider the pMC in Figure 4.1b: We aim to maximise the probability to reach ⚁, a classical instance of the optimal feasibility problem, but a method that is also helpful for just finding a feasible solution if the threshold is (almost) the maximum. Inspection of the graph structure shows that, in s_0, it is beneficial if x is as large as possible. The same argument applies to state s_3. However, in state s_4, the larger x, the more probability mass flows to ⚁. Thus, from state s_4, it is beneficial if x is as small as possible. Overall, the optimal value for x depends also on the value for y (as this determines the probability mass arriving in s_3) and is also reflected by Figure 4.5.

pMCs without parameter dependencies are considered in Section 5.1.1.3.

4.3 Encoding into Existential Theory of the Reals

In this section, we show that the feasibility problems may be formulated in the ETR. While we show that these results hold for arbitrary pMCs and pMDPs, we often present the ideas only for so-called *polynomial pMDPs*:

·polynomial

Definition 4.15 A pMDP is *polynomial*, if

$$\text{for all } s, s' \in S, \alpha \in \mathbb{A}\mathrm{ct}(s) : \mathcal{P}(s, \alpha, s') \in \mathbb{Q}[X] \text{ and } \mathsf{prew}(s) \in \mathbb{Q}[X].$$

The rationale is twofold. First, instantiating rational functions might yield undefined values when the denominator becomes zero. That induces an additional case distinction which is undesirable for a concise presentation of the theory. Furthermore, below we want to formulate ETR formulae using transition probabilities in the constraints. However, the ETR does not natively support rational functions[1].

The remainder of the section is structured as follows: We first show that well-defined and graph-preserving sets of valuations are indeed ETR-encodable and the sets of all well-defined (graph-preserving) valuations are thus regions, respectively. Then we give a detailed account on how to encode the feasibility problem on pMCs. We start with the qualitative properties and then continue step-wise with quantitative properties. First, we consider reachability probabilities and the easier case of (subregions of) graph-preserving regions, then in general for (subregions of) well-defined regions, and for

[1] it requires, e.g., defining whether $0/0 > 0$.

expected rewards. We then show the lifted encodings to both angelic and demonic nondeterminism in pMDPs. Finally, we show that for a pMC and (subregions of) a graph-preserving region, the solution function might be used to construct compact ETR formulae.

> **Remark 4.2** We encourage any reader who has seen encodings of Markov chains or MDPs into (fragments of) ETR before to skip the details in this section. On the other hand, if you are interested in solving feasibility problems for small pMCs and pMDPs, this section is all (theory) you need and there is likely little gain in reading beyond this section, except for intellectual merit.

4.3.1 Encoding valuations as regions

Below, we show that sets of all well-defined or graph-preserving valuations are indeed regions. As the ETR encodings are more natural on polynomial pMDPs, we first focus on this simpler case and then continue with the general case.

4.3.1.1 Polynomial pMDPs

The following set of constraints is a natural encoding of Definition 4.3 on page 65.

Constraints 4.1 (Well-defined regions) The following constraints capture well-defined valuations for a polynomial pMDP $\mathcal{M}$:

$$
\begin{aligned}
&0 \leqslant \mathcal{P}(s,\alpha,s') \leqslant 1 \text{ for all } s,s' \in S, \alpha \in \mathbb{A}\mathrm{ct}(s) \quad (\text{with } \mathcal{P}(s,\alpha,s') \neq 0),\\
&\sum_{s' \in S} \mathcal{P}(s,\alpha,s') = 1 \text{ for all } s \in S, \alpha \in \mathbb{A}\mathrm{ct}(s),\\
&\mathsf{prew}(s) \geqslant 0 \qquad \text{for all } s \in S.
\end{aligned}
$$

We denote the corresponding formula for this constraint system with $\Phi^{\mathcal{M}}_{\mathrm{wd}}$.

The constraints ensure that (1) all (non-zero) transitions are evaluated to a probability, (2) transition probabilities describe distributions, and (3) rewards are non-negative. The following proposition states that the encoding is sound and complete.

> **Proposition 4.4** Let $\mathcal{M}$ be a polynomial pMDP:
>
> $$\mathsf{sat}(\Phi^{\mathcal{M}}_{\mathrm{wd}}) = \mathsf{Val}^{\mathrm{wd}}_{\mathcal{M}}.$$

That is, the set of well-defined valuations of some $\mathcal{M}$ form a region: the *well-defined region* of $\mathcal{M}$. ·well-defined

Example 4.18 Recall the pMC $\mathcal{D}$ for the Knuth-Yao die from Figure 4.1b, with the well-defined valuations as in Example 4.4 on page 65. We have:

$$\begin{aligned}\Phi_{\text{wd}}^{\mathcal{D}} &= x \geqslant 0 \ \wedge\ 1 - x \geqslant 0 \ \wedge\ x + 1 - x \doteq 1 \\ &\wedge\ y \geqslant 0 \ \wedge\ 1 - y \geqslant 0 \ \wedge\ y + 1 - y \doteq 1.\end{aligned}$$

This formula simplifies to $0 \leqslant x \leqslant 1 \wedge 0 \leqslant y \leqslant 1$.
Now recall the pMDP $\mathcal{M}$ for the rock-paper-scissors pMDP from Figure 4.2a, with the well-defined valuations as in Example 4.4. We get:

$$\begin{aligned}\Phi_{\text{wd}}^{\mathcal{M}} &= x_R \geqslant 0 \ \wedge\ x_P \geqslant 0 \ \wedge\ x_S \geqslant 0 \ \wedge\ x_R + x_P + x_S \doteq 1 \\ &\wedge\ x'_R \geqslant 0 \ \wedge\ x'_P \geqslant 0 \ \wedge\ x'_S \geqslant 0 \ \wedge\ x'_R + x'_P + x'_S \doteq 1.\end{aligned}$$

This encoding is easily extended to describe graph-preserving valuations, based on Definition 4.13 on page 73.

Constraints 4.2 The formula $\Phi_{\text{gp}}^{\mathcal{M}}$ captures the graph-preserving valuations for a polynomial pMDP $\mathcal{M}$:

$$\Phi_{\text{gp}}^{\mathcal{M}} := \Phi_{\text{wd}}^{\mathcal{M}} \wedge \bigwedge_{\substack{s, s' \in S, \alpha \in \mathbb{A}\text{ct}(s) \\ \mathcal{P}(s,\alpha,s') \neq 0}} 0 \lessdot \mathcal{P}(s, \alpha, s').$$

The formula may be simplified to the following constraints:

$$\begin{aligned}&\mathcal{P}(s, \alpha, s') \gtrdot 0 && \text{for all } s, s \in S, \alpha \in \mathbb{A}\text{ct}(s) \text{ with } \mathcal{P}(s, \alpha, s') \neq 0, \\ &\sum_{s' \in S} \mathcal{P}(s, \alpha, s') = 1 && \text{for all } s \in S, \alpha \in \mathbb{A}\text{ct}(s), \\ &\mathsf{prew}(s) \geqslant 0 && \text{for all } s \in S.\end{aligned}$$

Proposition 4.5 Let $\mathcal{M}$ be a polynomial pMDP:

$$\mathsf{sat}(\Phi_{\text{gp}}^{\mathcal{M}}) = \mathsf{Val}_{\mathcal{M}}^{\text{gp}}.$$

Example 4.19 We continue with the pMC $\mathcal{D}$ and pMDP $\mathcal{M}$ from Example 4.18. We have:

$$\begin{aligned}\Phi_{\text{gp}}^{\mathcal{D}} &= x \gtrdot 0 \ \wedge\ 1 - x \gtrdot 0 \ \wedge\ x + 1 - x \doteq 1 \\ &\wedge\ y \gtrdot 0 \ \wedge\ 1 - y \gtrdot 0 \ \wedge\ y + 1 - y \doteq 1.\end{aligned}$$

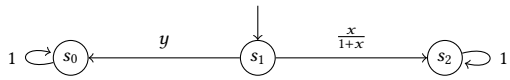


Figure 4.7: Non-polynomial pMC

This formula simplifies to $0 \lessdot x \lessdot 1 \wedge 0 \lessdot y \lessdot 1$. Furthermore:

$$\Phi_{\text{gp}}^{\mathcal{M}} = x_R \gtrdot 0 \ \wedge \ x_P \gtrdot 0 \ \wedge \ x_S \gtrdot 0 \ \wedge \ x_R + x_P + x_S \doteq 1$$
$$\wedge \ x'_R \gtrdot 0 \ \wedge \ x'_P \gtrdot 0 \ \wedge \ x'_S \gtrdot 0 \ \wedge \ x'_R + x'_P + x'_S \doteq 1.$$

4.3.1.2 Handling rational functions

In case the transition probability and reward function are not polynomials, the left-hand side of the (in)equalities in Constraints 4.1 or Constraints 4.2 would not be polynomials and the inequalities would not be constraints. We therefore perform the following transformations on these (in)equalities: Let $g_1, g_2 \in \mathbb{Q}[X]$ and $c \in \mathbb{Q}$.

- Transformation of equalities:
$$\frac{g_1}{g_2} = c \quad \text{becomes} \quad g_1 - c{\cdot}g_2 = 0 \wedge g_2 \neq 0.$$
- Transformation of inequalities $\trianglerighteq \ \in \{\gtrdot, \geqslant\}$,
$$\frac{g_1}{g_2} \trianglerighteq c \text{ becomes } g_2 \neq 0 \wedge \Big((g_2 > 0 \wedge g_1 \trianglerighteq c{\cdot}g_2) \vee (g_2 < 0 \wedge g_1 \trianglerighteq^{\text{inv}} c{\cdot}g_2)\Big)$$
with $\trianglerighteq^{\text{inv}}$ equals $\lessdot$ for $\gtrdot^{\text{inv}}$ and $\leqslant$ for $\geqslant^{\text{inv}}$.
- Transformation of $\lessdot, \leqslant$ follows analogously.
- Transformation of $g \neq g'$ (i.e., of the formula $g \lessdot g' \vee g \gtrdot g'$) follows the application on both inequalities.

The result is a formula with polynomial constraints that correctly describes graph preservation (or well-definedness).

Example 4.20 Consider the pMC in Figure 4.7. The graph-preservation inequalities are (after some simplification):

$$y \gtrdot 0 \text{ and } \frac{x}{1+x} \gtrdot 0 \text{ and } y + \frac{x}{1+x} \doteq 1.$$

Transforming the second statement as explained before results in:

$$1 + x \not\doteq 0 \wedge \Big((1 + x \gtrdot 0 \wedge x \gtrdot 0) \vee (1 + x \lessdot 0 \wedge x \lessdot 0)\Big),$$

while transforming the third statement yields:

$$(1 + x \not\doteq 0) \wedge y{\cdot}(1{+}x) - 1 \doteq 0.$$

Finally, we obtain the following formula (after some further simplifications):

$$y \gtrdot 0 \;\wedge\; (x \gtrdot 0 \vee x \lessdot -1) \;\wedge\; y \cdot (1 + x) - 1 \doteq 0,$$

and after even more simplifications:

$$y \gtrdot 0 \;\wedge\; x \gtrdot 0 \;\wedge y(1 + x) \doteq 1.$$

4.3.2 Encoding pMC feasibility

We have seen how to encode regions into ETR. This encoding allows us to consider the more interesting question how to actually encode reachability. We start with pMCs, which we consider extensively as the ideas for pMDPs are mostly straightforward extensions. Together, this subsection establishes that for every pMC and every (considered) property, the sets of accepting and rejecting valuations are regions.

4.3.2.1 Qualitative analysis

Before we treat the feasibility analysis for quantitative properties, we start with the qualitative case. For that, we generalise the graph-based characterisations to pMCs. First, observe that the zero- and one-states now depend on the valuation:

zero-states

Definition 4.16 Let $\mathcal{D}$ be a pMC. The *zero-states* for valuation $\mathsf{val} \in \mathsf{Val}^{\mathrm{wd}}_{\mathcal{D}}$ is the set

$$S^{\mathsf{val},T}_{=0} := \{s \in S \mid {}^{\mathbb{P}}\mathsf{sol}^{\mathcal{D}}_{s\to T}[\mathsf{val}] = 0\}$$

containing the states that reach the target with probability zero in instantiation val and the *one-states* for valuation $\mathsf{val} \in \mathsf{Val}^{\mathrm{wd}}_{\mathcal{D}}$ is the set

one-states

$$S^{\mathsf{val},T}_{=1} := \{s \in S \mid {}^{\mathbb{P}}\mathsf{sol}^{\mathcal{D}}_{s\to T}[\mathsf{val}] = 1\}$$

containing all states that reach the target almost surely.

Indeed, these sets vary for different valuations.

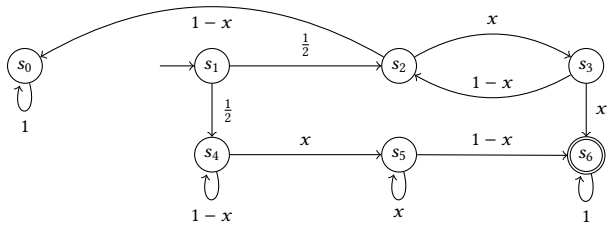


Figure 4.8: Example pMC

Example 4.21 Consider the pMC in Figure 4.8 with $T = \{s_6\}$. First, let $\mathsf{val} = \{x \mapsto \frac{1}{2}\}$. We infer from the graph that:

$$S_{=0}^{\mathsf{val},T} = \{s_0\}, \text{ and } S_{=1}^{\mathsf{val},T} = \{s_4, s_5, s_6\}.$$

For $\mathsf{val} = \{x \mapsto 0\}$ we have

$$S_{=0}^{\mathsf{val},T} = \{s_0, s_1, s_2, s_3, s_4\}, \text{ and } S_{=1}^{\mathsf{val},T} = \{s_5, s_6\},$$

and for $\mathsf{val} = \{x \mapsto 1\}$ we have

$$S_{=0}^{\mathsf{val},T} = \{s_0, s_4, s_5\}, \text{ and } S_{=1}^{\mathsf{val},T} = \{s_2, s_3, s_6\}.$$

We make the following observation:

Lemma 4.6 For any $\mathsf{val}, \mathsf{val}' \in \mathsf{Val}_{\mathcal{D}}^{\mathsf{wd}}$,

$$\mathsf{Vanish}_{\mathcal{D}}(\mathsf{val}) \subseteq \mathsf{Vanish}_{\mathcal{D}}(\mathsf{val}') \quad \text{implies} \quad S_{=0}^{\mathsf{val},T} \subseteq S_{=0}^{\mathsf{val}',T},$$

and

$$\mathsf{Vanish}_{\mathcal{D}}(\mathsf{val}) = \mathsf{Vanish}_{\mathcal{D}}(\mathsf{val}') \quad \text{implies} \quad S_{=0}^{\mathsf{val},T} = S_{=0}^{\mathsf{val}',T}.$$

Essentially, removing transitions may cut states from having a path to the target states, but never adds new paths. Furthermore, as for any graph-consistent region, the sets of vanishing transitions are invariant, we obtain from [HHZ11b] the following property[1]:

[1] the formulation using pMDPs allows for a more concise treatment later.

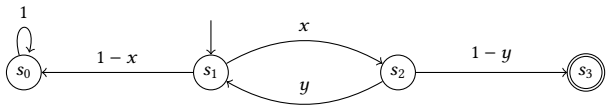


Figure 4.9: Another pMC

Proposition 4.7 Let $\mathcal{M}$ be a pMDP with target states T and graph-consistent region R. For any $M, M' \in \langle \mathcal{M} \mid R \rangle$ and $\sigma \in \Sigma^{\mathcal{M}}$:

$$Pr^{\sigma}_{M}(\Diamond T) = 0 \quad \text{implies} \quad Pr^{\sigma}_{M'}(\Diamond T) = 0, \qquad \text{and}$$
$$Pr^{\sigma}_{M}(\Diamond T) = 1 \quad \text{implies} \quad Pr^{\sigma}_{M'}(\Diamond T) = 1.$$

This proposition justifies the notation $S^{R,T}_{=0}$ and $S^{R,T}_{=1}$ as being the (unique) sets $S^{\mathsf{val},T}_{=0}$, $S^{\mathsf{val},T}_{=1}$ for some $\mathsf{val} \in R$, respectively. An alternative proof for Proposition 4.7 follows directly from the graph-based algorithms for qualitative properties [BK08]. Thus, for any fixed graph-consistent region, the sets $S^{\mathsf{val},T}_{=0}$ and $S^{\mathsf{val},T}_{=1}$ may be computed on the parameter-free $\mathcal{D}[\mathsf{val}]$. However, when regarding a non-graph-consistent region, this does not necessarily suffice.

We consider general (well-defined) regions. For positive reachability, we observe that removing transitions is never beneficial and (non-empty) graph-preserving regions are in that sense optimal for positive reachability. Formally:

Theorem 4.8 Given a pMDP $\mathcal{M}$ with $\mathsf{val} \in \mathsf{Val}^{\mathrm{gp}}_{\mathcal{M}}$, target states T and $\sigma \in \Sigma^{\mathcal{M}}$, then:

$$\text{exists } M \in \langle \mathcal{M} \rangle \text{ s.t. } Pr^{\sigma}_{M}(\Diamond T) > 0 \quad \text{iff} \quad Pr^{\sigma}_{\mathcal{M}[\mathsf{val}]}(\Diamond T) > 0.$$

A simple correctness argument follows again from the graph-based algorithms for positive reachability. Observe that strictly speaking, we don't even require val explicitly, instead, we may resort to running the graph-based algorithm for positive reachability of parameter-free MDPs on the graph of the MDP.

To the best of our knowledge, there is no analogous statement for almost-sure reachability. In particular, almost-sure reachability is about *preventing* to reach (bad) sink states, and thus, removing transitions may be beneficial, unlike in positive reachability. A discussion of the complexity of these problems is deferred until Section 5.4.

Below, we introduce encodings for both positive and almost-sure reachability. These encodings are not only used for qualitative analysis itself, but are also prominent building blocks for the encoding of quantitative measures later.

Positive reachability | In an MC, positive reachability follows from the existence of a path to the target. In particular, one may assume that this path is cycle-free. States along such a path may be ranked, i.e., they may be assigned numbers in a strictly increasing manner.

> **Example 4.22** Consider the pMC in Figure 4.9 with valuation $\{x, y \mapsto 1/2\}$. We rank s_3 the highest. State s_2 may be considered closer to the target than s_1. Formally, we rank s_2 higher than s_1. State s_2 has a transition to s_3, which is higher ranked, likewise for s_1, there is a transition to s_2 which is ranked higher. For s_0, the only successor state is s_0: A state cannot be ranked strictly higher than itself, thus there cannot be a path to the target.
> Now consider $\{x, y \mapsto 1\}$. For s_1 the only successor is s_2. If there is a path to the target, then s_2 must be ranked higher than s_1. For s_2, the only successor is s_1, thus also s_1 must be ranked higher than s_2. These requirements are contradicting and we conclude that there is no path from either states s_1, s_2 to the target.

The following encoding builds upon this idea, inspired by [WJÁK+14].

Constraints 4.3 Let $\mathcal{D}$ be a pMC with states S. Consider the Boolean variables $\{q_s \mid s \in S\}$, the real variables $\{r_s \mid s \in S\}$, and the parameter variables X.

$$
\begin{aligned}
&q_s && \text{for all } s \in T,\\
&q_s \leftrightarrow \bigvee_{s' \in S} \left(\mathcal{P}(s, s') \gtrdot 0 \wedge q_{s'} \wedge r_s \lessdot r_{s'}\right) && \text{for all } s \in S \setminus T.
\end{aligned}
$$

Intuitively: the variables r_s encode the rank—as described above—and the variables q_s are `true` when the reachability to the target is positive for some (fixed) valuation to X. More precisely, for any fixed values of X: a variable q_s is `true` if there is a path to the target in $\mathcal{D}[\mathsf{val}]$. In terms of the encoding, q_s is `true` iff there is a transition to a state s' which has a path to the target and is ranked higher than s. Equivalently, there is a path from s' to the target (and that path does not visit s).

> **Example 4.23** Applying the encoding from Constraints 4.3 to the pMC in Figure 4.9 yields:
>
> $$
> \begin{aligned}
> &q_3 \wedge \left(q_0 \leftrightarrow 1 \gtrdot 0 \wedge q_0 \wedge r_0 \lessdot r_0\right)\\
> &\wedge \Big(q_1 \leftrightarrow \big((1 - x \gtrdot 0 \wedge q_0 \wedge r_1 \lessdot r_0) \vee (x \gtrdot 0 \wedge q_2 \wedge r_1 \lessdot r_2)\big)\Big)\\
> &\wedge \Big(q_2 \leftrightarrow \big((y \gtrdot 0 \wedge q_1 \wedge r_2 \lessdot r_1) \vee (1 - y \gtrdot 0 \wedge q_3 \wedge r_2 \lessdot r_3)\big)\Big).
> \end{aligned}
> $$

For the zero-states, we have to negate the positive reachability variables q_s.

Proposition 4.9 The encoding in Constraints 4.3 yields for any satisfying assignment val and any state $s \in S$:

$$\mathsf{val}(q_s) = \texttt{false} \quad \text{iff} \quad s \in S_{=0}^{\mathsf{val}_{\downarrow X}, T}.$$

Almost sure reachability | To almost surely reach the target T, there cannot be a path to a bad state, that is, every path must be extensible such that the target states are reached. Equivalently, all cycle-free paths lead to the target. We describe this inductively as follows: Clearly, we reach the target states almost surely from the target states. For all other states, we almost surely reach T from state s if there is at least one path to T, and after taking any transition, we still almost surely reach the target. Concretely, starting in s, for every successor s', the target is also almost surely reachable from s'. These ideas are formalised in the following encoding.

Constraints 4.4 Let $\mathcal{D}$ be a pMC with states S. Consider the Boolean variables $\{q_s, w_s \mid s \in S\}$, the real variables $\{r_s \mid s \in S\}$, and the parameter variables X.

$$\begin{aligned}
&q_s && \text{for all } s \in T,\\
&q_s \leftrightarrow \bigvee_{s' \in S} (\mathcal{P}(s,s') \gtrdot 0 \wedge q_{s'} \wedge r_s \lessdot r_{s'}) && \text{for all } s \in S \setminus T,\\
&w_s && \text{for all } s \in T,\\
&w_s \leftrightarrow q_s \wedge \bigwedge_{s' \in S} (\mathcal{P}(s,s') \gtrdot 0 \rightarrow w_{s'}) && \text{for all } s \in S \setminus T.
\end{aligned}$$

We denote the corresponding formula with $\Phi^{\mathbb{P}_{=1}}(\mathcal{D})$.

The variables q_s and r_s are as before. The variables w_s encode almost-sure reachability from a state s. For any fixed values of X: a variable w_s is `true` if all cycle-free paths from s lead to the target. Using the encoding, w_s is `true` iff for all transitions we reach a state from which all cycle-free paths lead to the target. In particular, there has to be at least one path to the target.

Example 4.24 We consider the pMC in Figure 4.9 and instantiate the encoding from Constraints 4.4 by extending the encoding from Example 4.23 with:

$$\begin{aligned}
&w_3 \wedge \Big(w_0 \leftrightarrow (q_0 \wedge 1 \gtrdot 0 \rightarrow w_0)\Big)\\
&\wedge \Big(w_1 \leftrightarrow \big(q_1 \wedge (1 - x \gtrdot 0 \rightarrow w_0) \wedge (x \gtrdot 0 \rightarrow w_2)\big)\Big)\\
&\wedge \Big(w_2 \leftrightarrow \big(q_2 \wedge (y \gtrdot 0 \rightarrow w_1) \wedge (1 - y \gtrdot 0 \rightarrow w_3)\big)\Big).
\end{aligned}$$

The one-states may be determined straightforwardly using the following identity.

Proposition 4.10 The encoding in Constraints 4.4 yields for any satisfying assignment val and any state $s \in S$:

$$\mathsf{val}(w_s) = \mathtt{true} \quad \text{iff} \quad s \in S_{=1}^{\mathsf{val}_{\downarrow X}, T}.$$

Example 4.25 We include a complete (but slightly simplified) encoding for Example 4.21 on page 83 and instantiate the encoding from Constraints 4.4.

$$\begin{aligned}
& q_6 \wedge \Big(q_0 \leftrightarrow \big(q_0 \wedge r_0 < r_0\big)\Big) \wedge \Big(q_1 \leftrightarrow \big(q_2 \wedge r_1 < r_2\big) \vee \big(q_4 \wedge r_1 < r_4\big)\Big) \\
& \wedge \Big(q_2 \leftrightarrow \big(1 - x \gtrdot 0 \wedge q_0 \wedge r_2 < r_0\big) \vee \big(x \gtrdot 0 \wedge q_3 \wedge r_2 < r_3\big)\Big) \\
& \wedge \Big(q_3 \leftrightarrow \big(1 - x \gtrdot 0 \wedge q_2 \wedge r_3 < r_2\big) \vee \big(x \gtrdot 0 \wedge q_6 \wedge r_3 < r_6\big)\Big) \\
& \wedge \Big(q_4 \leftrightarrow \big(x \gtrdot 0 \wedge q_5 \wedge r_4 < r_5\big)\Big) \wedge \Big(q_5 \leftrightarrow \big(1 - x \gtrdot 0 \wedge q_6 \wedge r_5 < r_6\big)\Big) \\
& \wedge w_6 \wedge \Big(w_0 \leftrightarrow q_0 \wedge w_0\Big) \wedge \Big(w_1 \leftrightarrow q_1 \wedge w_2 \wedge w_4\Big) \\
& \wedge \Big(w_2 \leftrightarrow q_2 \wedge (1 - x \gtrdot 0 \rightarrow w_0) \wedge (x \gtrdot 0 \rightarrow w_3)\Big) \\
& \wedge \Big(w_3 \leftrightarrow q_3 \wedge (1 - x \gtrdot 0 \rightarrow w_2) \wedge (x \gtrdot 0 \rightarrow w_6)\Big) \\
& \wedge \Big(w_4 \leftrightarrow q_4 \wedge (x \gtrdot 0 \rightarrow w_5)\Big) \wedge \Big(w_5 \leftrightarrow q_5 \wedge (1 - x \gtrdot 0 \rightarrow w_6)\Big).
\end{aligned}$$

An alternative, incremental approach based on an encoding for POMDPs and finite memory strategies, as in [CCD16], may be beneficial for empirical performance of solvers. The relation between POMDPs and pMCs is discussed in Section 7.1.

4.3.2.2 Encoding reachability probabilities

We move from the qualitative setting to a quantitative one. The principle is again to generalise the parameter-free case, in particular, the linear equation formulation in Section 3.3.2.1. We develop the encoding in two steps: First, we consider an encoding only valid for graph-preserving valuations. In particular, it requires the zero-states to be known a-priori. Later, we combine this encoding with the earlier qualitative encodings to compute the zero-states on the fly.

Graph-preserving case | Recall from Constraints 3.1 on page 52 the equation system for MCs. We trivially lift this equation system to pMCs. Again, for technical reasons, we first restrict ourselves to polynomial pMCs.

Constraints 4.5 Let $\mathcal{D}$ be a polynomial pMC. We assume a graph-preserving region $R \subseteq \mathsf{Val}^{\mathrm{gp}}_{\mathcal{D}}$. Consider real variables $\{p_s \mid s \in S\}$ and variables for the parameters X:

$$
\begin{aligned}
p_s &\doteq 1 && \text{for all } s \in T,\\
p_s &\doteq 0 && \text{for all } s \in S^{R,T}_{=0},\\
p_s &\doteq \sum_{s' \in S} \mathcal{P}(s,s') \cdot p_{s'} && \text{for all } s \in S \setminus \left(T \cup S^{R,T}_{=0}\right).
\end{aligned}
$$

We denote the corresponding formula with $\Phi^{\mathbb{P}}_{\mathrm{gp}}(\mathcal{D})$.

The constraints are essentially identical to the constraints for parameter-free MCs. The key difference is that the transition probabilities are no longer constants. Therefore (in general[1]) the encodings are non-linear.

Example 4.26 Reconsider the pMC $\mathcal{D}$ from Figure 4.9. The equation system for reaching $T = \{s_3\}$, using p_i to denote p_{s_i}, is given by:

$$
\begin{aligned}
p_0 &\doteq 0, & p_1 &\doteq x \cdot p_2 + (1-x) \cdot p_0,\\
p_3 &\doteq 1, & p_2 &\doteq y \cdot p_1 + (1-y) \cdot p_3.
\end{aligned}
$$

Proposition 4.11 Let $\mathcal{D}$ be a polynomial pMC. Let $\mathsf{val}' \in \mathsf{sat}(\Phi^{\mathbb{P}}_{\mathrm{gp}}(\mathcal{D})) \cap \mathsf{Val}^{\mathrm{gp}}_{\mathcal{D}}$ and $\mathsf{val} := \mathsf{val}'_{\downarrow X}$. For any state $s \in S$:

$$\mathsf{val}'(p_s) \quad = \quad Pr_{\mathcal{D}[\mathsf{val}]}(s \models \Diamond T).$$

By construction, the constraints are satisfiable. We consider a simplification of this system in Section 4.3.4, where we infer the solution function from these equations. Analogously to Section 4.3.1.2 on page 81, if the transitions in the pMC are not polynomial but rational functions, then the equations are not polynomial constraints, hence their conjunction is not a formula. Instead, each $p_s = \sum \mathcal{P}(s,s')$ has to be transformed by the rules in Section 4.3.1.2 such that their conjunction is a formula. This transformation may always be applied, as we argue here: Recall that we are interested in analysing this equation system on a *graph-preserving*[2] parameter region R. Therefore, for any $\mathsf{val} \in R$, $\mathcal{P}(s,s')[\mathsf{val}] \in (0,1)$ for each $s, s' \in S$. Thus, when $\Phi^{\mathbb{P}}_{\mathrm{gp}}(\mathcal{D})$ is used in conjunction with ψ_R, we do not need to consider this special case of division by zero. We refrain from formalising the construction for non-polynomial pMCs, but do give this small example.

[1] The notable exceptions are systems where the parameters only occur in states where *all* successor states are sink- or target-states.

[2] The argument may be adapted for well-defined valuations and the encoding presented later.

Example 4.27 Recall the pMC from Figure 4.7 on page 81, which has a state with outgoing probabilities y and $x/_{1+x}$. In Example 4.20 we already inferred that x must be non-negative (otherwise, $x/_{1+x} > 1$). As an encoding, we obtain:

$$p_0 \doteq 0 \land p_2 \doteq 1 \land p_1 \cdot (1 + x) \doteq y \cdot (1 + x) \cdot p_0 + x \cdot p_2.$$

To check feasibility in graph-preserving regions, we have to restrict the parameter valuations accordingly and encode that the induced probability in the initial state compares $\bowtie \lambda$. We add these constraints and obtain:

Theorem 4.12 Let $\mathcal{D}$ be a polynomial pMC with target states T and let $R \subseteq \mathsf{Val}^{\mathrm{gp}}_{\mathcal{D}}$ be a region. Furthermore, let $\varphi = \mathbb{P}_{\bowtie\lambda}(\Diamond T)$. We define

$$\psi \coloneqq \Phi^{\mathbb{P}}_{\mathrm{gp}}(\mathcal{D}) \land p_\iota \bowtie \lambda \land \psi_R.$$

Then, for any $\mathsf{val} \in \mathsf{Val}$,

$$\mathsf{val} \in \mathsf{sat}(\psi) \quad \text{iff} \quad \mathcal{D}[\mathsf{val}_{\downarrow X}] \models \varphi.$$

Example 4.28 Reconsider the pMC from Figure 4.9 with $\mathbb{P}_{\geq \frac{1}{2}}(\Diamond\{s_3\})$. The constraint system using p_i to denote p_{s_i} is given by:

$$\begin{aligned} &p_1 \geqslant 1/2, && 0 < x < 1 \land 0 < y < 1, \\ &p_0 \doteq 0, && p_1 \doteq x \cdot p_2 + (1{-}x) \cdot p_0, \\ &p_3 \doteq 1, && p_2 \doteq y \cdot p_1 + (1{-}y) \cdot p_3. \end{aligned}$$

One satisfying assignment is

$$\{x \mapsto 3/4, y \mapsto 1/2, p_0 \mapsto 0, p_1 \mapsto 3/5, p_2 \mapsto 4/5, p_3 \mapsto 1\},$$

whereas the assignment

$$\{x, y, p_1, p_2, p_3 \mapsto 1, p_0 \mapsto 0\}$$

is not satisfying (it is not a graph-preserving valuation), and the assignment

$$\{x, y \mapsto 1/2, p_0 \mapsto 0, p_1 \mapsto 1/3, p_2 \mapsto 2/3, p_3 \mapsto 1\}$$

is not satisfying (the probability to reach the target is less than a half).

General case | We extend the encoding to feasibility for any (well-defined) parameter region. An essential assumption before was that the set of zero-states is fixed and may

be precomputed. This assumption is no longer valid for general valuations. We thus compute the zero-states using the encoding for positive reachability.

Constraints 4.6 Let $\mathcal{D}$ be a polynomial pMC with states S. Consider Boolean variables $\{q_s \mid s \in S\}$, real variables $\{p_s, r_s \mid s \in S\}$, and variables for the parameters:

$$
\begin{aligned}
&p_s = 1 && \text{for all } s \in T,\\
&q_s && \text{for all } s \in T,\\
&q_s \leftrightarrow \bigvee_{s' \in S} (\mathcal{P}(s,s') \gtrdot 0 \wedge (q_{s'} \wedge r_s \lessdot r_{s'})) && \text{for all } s \in S \setminus T,\\
&\neg q_s \rightarrow p_s = 0 && \text{for all } s \in S \setminus T,\\
&q_s \rightarrow p_s \doteq \sum_{s' \in S} \mathcal{P}(s,s') \cdot p_{s'} && \text{for all } s \in S \setminus T.
\end{aligned}
$$

We denote the corresponding formula with $\Phi^{\mathbb{P}}_{\mathrm{wd}}(\mathcal{D})$.

The meaning of the variables is as before: The variables q_s determine whether we have to compute the non-zero probability to the target or whether this probability is zero. The r_s variables are auxiliary variables ranking the states. The specialised constraints for the graph-preserving case (Constraints 4.5) are obtained by setting all variables of non-zero states q_s to `true`. The following theorem is the analogue to Theorem 4.12.

Theorem 4.13 Let $\mathcal{D}$ be a polynomial pMC with target states T and let $R \subseteq \mathsf{Val}^{\mathrm{wd}}_{\mathcal{D}}$ be a region. Furthermore, let $\varphi = \mathbb{P}_{\bowtie\lambda}(\Diamond T)$. We define

$$\psi := \Phi^{\mathbb{P}}_{\mathrm{wd}}(\mathcal{D}) \wedge p_\iota \bowtie \lambda \wedge \psi_R.$$

Then, for any $\mathsf{val} \in \mathsf{Val}$:

$$\mathsf{val} \in \mathsf{sat}(\psi) \quad \text{iff} \quad \mathcal{D}[\mathsf{val}_{\downarrow X}] \models \varphi.$$

Example 4.29 Reconsider the pMC from Figure 4.9 with $\mathbb{P}_{\geq \frac{1}{2}}(\Diamond\{s_3\})$. We present ψ from Theorem 4.13. We prune the encoding for conciseness and obtain:

$$
\begin{aligned}
&p_1 \geqslant \frac{1}{2} \wedge 0 \lessdot x \leqslant 1 \wedge 0 \leqslant y \leqslant 1 \wedge p_3 \doteq 1 \wedge q_3,\\
&\neg q_0 \wedge \big(q_1 \leftrightarrow (x \gtrdot 0 \wedge q_2)\big) \wedge \big(q_2 \leftrightarrow 1 - y \gtrdot 0\big),\\
&\neg q_0 \rightarrow p_0 \doteq 0 \wedge q_0 \rightarrow p_0 \doteq p_0,\\
&\neg q_1 \rightarrow p_1 \doteq 0 \wedge q_1 \rightarrow \big(p_1 \doteq x \cdot p_2 + (1-x) \cdot p_0\big),\\
&\neg q_2 \rightarrow p_2 \doteq 0 \wedge q_2 \rightarrow \big(p_2 \doteq y \cdot p_1 + (1-y) \cdot p_3\big).
\end{aligned}
$$

One satisfying assignment is (as before)

$$\{x \mapsto 3/4, y \mapsto 1/2, p_0 \mapsto 0, p_1 \mapsto 3/5, p_2 \mapsto 4/5, p_3 \mapsto 1,$$
$$q_0 \mapsto \texttt{false}, q_1, q_2, q_3 \mapsto \texttt{true}\},$$

another satisfying assignment is

$$\{x \mapsto 1, y \mapsto 0, p_0 \mapsto 0, p_1, p_2, p_3 \mapsto 1, q_0 \mapsto \texttt{false}, q_1, q_2, q_3 \mapsto \texttt{true}\}.$$

The assignment

$$\{x, y, p_1, p_2, p_3 \mapsto 1, p_0 \mapsto 0, q_0 \mapsto \texttt{false}, q_1, q_2, q_3 \mapsto \texttt{true}\}$$

is not satisfying (it does not correctly identify the states with positive reachability), likewise

$$\{x, y, p_1, p_2, p_3 \mapsto 1, p_0 \mapsto 0, q_0, q_1, q_2 \mapsto \texttt{false}, q_3 \mapsto \texttt{true}\}$$

is not satisfying (it does not adhere to the constraints for zero-states).

4

Rewards | For expected rewards, one may again handle graph-preserving regions separately, where the almost-sure reachability of the target states may be assured globally. Here, we immediately consider the general case.

Constraints 4.7 Let $\mathcal{D}$ be a polynomial pMC with states S. Consider real variables $\{p_s \mid s \in S\}$ along with variables for the parameters X:

$$p_s \doteq 0 \qquad \text{for all } s \in T,$$
$$p_s \doteq \mathsf{prew}(s) + \sum_{s' \in S} \mathcal{P}(s, s') \cdot p_{s'} \qquad \text{for all } s \in S \setminus T.$$

We denote the corresponding formula with $\Phi^{\mathbb{E}}_{\mathrm{wd}}(\mathcal{D})$.

The encoding generalises the expected reward from parameter-free to parametric MCs completely analogous to the setting for reachability probabilities:

Proposition 4.14 Let $\mathcal{D}$ be a polynomial pMC. Let $\mathsf{val}' \in \mathsf{sat}(\Phi^{\mathbb{P}}_{\mathrm{gp}}(\mathcal{D})) \cap \mathsf{Val}^{\mathrm{gp}}_{\mathcal{D}}$ and $\mathsf{val} := \mathsf{val}'_{\downarrow X}$. For any state $s \in S$ with $s \in S^{R,T}_{=1}$:

$$\mathsf{val}'(p_s) \quad = \quad Er_{\mathcal{D}[\mathsf{val}]}(s \models \Diamond T).$$

Observe that the constraints only correctly describe the rewards if the expected reward is finite, i.e., $s \in S^{R,T}_{=1}$. Therefore, the following theorem uses the encoding for almost-

sure reachability in Constraints 4.4. Furthermore, an infinite reward is satisfying any finite lower bounds on the reward, therefore, we adapt the encoding depending on the comparison relation:

Theorem 4.15 Let $\mathcal{D}$ be a pMC with target states T and region $R \subseteq \mathsf{Val}^{\mathrm{wd}}_{\mathcal{D}}$. Let $\varphi = \mathbb{E}_{\bowtie\lambda}(\Diamond T)$. We distinguish two cases:

- Either $\bowtie \in \{\leq, <\}$. We define
$$\psi := \Phi^{\mathbb{E}}_{\mathrm{wd}}(\mathcal{D}) \wedge \Phi^{\mathbb{P}_{=1}}(\mathcal{D}) \wedge p_\iota \bowtie \lambda \wedge w_\iota \wedge \psi_R,$$
- or $\bowtie \in \{\geq, >\}$, and we define
$$\psi := \Phi^{\mathbb{E}}_{\mathrm{wd}}(\mathcal{D}) \wedge \Phi^{\mathbb{P}_{=1}}(\mathcal{D}) \wedge \big(p_\iota \bowtie \lambda \vee \neg w_\iota\big) \wedge \psi_R.$$

Then, for any $\mathsf{val} \in \mathsf{Val}$,
$$\mathsf{val} \in \mathsf{sat}(\psi) \quad \text{iff} \quad \mathcal{D}[\mathsf{val}_{\downarrow X}] \models \varphi.$$

It suffices to check almost-sure reachability from the initial state: If the initial state almost surely reaches the target, then this extends also to any reachable state.

Example 4.30 Reconsider the pMC from Figure 4.9 with
$$\mathsf{prew} = \{s_0, s_3 \mapsto 0, s_1 \mapsto 1, s_2 \mapsto x\}.$$
For $\Phi^{\mathbb{P}_{=1}}(\mathcal{D})$, we use the simplified encoding of Example 4.24:
$$q_3 \wedge w_3 \wedge \neg q_0 \wedge \neg w_0 \wedge w_2 \leftrightarrow y \doteq 0 \wedge q_2 \leftrightarrow y \neq 1$$
$$q_1 \leftrightarrow \big(x \neq 0 \wedge q_2\big) \wedge \Big(w_1 \leftrightarrow \big(q_1 \wedge x \doteq 0 \wedge w_2\big)\Big).$$
In particular, observe that $x \neq 1$ immediately leads to w_1 being `false` in any satisfying assignment. For $\Phi^{\mathbb{E}}_{\mathrm{wd}}(\mathcal{D})$, we obtain:
$$p_0 \doteq p_3 \doteq 0 \wedge p_1 \doteq 1 + (1-x)\cdot p_0 + x \cdot p_2 \wedge p_2 \doteq x + y \cdot p_1 + (1-y)\cdot p_3.$$
First, let $\varphi := \mathbb{E}_{\leq \frac{1}{2}}(\Diamond\{s_3\})$. We obtain the first case:
$$\Phi^{\mathbb{E}}_{\mathrm{wd}}(\mathcal{D}) \wedge \Phi^{\mathbb{P}_{=1}}(\mathcal{D}) \wedge p_1 \leqslant 1/2 \wedge w_1 \wedge 0 \lessdot x \lessdot 1 \wedge 0 \leqslant y \leqslant 1.$$
Now, let $\varphi := \mathbb{E}_{> \frac{1}{2}}(\Diamond\{s_3\})$. We obtain the second case.
$$\Phi^{\mathbb{E}}_{\mathrm{wd}}(\mathcal{D}) \wedge \Phi^{\mathbb{P}_{=1}}(\mathcal{D}) \wedge \big(p_1 \geqslant 1/2 \vee \neg w_1\big) \wedge 0 \leqslant x \leqslant 1 \wedge 0 \leqslant y \leqslant 1.$$

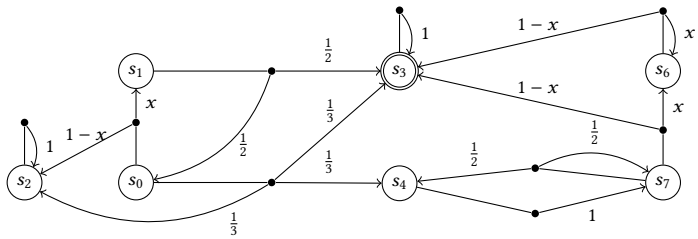


Figure 4.10: Example pMDP

4.3.3 Encoding pMDP feasibility

In this section, we generalise the encodings from pMCs to pMDPs. We distinguish between angelic and demonic nondeterminism. For conciseness, we omit the encodings for the graph-preserving special case, for almost-sure reachability, and for expected rewards. They are obtained by a careful but straightforward combination of the presented constructions. Together, this subsection establishes that for every pMDP and every (considered) property, the sets of accepting (rejecting) valuations are regions.

4.3.3.1 Qualitative analysis

Again, we first consider the qualitative case.

Definition 4.17 Let $\mathcal{M}$ be a pMDP. The *exist-zero states* for a valuation $\mathsf{val} \in \mathsf{Val}^{\mathrm{wd}}_{\mathcal{M}}$ is the set

exist-zero states

$$S^{\mathsf{val},T}_{\exists=0} := \{s \in S \mid \exists \sigma \in \Sigma^{\mathcal{M}} \text{s.t. } {}^{\mathbb{P}}\mathsf{sol}^{\mathcal{M}[\sigma]}_{s\to T}[\mathsf{val}] = 0\}$$

containing the states that reach the target with probability zero in instantiation val, and the *exist-one states* is the set

exist-one states

$$S^{\mathsf{val},T}_{\exists=1} := \{s \in S \mid \exists \sigma \in \Sigma^{\mathcal{M}} \text{s.t. } {}^{\mathbb{P}}\mathsf{sol}^{\mathcal{M}[\sigma]}_{s\to T}[\mathsf{val}] = 1\}.$$

The sets $S^{\mathsf{val},T}_{\forall=0}$ and $S^{\mathsf{val},T}_{\forall=1}$ are defined analogously.

Example 4.31 Consider the pMDP in Figure 4.10, with $T = \{s_3\}$, and s_0 and s_7 as only nondeterministic states. First, assume $\mathsf{val} := \{x \mapsto 1/2\}$. We have

$$S^{\mathsf{val},T}_{\exists=0} = \{s_2, s_4, s_7\}, \text{ and } S^{\mathsf{val},T}_{\forall=0} = \{s_2\}.$$

For $\mathsf{val} := \{x \mapsto 0\}$, we have

$$S^{\mathsf{val},T}_{\exists=0} = \{s_0, s_2, s_4, s_7\}, \text{ and } S^{\mathsf{val},T}_{\forall=0} = \{s_2\},$$

and for $\mathsf{val} := \{p \mapsto 1\}$, we have

$$S^{\mathsf{val},T}_{\exists=0} = \{s_2, s_4, s_6, s_7\} = S^{\mathsf{val},T}_{\forall=0}.$$

Angelic nondeterminism | For angelic strategies, a state has a positive probability to reach the target iff there is some action that leads to at least one successor which has a positive probability to reach the target and is closer to the target.

Constraints 4.8 (Positive reachability under angelic nondeterminism) Let $\mathcal{M}$ be a pMDP with states S. Consider the Boolean variables $\{q_s \mid s \in S\}$ and the real variables $\{r_s \mid s \in S\}$, together with the parameter variables X.

$$\begin{aligned} & q_s && \text{for all } s \in T, \\ & q_s \leftrightarrow \bigvee_{\alpha \in \mathbb{A}\mathrm{ct}(s)} \bigvee_{s' \in S} (\mathcal{P}(s, \alpha, s') \gtrdot 0 \rightarrow (q_{s'} \wedge r_s \lessdot r_{s'})) && \text{for all } s \in S \setminus T. \end{aligned}$$

Observe that we thus get to choose the parameters and the action with which to satisfy the constraint for a state. The disjunction is at each state locally, rather than a big disjunction over all possible strategies.

Lemma 4.16 For any $\mathcal{M}[\mathsf{val}]$, the encoding in Constraints 4.8 yields for any satisfying assignment val' and any state $s \in S$:

$$\mathsf{val}'(q_s) = \texttt{true} \quad \text{iff} \quad \text{exists } \sigma \in \Sigma^{\mathcal{M}} \text{s.t. } Pr^{\sigma}_{\mathcal{M}[\mathsf{val}]}(s \models \Diamond T) > 0.$$

To compute the zero-states via this encoding, care is necessary upon negation: The variable q_s is assigned `false`, iff for all strategies, the probability of reaching the target is zero. Formally:

Corollary 4.17 For any $\mathcal{D}[\mathsf{val}]$, the encoding in Constraints 4.8 yields for any satisfying assignment val' and any state $s \in S$:

$$\mathsf{val}'(q_s) = \texttt{false} \quad \text{iff} \quad s \in S^{\mathsf{val},T}_{\forall=0}.$$

Demonic nondeterminism | For demonic strategies, the probability to reach the target is positive iff for any action, there is some successor—reached by taking this action—that has a positive probability to reach the target and is closer to the target.

Constraints 4.9 (Positive reachability under demonic nondeterminism) Let $\mathcal{M}$ be a pMDP with states S. Consider the Boolean variables $\{q_s \mid s \in S\}$ and the real variables $\{r_s \mid s \in S\}$, together with the parameter variables X.

$$
\begin{aligned}
& q_s && \text{for all } s \in T, \\
& q_s \leftrightarrow \bigwedge_{\alpha \in \mathbb{A}\mathrm{ct}(s)} \bigvee_{s' \in S} (\mathcal{P}(s, \alpha, s') \gtrdot 0 \rightarrow (q_{s'} \wedge r_s \lessdot r_{s'})) && \text{for all } s \in S \setminus T.
\end{aligned}
$$

As above, there is an implicit negation necessary to compute the zero-states. We obtain the following analogue to Corollary 4.17.

Proposition 4.18 For any $M[\mathsf{val}]$, the encoding in Constraints 4.9 yields for any satisfying assignment val' and any state $s \in S$:

$$\mathsf{val}'(q_s) = \texttt{false} \quad \text{iff} \quad s \in S^{\mathsf{val},T}_{\exists=0}.$$

We omit the encodings for $S^{\mathsf{val},T}_{\exists=1}$ and $S^{\mathsf{val},T}_{\forall=1}$.

4

4.3.3.2 Reachability probabilities

For pMDPs, we omit the special case of graph-preservation. Instead, we consider angelic and demonic nondeterminism separately. Contrary to pMCs, but in line with parameter-free MDPs, we now also have to distinguish properties with lower bounds and properties with upper bounds.

Angelic nondeterminism | Angelic nondeterminism is conceptually simple, as we existentially quantify over both parameter values and strategies. In a game-theoretic sense, one player chooses both parameter values and strategies, and we may just generalise the pMC encoding and use the ETR (where the player selects the values for all variables). We may, however, avoid variables for the strategies by observing that the quantification over strategies is over a finite set, and that this choice may be represented by a (finite) disjunction. This disjunction ranges over exponentially many strategies. We avoid this explicit blowup by recalling that the nondeterminism is resolved locally. Instead of a disjunction over all strategies, we make disjunctions over the local action choices, similar to the encoding of the qualitative case. These insights yield a compact encoding, detailed below.

Constraints 4.10 (Upper-bounded reachability, angelic nondeterminism)

$$
\begin{aligned}
& p_s \doteq 1 && \text{for all } s \in T, \\
& q_s && \text{for all } s \in T, \\
& q_s \leftrightarrow \bigwedge_{\alpha \in \mathbb{A}\mathrm{ct}(s)} \bigvee_{s' \in S} (\mathcal{P}(s,\alpha,s') \gtrdot 0 \rightarrow (q_{s'} \wedge r_s \lessdot r_{s'})) && \text{for all } s \in S \setminus T, \\
& \neg q_s \rightarrow p_s \doteq 0 && \text{for all } s \in S \setminus T, \\
& q_s \rightarrow \bigvee_{\alpha \in \mathbb{A}\mathrm{ct}(s)} \left(p_s \doteq \sum_{s' \in S} \mathcal{P}(s,\alpha,s') \cdot p_{s'} \right) && \text{for all } s \in S \setminus T.
\end{aligned}
$$

We refer to the corresponding formula as $\Phi_{\mathrm{wd}}^{\mathbb{P},\trianglelefteq,\exists}(\mathcal{M})$.

Under angelic nondeterminism, we can freely choose the action at every state: the probability of reaching the target is the sum over the probabilities of reaching the target from the successors after taking this action. As we can choose the action, we thus have a disjunction over equalities for every state and these disjunctions are guarded by the flag that the probability is positive from this state, as for pMCs. For upper bounds on the reachability probability, under angelic nondeterminism, the strategy tries to minimise the probability. In particular, the strategies sets states to probability zero if there is any strategy to do so. As the interpretation of the q_s variables is positive reachability, i.e., q_s is `true` iff it is not a zero-state, we obtain a conjunction over all actions in the encoding.
Below, we give the encoding for lower bounds on the probability.

Constraints 4.11 (Lower-bounded reachability, angelic nondeterminism)

$$
\begin{aligned}
& p_s \doteq 1 && \text{for all } s \in T, \\
& q_s && \text{for all } s \in T, \\
& q_s \leftrightarrow \bigvee_{\alpha \in \mathbb{A}\mathrm{ct}(s)} \bigvee_{s' \in S} (\mathcal{P}(s,\alpha,s') \gtrdot 0 \rightarrow (q_{s'} \wedge r_s \lessdot r_{s'})) && \text{for all } s \in S \setminus T, \\
& \neg q_s \rightarrow p_s \doteq 0 && \text{for all } s \in S \setminus T, \\
& q_s \rightarrow \bigvee_{\alpha \in \mathbb{A}\mathrm{ct}(s)} \left(p_s \doteq \sum_{s' \in S} \mathcal{P}(s,\alpha,s') \cdot p_{s'} \right) && \text{for all } s \in S \setminus T.
\end{aligned}
$$

We refer to the corresponding formula as $\Phi_{\mathrm{wd}}^{\mathbb{P},\trianglerighteq,\exists}(\mathcal{M})$.

For lower bounds, (only) the computation of the zero states changes, as we now try to avoid setting a state to probability zero. Thus, we only set the reachability probability

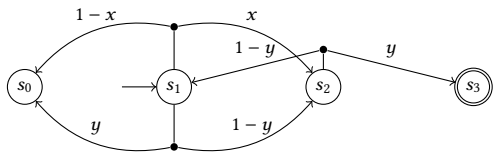


Figure 4.11: Example pMDP

to zero if all actions lead to zero-states. Again, as the interpretation of q_s is that s is not a zero-state, we obtain a disjunction over the actions.

Theorem 4.19 Let $\mathcal{M}$ be a pMDP with target states T. Let $R \subseteq \mathsf{Val}^{\mathrm{wd}}_{\mathcal{M}}$ be a region and let $\varphi = \mathbb{P}_{\bowtie\lambda}(\Diamond T)$. We distinguish two cases:

- Either $\bowtie \in \{\leq, <\}$. We define
$$\psi := \Phi^{\mathbb{P},\trianglelefteq,\exists}_{\mathrm{wd}}(\mathcal{M}) \wedge p_\iota \bowtie \lambda \wedge \psi_R,$$
- or $\bowtie \in \{\geq, >\}$, and we define
$$\psi := \Phi^{\mathbb{P},\trianglerighteq,\exists}_{\mathrm{wd}}(\mathcal{M}) \wedge p_\iota \bowtie \lambda \wedge \psi_R.$$

Then, for any $\mathsf{val} \in \mathsf{Val}$,

$$\mathsf{val} \in \mathsf{sat}(\psi) \quad \text{iff} \quad \mathcal{M}[\mathsf{val}_{\downarrow X}] \,{}^{\exists}\!\models \varphi.$$

Example 4.32 Consider the pMDP in Figure 4.11. For conciseness, we simplified several constraints. We first consider the encoding for $\varphi := \mathbb{P}_{\leq 1/2}(\Diamond T)$.

$$p_1 \leqslant 1/2 \wedge p_0 \doteq 0 \wedge \neg q_0 \wedge p_3 \doteq 1 \wedge q_3,$$
$$q_1 \leftrightarrow \Big(\big(x \gtrdot 0 \wedge q_2 \wedge r_1 \lessdot r_2\big) \wedge \big(1 - y \gtrdot 0 \wedge q_2 \wedge r_1 \lessdot r_2\big)\Big),$$
$$q_2 \leftrightarrow \Big(\big(y \gtrdot 0\big) \vee \big(1 - y \gtrdot 0 \wedge q_1 \wedge q_2 \lessdot q_1\big)\Big),$$
$$q_1 \rightarrow \Big(\big(p_1 \doteq x \cdot p_2\big) \vee \big(p_1 \doteq (1 - y) \cdot p_2\big)\Big) \wedge \neg q_1 \rightarrow p_1 \doteq 0,$$
$$q_2 \rightarrow \big(p_2 \doteq (1 - y) \cdot p_1 + y\big) \wedge \neg q_1 \rightarrow p_2 \doteq 0.$$

Below, we give the encoding for $\varphi := \mathbb{P}_{\geq 1/2}(\Diamond T)$. Compared to the encoding above, only the first constraint and one further connective in the second line changed.

$$
\begin{aligned}
&p_1 \geqslant 1/2 \wedge p_0 \doteq 0 \wedge \neg q_0 \wedge p_3 \doteq 1 \wedge q_3,\\
&q_1 \leftrightarrow \Big(\big(x \gtrdot 0 \wedge q_2 \wedge r_1 \lessdot r_2\big) \vee \big(1 - y \gtrdot 0 \wedge q_2 \wedge r_1 \lessdot r_2\big)\Big),\\
&q_2 \leftrightarrow \Big(\big(y \gtrdot 0\big) \vee \big(1 - y \gtrdot 0 \wedge q_1 \wedge q_2 \lessdot q_1\big)\Big),\\
&q_1 \rightarrow \Big(\big(p_1 \doteq x \cdot p_2\big) \vee \big(p_1 \doteq (1 - y) \cdot p_2\big)\Big) \wedge \neg q_1 \rightarrow p_1 \doteq 0,\\
&q_2 \rightarrow \big(p_2 \doteq (1 - y) \cdot p_1 + y\big) \wedge \neg q_1 \rightarrow p_2 \doteq 0.
\end{aligned}
$$

Demonic nondeterminism | For the demonic case, we existentially quantify over parameter values and universally over strategies. In a game-theoretic sense, we would need two players, which would yield a formula not captured by the ETR. The insight again is that the universal quantification is over a finite domain and may therefore be turned in a conjunction, analogously to the angelic case above. However, when applying the conjunction locally at the states, we have to ensure that we do not expect all equalities to hold simultaneously. Instead, we adapt the LP encoding of the Bellman inequations on parameter-free MDPs, cf. Section 3.3.2.2 on page 53, which leads to an ETR encoding. All further ideas are then straightforward analogues. Naturally we have to change the zero-states to the demonic case.

Constraints 4.12 (Upper-bounded reachability, demonic nondeterminism)

$$
\begin{aligned}
&p_s \doteq 1 && \text{for all } s \in T,\\
&q_s && \text{for all } s \in T,\\
&q_s \leftrightarrow \bigvee_{\alpha \in \mathbb{A}\mathrm{ct}(s)} \bigvee_{s' \in S} (\mathcal{P}(s, \alpha, s') \gtrdot 0 \rightarrow (q_{s'} \wedge r_s \lessdot r_{s'})) && \text{for all } s \in S \setminus T,\\
&\neg q_s \rightarrow p_s \doteq 0 && \text{for all } s \in S \setminus T,\\
&q_s \rightarrow \bigwedge_{\alpha \in \mathbb{A}\mathrm{ct}(s)} p_s \geqslant \sum_{s' \in S} \mathcal{P}(s, \alpha, s') \cdot p_{s'} && \text{for all } s \in S \setminus T.
\end{aligned}
$$

We refer to the corresponding formula as $\Phi_{\mathrm{wd}}^{\mathbb{P}, \lessdot, \forall}(\mathcal{M})$.

Here, for lower-bounded reachability, we change the zero-state computation and the inequalities on the probabilities.

Constraints 4.13 (Lower-bounded reachability, demonic nondeterminism)

$$
\begin{aligned}
&p_s \doteq 1 && \text{for all } s \in T,\\
&q_s && \text{for all } s \in T,\\
&q_s \leftrightarrow \bigwedge_{\alpha \in \mathbb{A}\mathrm{ct}(s)} \bigvee_{s' \in S} \left(\mathcal{P}(s,\alpha,s') \gtrdot 0 \rightarrow (q_{s'} \wedge r_s \lessdot r_{s'})\right) && \text{for all } s \in S \setminus T,\\
&\neg q_s \rightarrow p_s \doteq 0 && \text{for all } s \in S \setminus T,\\
&q_s \rightarrow \bigwedge_{\alpha \in \mathbb{A}\mathrm{ct}(s)} p_s \leqslant \sum_{s' \in S} \mathcal{P}(s,\alpha,s') \cdot p_{s'} && \text{for all } s \in S \setminus T.
\end{aligned}
$$

We refer to the corresponding formula as $\Phi_{\mathrm{wd}}^{\mathbb{P},\trianglerighteq,\forall}(\mathcal{M})$.

The accompanying encoding is then similar to the angelic case.

Theorem 4.20 Let $\mathcal{M}$ be a pMDP with target states T. Let $R \subseteq \mathsf{Val}_{\mathcal{M}}^{\mathrm{wd}}$ be a region, and let $\varphi = \mathbb{P}_{\bowtie\lambda}(\Diamond T)$. We distinguish two cases:

- Either $\bowtie \in \{\leq, <\}$. We define
$$\psi := \Phi_{\mathrm{wd}}^{\mathbb{P},\trianglelefteq,\exists}(\mathcal{M}) \wedge p_\iota \bowtie \lambda \wedge \psi_R,$$
- or $\bowtie \in \{\geq, >\}$, and we define
$$\psi := \Phi_{\mathrm{wd}}^{\mathbb{P},\trianglerighteq,\exists}(\mathcal{M}) \wedge p_\iota \bowtie \lambda \wedge \psi_R.$$

Then, for any $\mathsf{val} \in \mathsf{Val}$,

$$\mathsf{val} \in \mathsf{sat}(\psi) \quad \text{iff} \quad \mathcal{M}[\mathsf{val}_{\downarrow X}] \,\exists\!\models \varphi.$$

Example 4.33 Consider the pMDP in Figure 4.11. For conciseness, we simplified several constraints. We first consider the encoding for $\varphi := \mathbb{P}_{\leq 1/2}(\Diamond T)$.

$$
\begin{aligned}
&p_1 \leqslant 1/2 \wedge p_0 \doteq 0 \wedge \neg q_0 \wedge p_3 \doteq 1 \wedge q_3,\\
&q_1 \leftrightarrow \Big(\big(x \gtrdot 0 \wedge q_2 \wedge r_1 \lessdot r_2\big) \vee \big(1 - y \gtrdot 0 \wedge q_2 \wedge r_1 \lessdot r_2\big)\Big),\\
&q_2 \leftrightarrow \Big(\big(y \gtrdot 0\big) \vee \big(1 - y \gtrdot 0 \wedge q_1 \wedge q_2 \lessdot q_1\big)\Big),\\
&q_1 \rightarrow \big(p_1 \geqslant x \cdot p_2\big) \wedge \big(p_1 \geqslant (1-y) \cdot p_2\big)\Big) \wedge \neg q_1 \rightarrow p_1 \doteq 0,\\
&q_2 \rightarrow \big(p_2 \geqslant (1-y) \cdot p_1 + y\big) \wedge \neg q_1 \rightarrow p_2 \doteq 0.
\end{aligned}
$$

Below, we give the encoding for $\varphi := \mathbb{P}_{\geq 1/2}(\Diamond T)$. Compared to the encoding above, only the first constraint and one further connective in the second line changed.

$$
\begin{aligned}
&p_1 \geqslant 1/2 \wedge p_0 \doteq 0 \wedge \neg q_0 \wedge p_3 \doteq 1 \wedge q_3,\\
&q_1 \leftrightarrow \Big(\big(x \gtrdot 0 \wedge q_2 \wedge r_1 \lessdot r_2\big) \wedge \big(1 - y \gtrdot 0 \wedge q_2 \wedge r_1 \lessdot r_2\big)\Big),\\
&q_2 \leftrightarrow \Big(\big(y \gtrdot 0\big) \vee \big(1 - y \gtrdot 0 \wedge q_1 \wedge q_2 \lessdot q_1\big)\Big),\\
&q_1 \rightarrow \Big(\big(p_1 \leqslant x \cdot p_2\big) \wedge \big(p_1 \leqslant (1-y) \cdot p_2\big)\Big) \wedge \neg q_1 \rightarrow p_1 \doteq 0,\\
&q_2 \rightarrow \big(p_2 \leqslant (1-y) \cdot p_1 + y\big) \wedge \neg q_1 \rightarrow p_2 \doteq 0.
\end{aligned}
$$

4.3.4 Encoding via the solution function

The encodings presented above contain $O(|S| + |X|)$ many variables. As solving ETR is exponential in the number of variables, the large number of variables is a significant hurdle. In this section, we present encodings that prevent the dependency on the number of states, by incorporating the solution function.

We reconsider the encoding for pMCs under the assumption that we may precompute the zero-states. Reachability in an MC corresponds to a linear equation system $A \cdot \vec{p} = \vec{b}$ where $\vec{p}$ is the solution vector, and A is a matrix and b is a vector, cf. Section 3.3.2.1 on page 52. Following Proposition 4.11, for pMCs, $A \cdot \vec{p} = \vec{b}$ may be viewed as a linear equation system over the field $\mathbb{Q}(X)$ of rational functions with rational coefficients. That is, the entries of A are no longer rational numbers, but rational functions instead.

Example 4.34 We repeat the equation system from Example 4.26 describing the reachability in the pMC from Figure 4.9 on page 84.

$$
\begin{aligned}
&p_0 \doteq 0, && p_1 \doteq (1-x) \cdot p_0 + x \cdot p_2,\\
&p_3 \doteq 1, && p_2 \doteq y \cdot p_1 + (1-y) \cdot p_3.
\end{aligned}
$$

We bring this into standard form:

$$
\begin{aligned}
p_0 \qquad\qquad\qquad\qquad &\doteq 0\\
(1-x) \cdot p_0 \quad -p_1 + x \cdot p_2 \qquad\qquad &\doteq 0\\
y \cdot p_1 \quad -p_2 + (1-y) \cdot p_3 &\doteq 0\\
p_3 &\doteq 1.
\end{aligned}
$$

By basic linear algebra, we obtain the following key theorem:

Theorem 4.21 For a pMC $\mathcal{D}$ with targets T, there exist $f \in \mathbb{Q}(X)$ such that

$$^{\mathbb{P}}\mathsf{sol}^{\mathcal{D}}_{s\to T}[\mathsf{val}] = f[\mathsf{val}] \quad \text{for all } \mathsf{val} \in \mathsf{Val}^{\mathsf{gp}}_{\mathcal{D}}.$$

The rational function f is exactly the entry q_ι of the unique solution q for the system $A \cdot \vec{p} = \vec{b}$. Thus, solving linear equation systems (symbolically) is sufficient to find these solution functions. Details are given in Section 8.1. We observe that f is the restriction of $^{\mathbb{P}}\mathsf{sol}^{\mathcal{D}}_{s\to T}$ to $\mathsf{Val}^{\mathsf{gp}}_{\mathcal{D}}$. We denote this restriction with $_{\mathsf{gp}}^{\mathbb{P}}\mathsf{sol}^{\mathcal{D}}_{s\to T}$. Likewise, we denote the restrictions of, e.g., $^{\mathbb{P}}\mathsf{minsol}^{\mathcal{D}}_{T}$ with $_{\mathsf{gp}}^{\mathbb{P}}\mathsf{minsol}^{\mathcal{D}}_{T}$.

Example 4.35 The unique solution for the system in Example 4.34 is:

$$[0 \quad \frac{x \cdot (1-y)}{1-xy} \quad \frac{1-y}{1-xy} \quad 1]$$

To see that this is a solution, observe that

$$\frac{1-y}{1-xy} = \frac{y \cdot x \cdot (1-y)}{1-xy} + 1 - y = \frac{y \cdot x \cdot (1-y) + (1-xy)(1-y)}{1-xy}.$$

Consequently, we have:

$$_{\mathsf{gp}}^{\mathbb{P}}\mathsf{sol}^{\mathcal{D}}_{T} = \frac{x \cdot (1-y)}{1-xy}.$$

Remark 4.3 Under the standard restrictions that the expected reward is finite, the statements in this section may be extended to expected rewards. For conciseness, we omit these adaptions.

We consider further properties of this rational function in Section 5.2 on page 121. Here, we finish the discussion about encoding reachability in ETR.

Theorem 4.22 Let $\mathcal{D}$ be a pMC with targets T and threshold λ, and let $f/g := {}_{\mathsf{gp}}^{\mathbb{P}}\mathsf{sol}^{\mathcal{D}}_{T}$. Then:

$$\mathsf{val} \in \mathsf{sat}\Big(\big((g \gtrdot 0 \wedge f \bowtie \lambda \cdot g) \vee (g \lessdot 0 \wedge \lambda \cdot g \bowtie f)\big) \wedge \psi_R\Big) \;\text{ iff }\; \mathsf{val} \in \mathsf{acc}^{\mathcal{D}}_{\varphi} \cap R.$$

4

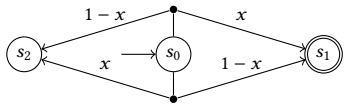


Figure 4.12: pMDP showing that ${}^{\mathbb{P}}_{\mathrm{gp}}\mathsf{minsol}^{\mathcal{D}}_{T}$ is not rational.

Example 4.36 We continue with the solution function from Example 4.35. Let $\varphi \coloneqq \mathbb{P}_{\geq 1/2}(\Diamond T)$. In particular, we obtain

$$\mathsf{acc}^{\mathcal{D}}_{\varphi} \cap \mathsf{Val}^{\mathrm{gp}}_{\mathcal{D}} = \mathsf{sat}\big(0 \lessdot x \lessdot 1 \wedge 0 \lessdot y \lessdot 1 \wedge x \cdot (1-y) \geqslant 1/2 \cdot (1-xy)\big).$$

We may lift the idea of using a solution function to pMDPs. However, the functions ${}^{\mathbb{P}}_{\mathrm{gp}}\mathsf{minsol}^{\mathcal{D}}_{T}$ and ${}^{\mathbb{P}}_{\mathrm{gp}}\mathsf{maxsol}^{\mathcal{D}}_{T}$ are in general not rational functions.

Example 4.37 Consider the pMDP in Figure 4.12. We have

$${}^{\mathbb{P}}_{\mathrm{gp}}\mathsf{minsol}^{\mathcal{D}}_{T}(x) \coloneqq \begin{cases} x & \text{if } x \leq 1/2, \\ 1-x & \text{otherwise.} \end{cases}$$

The function is not differentiable at $x = 1/2$, and ${}^{\mathbb{P}}_{\mathrm{gp}}\mathsf{minsol}^{\mathcal{D}}_{T}(1/2) = 1/2$. Therefore, this cannot be a rational function.

By definition, these functions are the maximum over (at most exponentially many) rational functions, leading to exponentially large conjunctions. We consider whether a smaller set of rational functions suffices further in Section 5.3 on page 135. Here, we only briefly show the ETR encodings:

Theorem 4.23 Let $\mathcal{M}$ be a pMDP with targets T, threshold λ, and let $R \subseteq \mathsf{Val}^{\mathrm{gp}}_{\mathcal{M}}$. Furthermore, let $f_\sigma/g_\sigma \coloneqq {}^{\mathbb{P}}\mathsf{sol}^{\mathcal{M}[\sigma]}_{T}$ with $g_\sigma > 0$ for all $\sigma \in \Sigma^{\mathcal{M}}$. Then:

$$\mathsf{acc}^{\mathcal{D}}_{\forall,\varphi} \cap R = \mathsf{sat}\Big(\psi_R \wedge \bigwedge_{\sigma \in \Sigma^{\mathcal{M}}} f_\sigma \bowtie \lambda \cdot g_\sigma\Big),$$

and

$$\mathsf{acc}^{\mathcal{D}}_{\exists,\varphi} \cap R = \mathsf{sat}\Big(\psi_R \wedge \bigvee_{\sigma \in \Sigma^{\mathcal{M}}} f_\sigma \bowtie \lambda \cdot g_\sigma\Big).$$

Naturally, this theorem only has practical relevance when the number of strategies (and thus the number of rational functions) is small.

Example 4.38 Consider the pMDP in Figure 4.12 and $\varphi = \mathbb{P}_{\leq\lambda}(\Diamond\{s_1\})$. We have

$$\mathsf{acc}^{\mathcal{M}}_{\forall,\varphi} \cap \mathsf{Val}^{\mathrm{gp}}_{\mathcal{M}} = \mathsf{sat}\Big(0 \lessdot x \lessdot 1 \wedge x \leqslant \lambda \wedge 1 - x \leqslant \lambda\Big),$$

and

$$\mathsf{acc}^{\mathcal{M}}_{\exists,\varphi} \cap \mathsf{Val}^{\mathrm{gp}}_{\mathcal{M}} = \mathsf{sat}\Big(0 \lessdot x \lessdot 1 \wedge \big(x \leqslant \lambda \vee 1 - x \leqslant \lambda\big)\Big).$$

We introduced parametric MDPs and MCs, known from the literature, and formalised various aspects and notions of these parametric models. Furthermore, we presented a list of famous parameter synthesis problems: The most important problems are feasibility (*Is there a valuation such that a property is accepted?*) and its dual, region verification. Finally, we studied an ETR encoding that generalises the standard linear (in)equations for the parameter-free models to feasibility in parametric models. We showed simpler encodings for various special cases of feasibility and concluded with encodings that use a closed form of the solution function.

5 Aspects of Parameter Synthesis

Fundamental traits and computational complexity

Summary | This section considers the feasibility problem in detail. We introduce prominent subclasses of pMDPs, increasing our understanding and allowing for concise theorems. We then dive into the shape and size of the solution function. From there, we learn that the class of polynomials corresponding to some solution function of an acyclic pMC is very general. Finally, we briefly consider the set of optimal strategies for pMDPs. These parts together are the foundation for the complexity results about the feasibility problem on pMCs and pMDPs, with the main result that many variants are ETR-complete. We finish the chapter with a comparison to robust strategies; they correspond to a feasibility problem with the quantification over parameters and strategies swapped.

Origins | The presentation is mostly based on [19], with some additions from [20]. The chapter contains various lemmas and propositions that never made it into the tight page limits of publications, but that ought to help understanding pMCs and pMDPs.

Background | We assume the reader is familiar with the feasibility problem for pMDPs and model checking techniques, as presented in the previous chapters. However, we provide back-references in case the reader missed or forgot some of their contents.

5.1 Classes of Parametric Models

Very often, we are interested in sets of MDPs that may be modelled with pMDPs that use only (very) restricted transition probabilities (e.g., only affine polynomials) and sets of valuations that are more restricted (e.g., graph-preserving). In this section, we investigate such subclasses and relate them to each other. These subclasses form the basis for much of the results later in this chapter.

5.1.1 Subclasses of parametric models

In pMDPs, rational functions over parameters allow for a very flexible way of modelling transition probabilities. However, this flexibility comes at the cost of additional

technicalities and avoids some straightforward examples and definitions. Needless to say, these technicalities also prevent providing clean algorithms. Therefore, it is desirable to consider subclasses. At the same time, it is important to ensure that these subclasses are sufficiently flexible in modelling. We consider useful and widespread subclasses of pMDPs. The restrictions carry over to pMCs in a straightforward way.

5.1.1.1 Conceptually simpler models

Acyclic models | pMDPs are acyclic, if the graph of the pMDP is acyclic, i.e., if all instantiations are acyclic. This class leads to simplifications that become apparent later.

Technically concise models | We already introduced polynomial pMDPs, i.e. pMDPs with polynomial transition probabilities, in Definition 4.15 on page 78. Most pMDPs in the literature are modelled as polynomial pMDPs. However, e.g. embedded pMCs of parametric CTMCs describing expected times to reach a target, or embedded pMDPs of parametric Markov automata, are not polynomial, cf. Section 1.2 for literature pointers on parametric CTMCs.

A similar motivation leads us to avoid considering the (parametric) reward function. To that end, we often consider pMDPs with constant rewards.

·constant-reward

Definition 5.1 (Constant-reward pMDP) A pMDP is a *constant-reward* pMDP, if for all $s \in S$, $\mathsf{prew}(s) \in \mathbb{R}_{\geq 0}$.

To clarify that rewards are constant, we write rew instead of prew. The typical example for parametric rewards is, again, the computation of expected time on a parametric CTMC. In that context, the same parameters occur on both the probabilities and the rewards. Other examples include uncertainty over resources (time, energy, ...). Mostly, the parameters that then occur in rewards do not occur on the probabilities.

Proposition 5.1 For every pMDP $\mathcal{M}$ and for any (infinite horizon) property φ, there exists a constant-reward pMDP $\mathcal{M}'$ such that:

$$\mathsf{acc}_{\varphi}^{\mathcal{M}} = \mathsf{acc}_{\varphi}^{\mathcal{M}'}.$$

As we do not rely further on this observation, we only exemplify the construction.

Example 5.1 Consider the pMC in Figure 5.1a. The reward from state s_1 onward is described by

$$p_1 = f + 1 \cdot p_2$$

(a) pMC with $\mathsf{prew}(s_1) = f$ and $\mathsf{prew}(s_0) = \mathsf{prew}(s_2) = 0$.

(b) pMC with $\mathsf{prew}(s_3) = 1$ and $\mathsf{prew}(s_0) = \mathsf{prew}(s_1) = \mathsf{prew}(s_2) = 0$.

Figure 5.1: pMC and equivalent constant-reward pMC

and depends on a polynomial $f \in \mathbb{Q}[X]$. This pMC is transformed to the pMC in Figure 5.1b, where reward one is collected in state s_3. From state s_1, the expected reward is described by

$$p_1 \doteq \frac{f}{1+f} \cdot p_3 + \frac{1}{1+f} \cdot p_2, \quad \text{and} \quad p_3 \doteq 1 + p_1.$$

Substituting these two equations yields:

$$p_1 \doteq \frac{f}{1+f} \cdot (1 + p_1) + \frac{1}{1+f} \cdot p_2$$

and after simplification, we obtain

$$\frac{1}{1+f} \cdot p_1 \doteq \frac{f}{1+f} + \frac{1}{1+f} \cdot p_2$$

which (after multiplying both sides with $1 + f$ is equivalent to first equation for the pMC in Figure 5.1a.

After this transformation, a polynomial pMDP is no longer polynomial. Furthermore, models which only contain parameters in the rewards yield linear encodings for the expected reward and are therefore significantly easier to handle [19].

Models for tractable realisability | Polynomial pMDPs allow for a flexible modelling of transition probabilities. In particular, the polynomials may become arbitrarily complex. This complexity leads to problems. Whereas pMDPs are naturally used as generator of a set of MDPs, even checking *realisability*—i.e., whether $\langle \mathcal{M} \rangle = \emptyset$—becomes intricate: Indeed, the following theorem formalises this observation. realisability

Theorem 5.2 ([LMT07]) Given a polynomial pMDP with two states, deciding:

is the pMDP realisable? is ETR-complete.

5

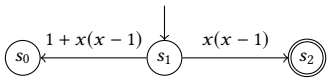


Figure 5.2: pMC with an unconnected set $\mathsf{Val}^{\mathrm{wd}}_{\mathcal{D}}$

Proof (sketch). Consider a pMC with two states and a single transition between them with probability $f \in \mathbb{Q}[X]$. The constraints for well-definedness collapse to $f \doteq 1$, or equivalently $f - 1 \doteq 0$. For multivariate polynomials of degree at least four, answering this question is ETR-complete, cf. Proposition 2.3. ■

This complexity drops when all transition probabilities become linear.

·affine

Definition 5.2 A polynomial pMDP is *affine*, if

for all $s, s' \in S, \alpha \in \mathbb{A}\mathrm{ct}(s) : \mathcal{P}(s, \alpha, s')$ is affine and $\mathsf{prew}(s)$ is affine.

The rock-paper-scissor pMDP from Figure 4.2a is affine, as is the Knuth-Yao die pMC from Figure 4.1b.

Proposition 5.3 Given an affine pMDP, deciding:

is the pMDP realisable? is in P.

Proof (sketch). The encoding in Constraints 4.1 is a conjunction of affine constraints. ■

Affine polynomials are still flexible. The typical instances of non-linear behaviour occur from the parallel execution of affine pMCs, as illustrated by the following example (inspired by [AVBK+17]).

Example 5.2 We flip two coins in parallel and go to the target state iff both coins show heads. Assume that heads is flipped with probabilities x and y, then the transition from initial state to the target has the (non-affine) probability $x \cdot y$.

Non-affine probabilities often occur as product of affine polynomials, e.g., of $\{x, 1 - x \mid x \in X\}$. Such products may be modelled by a chain of transitions, where each transition is affine. Affine pMDPs have another important property. We first observe that the well-defined region of a polynomial pMDP is not necessarily connected.

Example 5.3 Consider the pMC in Figure 5.2. It holds: $\mathsf{Val}^{\mathrm{wd}}_{\mathcal{D}} = \{0, 1\}$.

However, for affine pMDPs, we may exploit that the well-defined and graph-preserving regions are connected.

Proposition 5.4 For any affine pMDP, its well-defined (graph-preserving) region is connected.

Proof. The well-defined (graph-preserving) region is a convex polytope. ■

So far, we have restricted the form of the transition probabilities:

$$\text{affine pMDPs} \subseteq \text{polynomial pMDPs} \subseteq \text{pMDPs},$$

and of the reward probabilities (where we only distinguished constant or not). Another restriction to decrease the complexity of deciding realisability is action-univariate pMDPs. In such pMDPs, for any state-action pair, a single parameter occurs:

Definition 5.3 (Action-univariate) A pMDP is *action-univariate*, if ·action-univariate

$$\text{for all } s \in S, \alpha \in \mathbb{A}\mathrm{ct}(s) : |\mathsf{occurs}(s, \alpha)| = 1$$

Example 5.4 The Knuth-Yao die pMC in Figure 4.1b is action-univariate. In every state, the outgoing transitions are labelled with $\{x, 1-x\}$ or $\{y, 1-y\}$. Only one parameter occurs in this set of polynomials. The rock-paper-scissors in Figure 4.2a is not action-univariate, as for a state-action pair, there are up to three parameters on the set of polynomials that occur on the outgoing transitions.

A special case are univariate pMDPs.

Definition 5.4 (Univariate) A pMDP is *univariate* if $|X| = 1$. ·univariate

A pMDP that is not univariate is called *multivariate*. ·multivariate

Proposition 5.5 Given an action-univariate pMDP deciding:

is the pMDP realisable? is in P.

Proof (sketch). The essential insight is formulating the following characterisation:

$$\mathsf{Val}^{\mathrm{wd}}_{\mathcal{M}} = \bigtimes_{x \in X} V_x \quad \text{with } V_x \subseteq \mathbb{R}.$$

Then, $\mathsf{Val}^{\mathrm{wd}}_{\mathcal{M}} = \emptyset$ iff $V_x = \emptyset$ for some $x \in X$. The encoding for every V_x is obtained by collecting the constraints from Constraints 4.1 only for states with parameter x on its outgoing transitions. This encoding may be nonlinear, but contains only a single variable x. Therefore, checking whether it has a feasible solution may be done in polynomial time (Proposition 2.4). If we do this for every variable separately, we obtain a polynomial time algorithm. ■

The statement may be generalised to multiple parameters. Indeed, it only requires some a-priori fixed bound on the number of parameters in any outgoing transitions.

Simplex and simple pMDPs | Two—on first sight—restrictive subclasses combine these aspects and have particular practical value. Let us therefore consider that often, parameters encode unknown distributions.

Example 5.5 Recall the pMDP for rock-paper-scissors from Example 4.2. We modelled an arbitrary distribution over three choices by three parameters x_R, x_P, x_S and another distribution with the three parameters x'_R, x'_P, x'_S. No complex expressions are required.

To support modelling unknown distributions, the following model thus suffices.

Definition 5.5 (Expanded simplex pMDPs) Let $\mathcal{M}$ be a pMDP with parameters X. Let $X = X_1 \uplus X_2 \uplus \ldots \uplus X_m$ be a partitioning of the parameters. A constant-reward pMDP is *(expanded) simplex*, if

grouped parameters

- it has *grouped parameters*, i.e., if for all $s \in S$, $\alpha \in \mathbb{A}\mathrm{ct}(s)$:

$$\mathsf{occurs}(s, \alpha) = X_i \text{ for some } 1 \leq i \leq m,$$

- all transition probabilities are parameters or constants, that is:

$$\mathcal{P}(s, \alpha, s') \in X \cup \mathbb{R}_{\geq 0} \text{ for all } s, s' \in S \text{ and } \alpha \in \mathbb{A}\mathrm{ct}(s).$$

The name is based on the geometrical interpretation of the well-defined valuations: It is the Cartesian product of the simplices for the parameter groups.

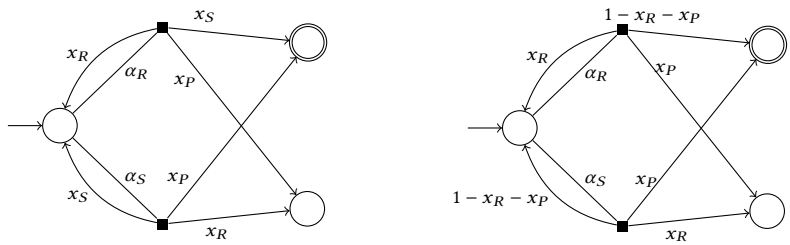
Definition 5.6 ([BV04]) Let $\boldsymbol{x}$ be a set of variables. The set $\mathsf{Simplex}(\boldsymbol{x})$ is the simplex for $\boldsymbol{x}$ such that:

$$\mathsf{Simplex}(\boldsymbol{x}) = \mathsf{sat}\Big(\sum_{x \in \boldsymbol{x}} x \doteq 1 \wedge \bigwedge_{x \in \boldsymbol{x}} x \geqslant 0 \Big).$$

Example 5.6 Recall the pMDP $\mathcal{M}$ for rock-paper-scissors from Example 4.2. The pMDP is an expanded simplex pMDP with two parameter groups, $X_1 := \{x_R, x_P, x_S\}$ and $X_2 := \{x'_R, x'_P, x'_S\}$. Consequently:

$$\mathsf{val} \in \mathsf{Val}^{\mathrm{wd}}_{\mathcal{M}} \quad \text{iff} \quad \mathsf{val}_{\downarrow X_1} \in \mathsf{Simplex}(X_1) \text{ and } \mathsf{val}_{\downarrow X_2} \in \mathsf{Simplex}(X_2).$$

Observe that in a simplex over k variables, there are $k - 1$ degrees of freedom. Put differently, once the first $k - 1$ values in the valuation are fixed, the last value is uniquely determined. This observation allows us to reduce parameters. In particular, for Bernoulli distributions (coin flips), the number of parameters is halved. This observation leads to the following adaption of simplex pMDPs.



(a) expanded simplex pMDP

(b) Compact simplex pMDP

Figure 5.3: Expanded and compact simplex pMDPs

Definition 5.7 (Compact simplex pMDPs) Let $\mathcal{M}$ be a pMDP with parameters X. Let $X = X_1 \uplus X_2 \uplus \ldots \uplus X_m$ a partitioning of the parameters. A constant-reward pMDP is simplex, if

- it has *grouped parameters* as in Definition 5.5,
- transitions syntactically sum to 1, i.e., for all $s \in S$ and $\alpha \in \mathbb{A}\mathrm{ct}(s)$:

$$\sum_{s' \in S} \mathcal{P}(s, \alpha, s') = 1, \text{ and}$$

- for all $s, s' \in S$ and $\alpha \in \mathbb{A}\mathrm{ct}(s)$:

$$\mathcal{P}(s, \alpha, s') \in \{x \mid x \in X\} \cup \{1 - \sum_{x \in X} x\} \cup \mathbb{Q}_{\geq 0}.$$

As the models are straightforwardly related, we often omit the prefix expanded or compact. In particular, expanded simplex pMDPs are easier to describe and transform.

Example 5.7 The Knuth-Yao die pMC in Figure 4.1b is a compact simplex pMC. We illustrate the correspondence between expanded and compact simplex pMDPs in Figure 5.3, where we replaced x_S by $1 - x_R - x_P$.

More formally:

Corollary 5.6 Let $\mathcal{M}$ be an expanded simplex pMDP. There exists a compact simplex pMDPs $\mathcal{M}'$ with the same topology such that $\langle \mathcal{M} \rangle = \langle \mathcal{M}' \rangle$. Let $\mathcal{M}$ be a compact simplex pMDP. There exists an expanded simplex pMDPs $\mathcal{M}'$ with the same topology such that $\langle \mathcal{M} \rangle = \langle \mathcal{M}' \rangle$.

We refrain from a formal construction here, but for completeness one may use the Definitions 5.10 to 5.12 to extract formal constructions of the translations.

In our final restriction, we combine the restrictions from above and consider a model tailored to (parametric) Bernoulli experiments.

·simple

Definition 5.8 (Simple pMDPs) A simplex action-univariate pMDP is *simple*.

Example 5.8 The Knuth-Yao die pMC in Figure 4.1b is simple. The pMDPs in Figure 5.3 are *not* simple.

Throughout this thesis, we often use simple pMDPs for their nice properties. We may use the following two characterisations of simple pMDPs.

Corollary 5.7 A pMDP is simple if the following two properties are satisfied:

- $\mathcal{P}(s, \alpha, s') \in \{x, 1 - x \mid x \in X\} \cup \mathbb{Q}_{\geq 0}$ for all $s, s' \in S$ and $\alpha \in Act$; and
- $\sum_{s' \in S} \mathcal{P}(s, \alpha, s') = 1$ for all $s \in S$ and $\alpha \in \mathbb{A}\mathrm{ct}(s)$.

Alternatively, a pMDP is simple if it is a simplex pMDP and all parametric actions have out-degree two, i.e., for all $s \in S$, $\alpha \in \mathbb{A}\mathrm{ct}(s)$: either

- $\mathcal{P}(s, \alpha, s') \in \mathbb{Q}$ for all $s' \in S$, or
- $|\{s' \in S \mid \mathcal{P}(s, \alpha, s') \neq 0\}| = 2$.

The well-defined and graph-preserving regions are represented by $[0, 1]^X$ and $(0, 1)^X$, respectively. Among others, this makes simple pMDPs comfortable to handle. In the remainder of this chapter, we focus on simple and simplex pMDPs.

5.1.1.2 Relating simple models

As simple(x) pMDPs are both widespread in the literature and heavily used in this thesis, we clarify their relation a bit more. To that end, we propose a selection of transformations. These transformations allow us to pick any of the classes as we see fit for our purposes.

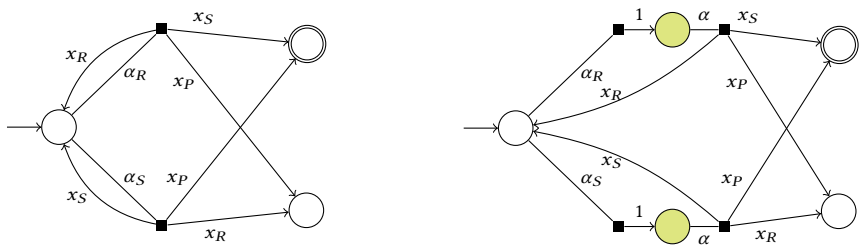
Remark 5.1 In this section, we introduce some transformations on pMDPs. We include formal constructions as these transformations are implicitly exploited in subsequent claims. We recommend the reader focusses on the provided examples.

Alternating pMDPs | Any pMDP can be transformed into an alternating pMDP. This transformation simplifies the formal construction of various preprocessing steps.

·alternating

Definition 5.9 A pMDP is *alternating*, if for all $s, s' \in S$, $\alpha \in Act$:

$$|\mathbb{A}\mathrm{ct}(s)| > 1 \quad \text{implies} \quad \mathcal{P}(s, \alpha, s') = 1,$$



(a) Non-alternating pMDP

(b) Alternating pMDP

Figure 5.4: From pMDP to alternating pMDP

and for all $s \in S$, $\alpha \in Act$:

$$|\mathbb{A}\mathrm{ct}(s)| > 1 \quad \text{implies} \quad \mathsf{succ}(s) = |\mathbb{A}\mathrm{ct}(s)|.$$

The first criterium ensures that between two states, we do not have both nondeterministic and probabilistic branching, and is inspired by [ST05]. The second criterium is a technicality to ease a later construction in Definition 5.13, cf. Example 5.11.

Example 5.9 Consider the pMDP from Figure 5.4a. The pMDP is not alternating, as in the initial state there are two actions and their distributions are not Dirac. We may transform the pMDP into Figure 5.4b: This pMDP is alternating, by introducing auxiliary states for every (non-trivial) action choice.

An example of a pMDP that violates (only) the second criterium is given in Figure 5.8b. The second criterium may always be ensured by the duplication of states. To ensure both criteria we introduce the following translation.

Definition 5.10 Let $\mathcal{M} = (S, \iota, Act, X, \mathcal{P}, \mathsf{prew})$ be a pMDP. We define ·alt

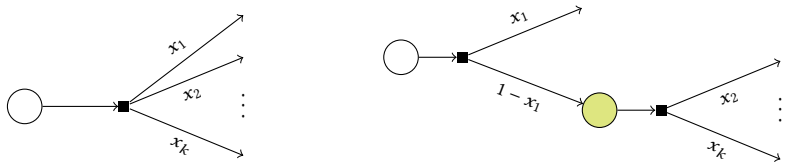
$$\mathsf{alt}(\mathcal{M}) := (S \times (Act \cup \{\bot\}), \langle \iota, \bot \rangle, Act \cup \{\beta\}, X, \mathcal{P}', \mathsf{prew}')$$

with $\mathcal{P}'$ defined by

$$\mathcal{P}'\big(\langle s, \alpha \rangle, \beta, \langle s', \alpha' \rangle\big) := \begin{cases} 1 & \text{if } \alpha = \bot, s = s', \beta = \alpha' \in \mathbb{A}\mathrm{ct}(s), \\ \mathcal{P}(s, \alpha, s') & \text{if } \alpha \neq \bot = \beta = \alpha', \\ 0 & \text{otherwise,} \end{cases}$$

and prew' such that $\mathsf{prew}(\langle s, \alpha \rangle) := \mathsf{prew}(s)$.

Obviously, $\mathsf{alt}(\mathcal{M})$ is an alternating pMDP. The transformation leads directly to the following statement.



(a) (Fragment) of a simplex pMDP

(b) (Fragment) of an equivalent simple pMDP

Figure 5.5: From simplex to simple pMDP (scheme)

Proposition 5.8 Let $\mathcal{M}$ be a pMDP and φ an (infinite horizon) property:

$$\mathsf{acc}_{\varphi}^{\mathcal{M}} = \mathsf{acc}_{\varphi}^{\mathsf{alt}(\mathcal{M})}.$$

From simplex to simple pMDPs | We may also make any simplex pMDP simple by introducing auxiliary states, see Figure 5.5.

Definition 5.11 Let $\mathcal{M} = (S, \iota, Act, X, \mathcal{P}, \mathsf{rew})$ be an alternating expanded simplex pMDP with parameter groups $X_1, \ldots, X_m$. Let $n \coloneqq \max_i |X_i|$. W.l.o.g., let $\beta \in Act$ be the unique action (after renaming) s.t. $\mathcal{P}(s, \beta, s') \neq 1$ for all $s, s' \in S$. The pMDP
·simple $\mathsf{simple}(\mathcal{M}) \coloneqq (S', \iota', Act, X, \mathcal{P}', \mathsf{rew}')$ with

$$S' = S \times \{1, \ldots, n-1\} \text{ with } \iota' = \langle \iota, 1 \rangle,$$

$\mathcal{P}'$ is defined by

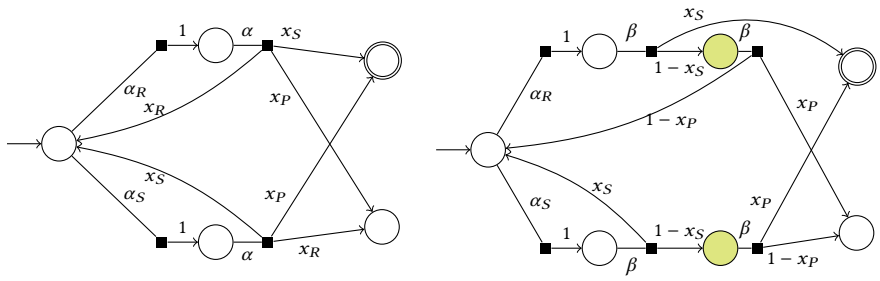
$$\mathcal{P}'(\langle s, i \rangle, \beta, \langle s', j \rangle) \coloneqq$$
$$\begin{cases} \mathsf{occurs}(s, \beta)_i & \text{if } j = 1, \mathcal{P}(s, \beta, s') = \mathsf{occurs}(s, \beta)_i, \\ 1 - \mathsf{occurs}(s, \beta)_i & \text{if } s = s', j = i + 1 < |\mathsf{occurs}(s, \beta)|, \\ 1 - \mathsf{occurs}(s, \beta)_i & \text{if } i + 1 = |\mathsf{occurs}(s, \beta)|, j = 1, \mathcal{P}(s, \beta, s') = \mathsf{occurs}(s, \beta)_i, \end{cases}$$

using that occurring parameters may be ordered, and

$$\mathcal{P}'(\langle s, i \rangle, \alpha \neq \beta, \langle s', j \rangle) \coloneqq \begin{cases} \mathcal{P}(s, \alpha, s') & \text{if } i = j = 1, \\ 0 & \text{otherwise,} \end{cases}$$

and rew defined by:

$$\mathsf{rew}'(\langle s, i \rangle) = \begin{cases} \mathsf{rew}(s) & \text{if } i = 1, \\ 0 & \text{otherwise.} \end{cases}$$



(a) Alternating simplex pMDP

(b) Alternating simple pMDP

Figure 5.6: From simplex to simple pMDP (example)

Observe that this construction reduces the number of occurring parameters.

Example 5.10 In Figure 5.6 we give an example where we first split first using x_S and then using x_P. It is important that we split on the same parameters at all states with the same parameter group.

Proposition 5.9 Let $\mathcal{M}$ be a simplex pMDP. Then:

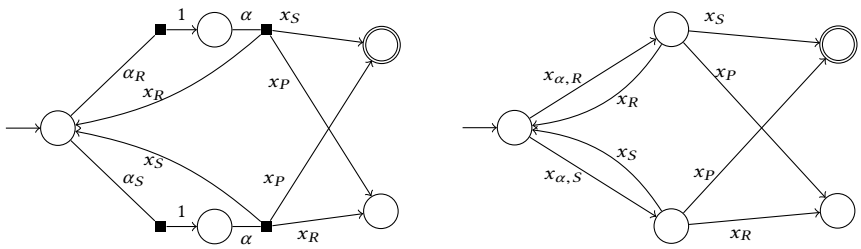
$$\langle \mathcal{M} \rangle \simeq \langle \mathsf{simple}(\mathcal{M}) \rangle.$$

The additional steps invalidate step-bounded properties, but this may be accommodated for by adding spurious states. Moreover, it may be practically better to use a binary tree as that gives an additional number of states logarithmic in the number of actions (but its formal encoding is slightly less concise).
For completeness, we define the translation from compact to expanded simplex pMDP.

Definition 5.12 Let $\mathcal{M} = (S, \iota, Act, X, \mathcal{P}, \mathsf{rew})$ be a compact simplex pMDP. We define the *corresponding expanded simplex* pMDP $\mathcal{M}' := (S, \iota, Act, X \cup X', \mathcal{P}', \mathsf{rew})$ with $X' := \{x' \mid x \in X\}$ and $\mathcal{P}$ defined by: ·expanded simplex

$$\mathcal{P}'(s, \alpha, s') := \begin{cases} x & \text{if } \mathcal{P}(s, \alpha, s') = x \text{ for some } x \in X, \\ x' & \text{if } \mathcal{P}(s, \alpha, s') = 1 - x \text{ for some } x \in X, \\ 0 & \text{otherwise.} \end{cases}$$

Replacing parameters with nondeterminism | For a (parametric) MDP $\mathcal{M}$, we may replace all nondeterminism by parameters and obtain a pMC, inspired by [HHZ11b] (they take Boolean variables).



(a) Alternating simplex pMDP

(b) simple pMC with strategy parameters

Figure 5.7: Replacing nondeterminism with parameters in an alternating pMDP

The idea is similar to our modelling of rock-paper-scissors. Intuitively, a randomised strategy takes at a particular state a particular action with some probability. This probability will be encoded by fresh variables. Observe that in rock-paper-scissors, we additionally have dependencies between states, which are not present in (p)MDPs: nondeterminism is resolved locally in these models. Below, we formalise the idea. The following definition is restricted to alternating pMDPs for conciseness.

·ntp

Definition 5.13 Let $\mathcal{M} = (S, \iota, Act, X, \mathcal{P}, \mathsf{rew})$ be an alternating pMDP. We define the pMC $\mathsf{ntp}(\mathcal{M}) = (S, \iota, X', \mathcal{P}', \mathsf{rew})$ with

$$X' := X \cup \{x_{s,\alpha} \mid s \in S, \alpha \in \mathbb{A}\mathrm{ct}(s)\},$$

and $\mathcal{P}$ is defined by:

$$\mathcal{P}'(s, s') := \begin{cases} \mathcal{P}(s, \alpha, s') & \text{if } \mathbb{A}\mathrm{ct}(s) = \{\alpha\}, \\ \sum_{\alpha \in \mathbb{A}\mathrm{ct}(s)} x_{s,\alpha} \cdot \mathcal{P}(s, \alpha, s') & \text{otherwise.} \end{cases}$$

The case distinction for $\mathcal{P}$ ensures that Corollary 5.11 below holds.

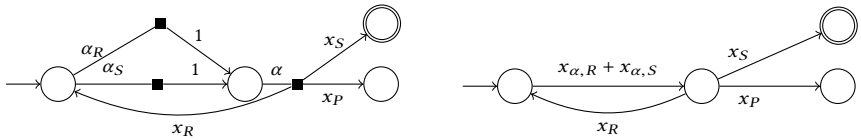
Proposition 5.10 For pMDP M, it holds:

$$\{M[\sigma] \mid M \in \langle \mathcal{M} \rangle, \sigma \in \mathfrak{S}_m^{\mathcal{M}}\} = \langle \mathsf{ntp}(\mathcal{M}) \rangle.$$

Observe that, by Proposition 3.7, randomised strategies are always dominated by deterministic ones. This dominance translates to the valuations in the pMC. In particular, if we start with a parameter-free MDP, it thus suffices to consider only pure valuations.

To ensure that the resulting pMC is simplex, we may make a simplex pMDP alternating.

Corollary 5.11 For a simplex $\mathcal{M}$, the pMC $\mathsf{ntp}(\mathsf{alt}(\mathcal{M}))$ is simplex.



(a) simplex pMDP

(b) pMC with strategy parameters

Figure 5.8: Replacing nondeterminism with parameters in an non-alternating pMDP

Example 5.11 We apply the transformation on the alternating pMDP Figure 5.7a. The result is the simplex pMC in Figure 5.7b. We also apply the construction to the pMDP in Figure 5.8a. This pMDP is not alternating as it violates the second criterium of Definition 5.9. Therefore, the pMC in Figure 5.8b is not simplex.

5.1.1.3 Relation to interval MCs

From Proposition 5.10, we conclude the following:

Proposition 5.12 Let M be an MDP with target states T and a property φ. Let $\mathcal{D} := \mathsf{ntp}(M)$. It holds:

$$M \not\models \varphi \quad \text{iff} \quad D \models \varphi \text{ for all } D \in \langle \mathcal{D} \rangle.$$

The proposition motivates us to consider pMCs which are the result of the $\mathsf{ntp}(M)$ operation, as they may be analysed efficiently using MDP model checking—which we do in Chapter 12. Observe that the operation $\mathsf{ntp}(M)$ generates fresh variables for all nondeterminism: each nondeterministic choice is a distribution that may be modelled by a parameter group of a simplex pMC.

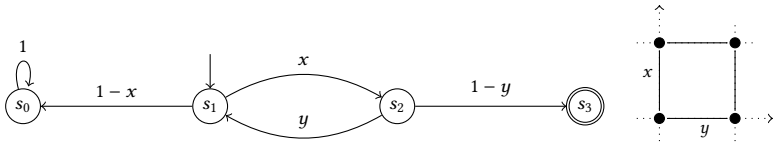
Definition 5.14 A simplex pMDP is *dependency-free*, if for states s, s' and actions α, α' it holds that ·dependency-free

$$\langle s, \alpha \rangle \neq \langle s', \alpha' \rangle \quad \text{implies} \quad \mathsf{occurs}(s, \alpha) \cap \mathsf{occurs}(s', \alpha') = \emptyset.$$

Without parameter dependencies, finding optimal valuations in a pMC is simpler. Locally optimising a parameter value has no negative effects on other states, much like picking an optimal action in some MDP state has no negative effect on reachability from another state.

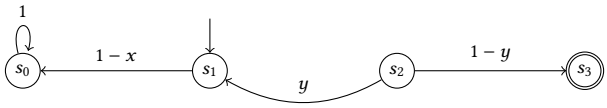
Given a dependency-free simplex pMCs together with a rectangular (not necessarily well-defined) region, we may replace the parameter ranges with intervals. We obtain a model that is called an *interval MC* [JL91]. Interval MCs thus represent dependency-free pMCs together with a particular region. For any $x \in \mathsf{occurs}(s)$, we may pick a value interval MC

5



(a) A small pMC

(b) Graph-consistent regions (schematic)



(c) Small pMC after removing vanishing transition

Figure 5.9: pMCs without vanishing transitions

in $I(x)$ that maximises the probability to reach T from state s. There is no (negative) effect for the reachability probability at the other states as $x = x_i^s$ only occurs at s. Optimal valuations can thus be determined *locally* (at the states). We focus on solving pMCs and pMDPs which are not dependency-free, and do not further discuss interval MC as they would widen the scope of this thesis considerably.

5.1.2 Subclasses of valuations

Before, we have restricted the structure of the pMDP. Here, we briefly consider subsets of the well-defined valuations. We have already seen graph-preserving valuations in Definition 4.13. Considering graph-preserving valuations is often sufficient from a practical point of view and comes with significant advances in terms of analysis.

Decomposing well-defined regions | When considering regions that are not a subset of the graph-preserving region, one approach is to partition the region. Any well-defined region may be partitioned into *graph-consistent regions*, see Definition 4.14.

-graph-consistent

Proposition 5.13 Let $\mathcal{M}$ be a pMDP and R a graph-consistent region. There exists a pMDP $\mathcal{M}'$ such that

$$R = \mathsf{Val}^{\mathrm{gp}}_{\mathcal{M}'} \quad \text{and} \quad \mathcal{M}[\mathsf{val}] = \mathcal{M}'[\mathsf{val}] \text{ for all } \mathsf{val} \in R.$$

In particular, these pMDPs are obtained by removing the vanishing transitions.

Example 5.12 Reconsider the pMC $\mathcal{D}$ from Figure 5.9a and let $R \coloneqq \mathsf{Val}^{\mathrm{wd}}_{\mathcal{D}}$. Region R may be partitioned into 9 regions, see Figure 5.9b where each dot, line

segment, and the inner region are subregions of R. All subregions are graph-preserving on some sub-pMC of $\mathcal{D}$. Consider, e.g., the line-region $R' = \{\mathsf{val} \in R \mid \mathsf{val}(x) = 0 \wedge 0 < \mathsf{val}(y) < 1\}$. The pMC $\mathcal{D}$ is not graph-preserving on R', as the transition $s_1 \xrightarrow{x} s_2$ vanishes when $x[\mathsf{val}] = 0$. However, R' is graph-preserving on the sub-pMC $\mathcal{D}'$ in Figure 5.9c, which is obtained from $\mathcal{D}$ by removing the transitions on the line-region.

We construct the graph-consistent sets and immediately show that they are regions by providing an encoding: For a given well-defined region R and pMDP $\mathcal{M}$, let $\Gamma_R^{\mathcal{M}}$ describe the set of constraints:

$$\Gamma_R^{\mathcal{M}} := \{\mathcal{P}(s, \alpha, s') \doteq 0 \mid s, s' \in S \text{ and } \alpha \in \mathbb{A}\mathrm{ct}(s) \text{ and } \mathcal{P}(s, \alpha, s') \neq 0\}.$$

For any $\Gamma \subseteq \Gamma_R^{\mathcal{M}}$, the subregion $R_\Gamma \subseteq R$ is defined as:

$$\psi_{R_\Gamma} = \psi_R \wedge \bigwedge_{\gamma \in \Gamma} \gamma \wedge \bigwedge_{\gamma \in \Gamma_R^{\mathcal{M}} \setminus \Gamma} \neg\gamma.$$

It follows that Γ uniquely characterises which transition probabilities in $\mathcal{M}$ vanish. In fact, each valuation in R_Γ is graph-preserving (and thus well-defined) for the unique sub-pMDP $\mathcal{M}'$ of $\mathcal{M}$ obtained from $\mathcal{M}$ by removing all vanishing transitions in R_Γ. By construction, it holds that $\mathcal{M}[\mathsf{val}] = \mathcal{M}'[\mathsf{val}]$ for all instantiations $\mathsf{val} \in \mathsf{Val}_{\mathcal{M}'}^{\mathrm{gp}}$.

Corollary 5.14 For any pMDP $\mathcal{M}$ and region $R \subseteq \mathsf{Val}_{\mathcal{M}}^{\mathrm{wd}}$, there are finitely many graph-consistent regions $R_1, \ldots, R_m$ such that

$$R = \bigcup_{i=1}^{m} R_i.$$

Observe that some of these regions may be empty. For (expanded) simplex pMDPs, one can directly construct all sub-pMDPs with non-empty well-defined regions.

Proposition 5.15 Let $\mathcal{M}$ be an expanded simplex pMDP with parameter groups $X_1, \ldots, X_m$. The well-defined region may be partitioned into $\prod_i \left(2^{|X_i|} - 1\right)$ maximal graph-consistent regions. Consequently, the well-defined valuations may be partitioned into 3^k maximal graph-consistent regions.

ε-preserving valuations | Graph-preserving valuations still come with some drawbacks on the analysis side. First, $\langle \mathcal{M} \mid \mathrm{gp} \rangle$ and $\langle \mathcal{M} \rangle$ contain MDPs with arbitrarily small probabilities. The analysis of such MDPs is numerically challenging and at probabilities beyond 10^{-15} events are *extremely unlikely*[1]. Second, there is generally

[1]That is, roughly it is more likely that an asteroid kills you while reading this sentence.

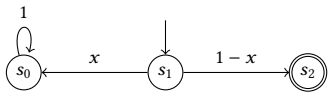


Figure 5.10: pMC without a maximum probability to target among $\mathsf{Val}^{\mathrm{gp}}_{\mathcal{D}}$

no MC generated by a pMC under graph-preserving instantiations that minimises (or maximises) the probability of reaching a target.

Example 5.13 Consider the pMC $\mathcal{D}$ in Figure 5.10. The valuation is graph-preserving if $0 < x < 1$. On that domain, the infimum of ${}^{\mathbb{P}}_{\mathrm{gp}}\mathsf{sol}^{\mathcal{D}}_{T}$ is zero, but there is no valuation $\mathsf{val} \in \mathsf{Val}^{\mathrm{gp}}_{\mathcal{D}}$ such that ${}^{\mathbb{P}}_{\mathrm{gp}}\mathsf{sol}^{\mathcal{D}}_{T}[\mathsf{val}]$ is zero.

These downsides are alleviated by the following restriction.

·ε-preserving

Definition 5.15 (Epsilon-preserving valuations) A valuation $\mathsf{val} \in \mathsf{Val}^{\mathrm{wd}}_{\mathcal{M}}$ is ε-*preserving* if for all $s, s' \in S$ and $\alpha \in \mathbb{A}\mathrm{ct}(s)$:

$$\mathcal{P}(s, \alpha, s') = 0 \quad \text{or} \quad \mathcal{P}(s, \alpha, s')[\mathsf{val}] \geq \varepsilon.$$

Observe that for $\epsilon = 0$, this coincides with the well-defined valuations, and for $\varepsilon > 0$, the valuation is necessarily graph-preserving.

Pure valuations | The valuations above yield uncountable sets (on simplex pMDPs). However, sometimes finite (large) sets represent all practically interesting cases. There are several possible restrictions to finite sets, the following is particularly intuitive.

·pure

Definition 5.16 (Pure valuations) Let $\mathcal{M}$ be a pMDP. A valuation $\mathsf{val} \in \mathsf{Val}^{\mathrm{wd}}_{\mathcal{M}}$ is *pure*, if $\mathsf{val}(x) \in \{0, 1\}$ for all $x \in X$. The set $\mathsf{Val}^{\mathrm{pure}}_{\mathcal{M}}$ contains all pure valuations.

Such a class of instantiations comes natural, e.g., when considering the choice of adding a particular hardware component: A component is simply present or not. When paired with simplex pMDPs, pure valuations allows to simply select sets of transitions by selecting parameter values. In Chapter 6, we discuss pure valuations using an equivalent formulation.

Example 5.14 The set $\mathsf{Val}^{\mathrm{pure}}_{\mathcal{M}}$ for the pMDP $\mathcal{M}$ from Figure 5.7a contains three valuations.

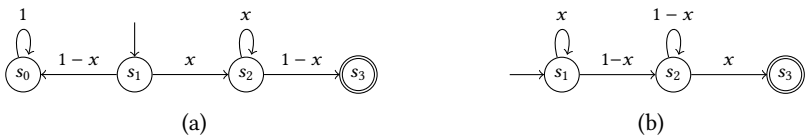


Figure 5.11: Two pMCs with non-continuous solution functions

5.2 Solution Functions and Accepting Sets

We consider the solution function—the mapping from parameter values to probabilities (or rewards)—in greater detail. In the first half of this section, we look at some attributes of these functions, in particular shape and size. For the second half, we show some results indicating that solution functions span a broad class of functions. Even stronger, considering accepting sets of pMCs, we construct pMCs for any accepting set described by a conjunction of the graph-preserving constraints and a single polynomial inequality over its parameters.

> **Remark 5.2** The remaining chapter provides deeper insights and constructions for (the proofs of the) complexity results. A reader new to the topic might want to skip to the complexity results directly (and then skip those proofs).

5.2.1 The solution function

In this subsection, we derive some properties of the solution function.

5.2.1.1 Shape of the solution function

We start with a simple observation:

> **Lemma 5.16** The solution function is continuous on $\mathsf{Val}^{\mathrm{gp}}_{\mathcal{D}}$.

Proof. The function ${}^{\mathbb{P}}_{\mathrm{gp}}\mathsf{sol}^{\mathcal{D}}_{T}$ is a rational function bounded by $[0, 1]$ on $\mathsf{Val}^{\mathrm{gp}}_{\mathcal{D}}$. ■

Consequently, the minimum and maximum over closed regions exist.

> **Corollary 5.17** Let $R \subseteq \mathsf{Val}^{\mathrm{gp}}_{\mathcal{D}}$ be a closed graph-preserving region. Then, there exists $\mathsf{val} \in R$ such that ${}^{\mathbb{P}}_{\mathrm{gp}}\mathsf{sol}^{\mathcal{D}}_{T}[\mathsf{val}] \geq {}^{\mathbb{P}}_{\mathrm{gp}}\mathsf{sol}^{\mathcal{D}}_{T}[\mathsf{val}']$ for all $\mathsf{val}' \in R$, and analogously, there exists $\mathsf{val} \in R$ such that ${}^{\mathbb{P}}_{\mathrm{gp}}\mathsf{sol}^{\mathcal{D}}_{T}[\mathsf{val}] \leq {}^{\mathbb{P}}_{\mathrm{gp}}\mathsf{sol}^{\mathcal{D}}_{T}[\mathsf{val}']$ for all $\mathsf{val}' \in R$.

The solution function is *not* continuous on $\mathsf{Val}^{\mathrm{wd}}_{\mathcal{D}}$. Let us consider two examples.

Example 5.15 Consider the two (cyclic) pMCs in Figure 5.11. Their solution functions are

$$^{\mathbb{P}}\mathsf{sol}_T^{\mathcal{D}_a}[\mathsf{val}] = \begin{cases} \mathsf{val}(x) & \text{if } \mathsf{val}(x) < 1, \\ 0 & \text{if } \mathsf{val}(x) = 1, \end{cases} \quad \text{and } ^{\mathbb{P}}\mathsf{sol}_T^{\mathcal{D}_b}[\mathsf{val}] = \begin{cases} 1 & \text{if } 0 < \mathsf{val}(x) < 1, \\ 0 & \text{if } \mathsf{val}(x) \in \{0, 1\}, \end{cases}$$

respectively. These functions are non-continuous and their zero-states depend on the parameter valuation. The jumps in the solution function can (only) occur on either one or both ends of the parameter range. In particular, there are jumps when the set of zero-states changes under different valuations.

We may generalise the observations from the example to a general setting:

The main intuition behind the argument presented herein is the following: For any equation system induced by a fixed set of zero-states, the solution is a rational function and thus continuous. However, if this equation system changes structurally, there may be discontinuities. These changes—and thus the discontinuities—are induced by explicitly having more (state) variables being assigned to zero. Assigning zero is definitively lower than any probability they will be assigned to otherwise. Formally, we obtain the following result.

Definition 5.17 A function f is *lower semi-continuous* at an instantiation val if for all $\epsilon > 0$ there exists a neighbourhood U around val s.t. $f[\mathsf{val}] \leq f[\mathsf{val}'] + \epsilon$ for all $\mathsf{val}' \in U$.

Theorem 5.18 For each polynomial pMC $\mathcal{D}$, the function $^{\mathbb{P}}\mathsf{sol}_T^{\mathcal{D}}$ is lower semi-continuous. For each acyclic polynomial pMC $\mathcal{D}$, $^{\mathbb{P}}\mathsf{sol}_T^{\mathcal{D}}$ is continuous on $\mathsf{Val}_{\mathcal{D}}^{\mathsf{wd}}$.

Proof. For each $\mathsf{val} \in \mathsf{Val}_{\mathcal{D}}^{\mathsf{wd}}$, S may be partitioned into: $S = T \uplus S_{=0}^{\mathsf{val},T} \uplus S'$. For conciseness, first assume that the pMC is acyclic. We use structural induction over the graph of the pMC.

- For $s \in T$, $^{\mathbb{P}}\mathsf{sol}_{s \to T}^{\mathcal{D}}$ is constant and thus continuous.
- For $s \in S_{=0}^{\mathsf{val},T}$, $^{\mathbb{P}}\mathsf{sol}_{s \to T}^{\mathcal{D}}[\mathsf{val}] = 0$ and $^{\mathbb{P}}\mathsf{sol}_{s \to T}^{\mathcal{D}}$ is non-negative, so it is certainly lower semi-continuous.
- For $s \in S'$, $^{\mathbb{P}}\mathsf{sol}_{s \to T}^{\mathcal{D}} = \sum_{s' \in S} \mathcal{P}(s, s') \cdot {}^{\mathbb{P}}\mathsf{sol}_{s' \to T}^{\mathcal{D}}$. By induction, $^{\mathbb{P}}\mathsf{sol}_{s' \to T}^{\mathcal{D}}$ is lower semi-continuous for each $s' \in S$. Then, the (weighted) sum of lower semi-continuous functions is also lower semi-continuous on U.

For cyclic pMCs, observe that in each strongly connected component (SCC), either all states have probability zero to reach a target, or none. Then, apply the structural induction on the level of SCCs (which, for graph-preserving instantiations are preserved). More formally, we may consider a new pMC in which the SCCs have been

contracted. SCC-contraction preserves reachability probabilities. The new pMC is acyclic and thus amenable for the induction above.

To prove continuity on acyclic MCs, consider again the structural induction. As we are no longer interested in extending this to cyclic MCs, we do consider the absorbing, non-target states $S_{\circlearrowright}$ instead of $S_{=0}^{\mathsf{val}'}$. On acyclic MCs, $S_{\circlearrowright}$ is independent of parameter values. For each $\mathsf{val} \in \mathsf{Val}_{\mathcal{D}}^{\mathrm{wd}}$, S then may be partitioned into: $S = T \uplus S_{\circlearrowright} \uplus S'$. The structural induction simplifies to:

- For $s \in T$, ${}^{\mathbb{P}}\mathsf{sol}_{s \to T}^{\mathcal{D}} = 1$ is constant and thus continuous.
- For $s \in S_{\circlearrowright}$, ${}^{\mathbb{P}}\mathsf{sol}_{s \to T}^{\mathcal{D}} = 0$ is constant and thus continuous.
- For $s \in S'$, ${}^{\mathbb{P}}\mathsf{sol}_{s \to T}^{\mathcal{D}} = \sum_{s' \in S} \mathcal{P}(s, s') \cdot {}^{\mathbb{P}}\mathsf{sol}_{s' \to T}^{\mathcal{D}}$. By induction, ${}^{\mathbb{P}}\mathsf{sol}_{s' \to T}^{\mathcal{D}}$ is continuous for each $s' \in S$. Then, the (weighted) sum of continuous functions is also continuous on U. ■

This construction cannot be applied to cyclic pMCs, as SCCs might be part of the $S_{=0}^{\mathsf{val},T}$ states. The presence of such states would induce multiple solutions.

The following results follow immediately from the theorem: First, solution functions for acyclic pMCs naturally extend from the graph-preserving to all well-defined valuations.

Corollary 5.19 For each acyclic polynomial pMC $\mathcal{D}$, ${}^{\mathbb{P}}\mathsf{sol}_{T}^{\mathcal{D}}$ is described by a polynomial—even on $\mathsf{Val}_{\mathcal{D}}^{\mathrm{wd}}$.

For acyclic pMCs, combining some of our earlier observations we propose the following class of polynomials.

Definition 5.18 (Adequate polynomials) A polynomial $f \in \mathbb{Q}[X]$ is *adequate* if $0 < f[\mathsf{val}] < 1$ for all $\mathsf{val} \colon X \to (0, 1)$, and $0 \leq f[\mathsf{val}] \leq 1$ for all $\mathsf{val} \colon X \to [0, 1]$. ·adequate

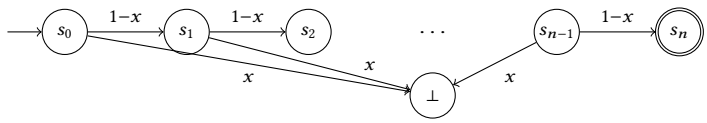
The following lemma formalises their relevance. It is an immediate consequence of Corollary 5.19 and Proposition 4.7.

Lemma 5.20 Let $\mathcal{D}$ be a simple acyclic pMC. The function ${}^{\mathbb{P}}\mathsf{sol}_{T}^{\mathcal{D}}$ is either adequate, constantly zero, or constantly one.

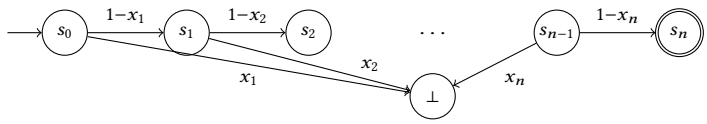
Second, while the maximum over well-defined valuations does not necessarily exist, the minimum always exists:

Corollary 5.21 Let $\mathcal{D}$ be a pMC. Then there is a valuation $\mathsf{val} \in \mathsf{Val}_{\mathcal{D}}^{\mathrm{wd}}$ such that ${}^{\mathbb{P}}\mathsf{sol}_{T}^{\mathcal{D}}[\mathsf{val}] \leq {}^{\mathbb{P}}\mathsf{sol}_{T}^{\mathcal{D}}[\mathsf{val}']$ for all $\mathsf{val}' \in \mathsf{Val}_{\mathcal{D}}^{\mathrm{wd}}$.

Third, these properties carry over to solution functions for pMDPs.



(a) Small univariate pMC with dense solution function.



(b) Small multivariate pMC with exponentially larger solution function.

Corollary 5.22 For all pMDPs, the functions ${}^{\mathbb{P}}\mathsf{minsol}_T^{\mathcal{M}}$ and ${}^{\mathbb{P}}\mathsf{maxsol}_T^{\mathcal{M}}$ are lower semi-continuous and have a minimum. For acyclic pMDPs, these functions are continuous.

Furthermore, we observe the following:

Lemma 5.23 Let $\mathcal{D}$ be a pMC with targets T and $R \subseteq \mathsf{Val}_{\mathcal{D}}^{\mathrm{gp}}$ connected. Let $f/g := {}^{\mathbb{P}}_{\mathrm{gp}}\mathsf{sol}_T^{\mathcal{D}}$. Then either

$$g[\mathsf{val}] > 0 \text{ or } g[\mathsf{val}] < 0 \text{ for all } \mathsf{val} \in R.$$

Proof. Observe that g is a polynomial and thus continuous, and has no roots in $\mathsf{Val}_{\mathcal{D}}^{\mathrm{gp}}$ as f/g is bounded from above and below. Therefore, the sign of g is invariant on $\mathsf{Val}_{\mathcal{D}}^{\mathrm{gp}}$. ■

5.2.1.2 Size of the solution function for graph-preserving valuations

Solution functions are a concise description of how parameter valuations affect the induced reachability probability. In many examples, they are (purposefully) compact. Yet, we already saw in the Knuth-Yao example (Example 4.7 on page 69) that the solution function may grow significantly. In this section, we give upper and lower bounds on the degree and the number of monomials in the solution function (for the graph-preserving part). The main result we present is that *computing the solution function is exponential (only) in the number of parameters.*

Example 5.16 Consider the pMC in Figure 5.12a. The solution function is

$$(1-x)^n = \sum_{i=0}^{n} (-1)^{n-i} \binom{n}{i} \cdot x^i.$$

The solution function obtained by substituting x by $1 - y$ is y^n: It has the same degree but only one monomial.

Intuitively (for affine pMCs), the degree of the numerator is capped by multiplying the polynomials along the longest cycle-free path: additional cycles are only rescaling the probabilities. The longest cycle-free path is capped by the number of states. The bound is formally stated below.

Proposition 5.24 ([HBK17]) Let $\mathcal{D}$ be a polynomial pMC and let $f/g := {}^{\mathbb{P}}_{\mathrm{gp}}\mathsf{sol}^{\mathcal{D}}_{s\to T}$. Both f and g have total degree at most $|S| \cdot d$, where d is the largest total degree occurring in transitions of $\mathcal{D}$.

The proof uses the linear equation system together with Cramer's rule to express the rational function as the fraction of two matrix determinants, and the Leibniz formula for matrix determinants to prove the bound. The example also illustrates that even for a high degree, the function may be, but not is not necessarily, large. The following formalises simple combinatorics applied on the maximal degree of the solution function.

Proposition 5.25 ([HBK17]) Let $\mathcal{D}$ be an affine pMC with n states and let $f/g := {}^{\mathbb{P}}_{\mathrm{gp}}\mathsf{sol}^{\mathcal{D}}_{s\to T}$. Both f and g have at most $\binom{n+k}{k}$ monomials.

We also have an exponential lower bound for the number of monomials.

Proposition 5.26 There is a family $(\mathcal{D})_{k\geqslant 2}$ of acyclic simple pMCs where $\mathcal{D}_n$ has n parameters and $n{+}2$ states, such that ${}^{\mathbb{P}}_{\mathrm{gp}}\mathsf{sol}^{\mathcal{D}}_{T}$ is a polynomial for which even the shortest sum-of-monomial representation has 2^n monomials.

Proof (sketch). Consider the pMC in Figure 5.12b. The pMC $\mathcal{D}_n := (S, \iota, X, \mathcal{P})$ has states $S := \{s_0, \ldots, s_n, \bot\}$ with $\iota := s_0$, parameters $X := \{x_1, \ldots, x_n\}$ and $\mathcal{P}$ is given by

$$\mathcal{P}(s, s') := \begin{cases} 1 - x_i & \text{if } s = s_{i-1} \text{ and } s' = s_i, \\ x_i & \text{if } s = s_{i-1} \text{ and } s' = \bot, \\ 1 & \text{if } s = s' = \bot \text{ or } s = s' = s_n, \\ 0 & \text{otherwise.} \end{cases}$$

The solution function is:

$${}^{\mathbb{P}}_{\mathrm{gp}}\mathsf{sol}^{\mathcal{D}}_{T} = \prod_{i=1}^{n}(1 - x_i) = \sum_{I \subseteq \{1,\ldots,n\}} -1^{|I|} \cdot \prod_{i \in I} x_i.$$

In expanded form, this function has exponentially many monomials. ■

5

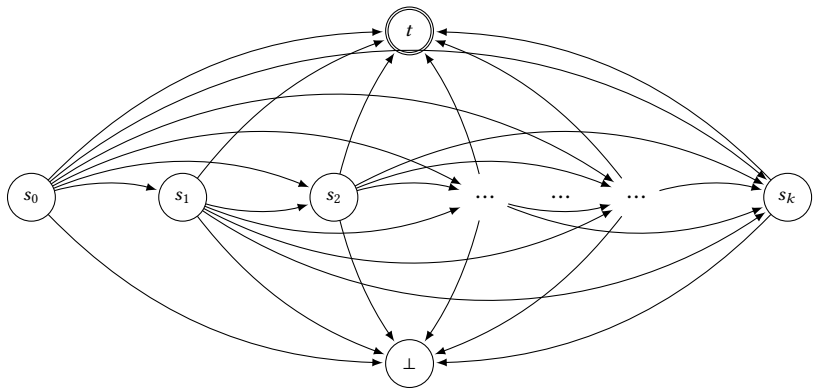


Figure 5.13: Graph structure of an acyclic pMC with transition probabilities $\frac{1}{k+2}$ for transitions from s_0 to any other state, x_i for transitions from state s_i, $i = 1, \ldots, k$, to t, $\frac{1}{k-i+1} \cdot (1 - x_i)$ for transitions from s_i to either $\bot$ or s_j with $j > i$.

The following slightly more complex construction from [HBK17] shows that exponential bounds may also be obtained for functions that cannot be factorised straightforwardly: We construct a family of pMCs $\mathcal{D}_k$ as in Figure 5.13. In particular, $\mathcal{D}_k = (S, \iota, X, \mathcal{P})$ with $S \coloneqq \{s_0, \ldots, s_k, \bot, t\}$, $\iota \coloneqq s_0$, and

$$\mathcal{P}(s, s') = \begin{cases} \frac{1}{k+2} & \text{if } s = s_0,\ s' \neq s_0, \\ x_i & \text{if } s = s_i,\ s' = t,\ 0 < i \leq k, \\ \frac{1-x_i}{k-i+1} & \text{if } s = s_i,\ s' = s_j \text{ with } 0 < i < j \leq k, \text{ or } s = s_i,\ s' = \bot,\ 0 < i \leq k, \\ 0 & \text{otherwise.} \end{cases}$$

By an inductive argument, we obtain:

$${}^{\mathbb{P}}_{\mathrm{gp}}\mathsf{sol}^{\mathcal{D}}_{s \to T} = \frac{1}{k+2} + \frac{1}{k+2} \cdot \sum_{\substack{1 \leqslant m \leqslant k \\ i_1 < \ldots < i_m \leqslant k}} x_{i_m} \cdot \prod_{j=1}^{m-1} \frac{1 - x_{i_j}}{k - i_j + 1}.$$

Thus[1], the number of monomials is 2^k. Using ideas from Section 5.1, we can also turn this pMC into an equivalent simple pMC by just doubling the states.

As a consequence of the size of the solution function, there is no hope for a polynomial-time algorithm that computes the sum-of-terms representation of the solution function. However, for any fixed number of parameters, the function is only

[1]The full deviation of the solution function and this latter form may be found in [20].

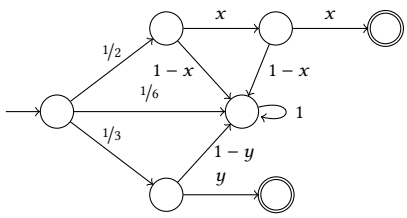


Figure 5.14: pMC for the polynomial $\frac{1}{2}x^2 + \frac{1}{3}y$

polynomially large. Indeed, the runtime of one particular algorithm is only exponential in the number of parameters k.

Theorem 5.27 ([HBK17]) The function ${}^{\mathbb{P}}_{\mathrm{gp}}\mathsf{sol}^{\mathcal{D}}_{T}$ is computable in $\mathcal{O}(\mathsf{poly}(n \cdot d)^k)$ where n is the number of states, d is the maximal degree of a transition probability, and k is the number of parameters in the pMC.

This theorem is a corollary to Proposition 8.1, details about this algorithm are given there (in Section 8.1 on page 207).

5.2.2 From polynomials to solution functions

We continue by showing that a large class of polynomials may be described by pMCs. More precisely, we consider the question: Given a polynomial f, does there exist a (simple, acyclic) pMC such that ${}^{\mathbb{P}}\mathsf{sol}^{\mathcal{D}}_{T} = f$? We start with a positive example.

Example 5.17 The polynomial $f = \frac{1}{2}x^2 + \frac{1}{3}y$ corresponds to the solution function of the pMC in Figure 5.14.

The polynomial in the example is easy to translate, in particular, all coefficients are positive and they sum up to a value less than one. In the pMC, all transitions of the form $1 - x$ (for any parameter x) go to the sink state immediately. To handle negative coefficients, we are going to make a more flexible use of the $1 - x$ transitions. We first reformulate the polynomials.

Lemma 5.28 ([Cho17]) Let $f \in \mathbb{Q}[\boldsymbol{x}]$ be a polynomial. We can rewrite f as:

$$f = \sum_{i=1}^{m} a_i \cdot h_i + b \qquad \text{with } h_i = \prod_{1 \le j \le k} x_j^{e_{i,j}} \cdot (1 - x_j)^{e'_{i,j}} \tag{5.1}$$

with $a_i \in \mathbb{Q}_{>0}$, $e_{i,j}, e'_{i,j} \in \mathbb{N}$, and $b \in \mathbb{Q}$.

Proof. Observe that a monomial $-x_1 \cdot \ldots \cdot x_d$ of degree $d \geq 0$ may be written as

$$-x_1 \cdot \ldots \cdot x_d = -1 + \sum_{i=1}^{d} (1 - x_i) \cdot x_{i+1} \cdot \ldots \cdot x_d, \tag{5.2}$$

which is proved by induction on d: For $d = 0$, both sides are -1 (an empty product equals 1). For $d \geq 0$, we multiply both sides of (5.2) by x_{d+1} to obtain

$$\begin{aligned} -x_1 \cdot \ldots \cdot x_d \cdot x_{d+1} &= -x_{d+1} + \sum_{i=1}^{d} (1 - x_i) \cdot x_{i+1} \cdot \ldots \cdot x_d \cdot x_{d+1} \\ &= (1 - x_{d+1}) - 1 + \sum_{i=1}^{d} (1 - x_i) \cdot x_{i+1} \cdot \ldots \cdot x_d \cdot x_{d+1} \\ &= -1 + \sum_{i=1}^{d+1} (1 - x_i) \cdot x_{i+1} \cdot \ldots \cdot x_d \cdot x_{d+1}. \end{aligned}$$

Hence applying (5.2) to every term of f we obtain Equation (5.2) where the a_i are *positive* rational coefficients, the h_i are nonempty products of terms from $\{x, (1 - x) \mid x \in X\}$ and $b \in \mathbb{Q}$ is a constant term. We may assume that $b \leq 0$, otherwise $b = b \cdot x + b \cdot (1 - x)$ for any $x \in X$ and we may pull b inside the sum. ■

Example 5.18 Consider $f = 1 - \frac{1}{2}x^2y$. We reformulate this to:

$$\frac{1}{2} \cdot \big((1 - x)xy + (1 - x)y + (1 - y) - 1\big) + 1$$

Some adequate polynomials in the obtained form have a negative b: Consider

$$f' = -0.25x^3 + 0.25x^2 + 0.5x + 0.15 = 0.25(1 - x)^3 + 0.5x(1 - x) + 0.75x - 0.1.$$

Following the construction in the proof, we obtain $0.25\big((1 - x)x^2 + (1 - x)x + (1 - x)\big) + 0.25x^2 + 0.5x + 0.15$. The example also shows that this form is not canonical.

In fact, this translation allows to translate and scale a polynomial f such that there exists a pMC $\mathcal{D}$ with targets T and

$$\frac{f + A}{B} = {}^{\mathbb{P}}\mathsf{sol}_T^{\mathcal{D}} \quad \text{for some } A \in \mathbb{Q}_{\geq 0}, B \in \mathbb{Q}_{>0}$$

Example 5.19 Consider the polynomial $-2x^2y + y$. We reformulate this to:

$$2 \cdot ((1 - x)xy + (1 - x)y + (1 - y) - 1) + y$$

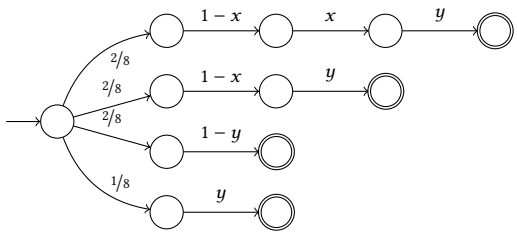


Figure 5.15: pMC with ${}^{\mathbb{P}}_{\mathrm{gp}}\mathsf{sol}^{\mathcal{D}}_{T} = {}^{-2x^2y+y+2/8}$

and then to

$$2 \cdot (1-x)xy + 2 \cdot (1-x)y + 2 \cdot (1-y) + y - 2.$$

After shifting upwards (with +2) and rescaling (with $\frac{1}{8}$), we can construct the pMC $\mathcal{D}$ depicted in Figure 5.15.

Formally, we show the following slightly more general proposition.

Proposition 5.29 ([Cho17]) Let f be a polynomial. For any A and B sufficiently large, there exists a pMC $\mathcal{D}$ with targets T such that

$$\frac{f+A}{B} = {}^{\mathbb{P}}\mathsf{sol}^{\mathcal{D}}_{T}.$$

Moreover, if d is the total degree of f, t the number of terms in f and κ a bound on the (bit-)size of the coefficients and the thresholds μ, λ, then $\mathcal{D}$ may be constructed in time $O(poly(d, t, \kappa))$.

Proof. Recall that f may be written as

$$f = \sum_{i=1}^{m} a_i \cdot h_i + b$$

with $a_i \in \mathbb{Q}_{\geq 0}$ and $b \in \mathbb{Q}_{<0}$. Let $A > b$. We can write

$$f + A = \sum_{i=1}^{m} a_i \cdot h_i + b'$$

with $a_i \in \mathbb{Q}_{\geq 0}$ and $b' = (b + A) \in \mathbb{Q}_{\geq 0}$. Let $B > \sum_{i=1}^{m} a_i + b'$. We can write

$$\tilde{f} := \frac{f+A}{B} = \sum_{i=1}^{m} \tilde{a}_i \cdot h_i + \tilde{b}$$

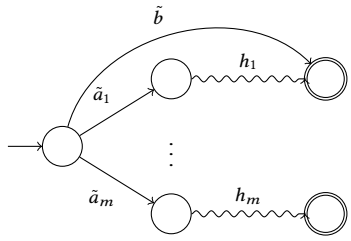


Figure 5.16: The essential construction of the pMC in Proposition 5.29: Any probability mass not drawn goes to a sink.

with $\tilde{a}_i, \tilde{b} \in \mathbb{Q}_{\geq 0}$ and $\sum_{i=1}^{m} \tilde{a}_i + \tilde{b} < 1$. The modified polynomial $\tilde{f}$ naturally corresponds to a simple acyclic pMC $\tilde{\mathcal{D}}$ with ${}^{\mathbb{P}}\mathsf{sol}_T^{\mathcal{D}} = \tilde{f}$ as shown in Figure 5.16.

For the complexity of the construction, notice that m in the sum (5.1) is in $\mathcal{O}(td)$ where d and t are bounds on the total degree and the number of terms of f, respectively. The h_i are products of at most d terms. The a_i are the absolute values of the original coefficients of f and b is the sum of at most t of those coefficients. Hence a_i, b, A, B and the polynomial $\tilde{f}$ may be computed in time $\mathcal{O}(poly(t, d, \kappa))$. The same then also holds for the pMC $\tilde{\mathcal{D}}$. ■

The proposition is essential for later statements on the shape of accepting sets. For now, we consider how we can do better than the previous result. First, we show that principally, we may get rid of the additive factor A above, by relying on the following deep result.

Lemma 5.30 (Handelman's theorem [Han88]) Let $g_1 \geqslant 0, \ldots, g_m \geqslant 0$ be affine constraints that define a closed convex polyhedron $\mathcal{H} \subseteq \mathbb{R}^k$ with interior. If a polynomial $f \in \mathbb{R}[\boldsymbol{x}]$ is strictly positive on $\mathcal{H}$, then f may be written as

$$f = \sum_{i=1}^{n} a_i \cdot h_i \qquad \text{with } h_i = g_1^{e_{i,1}} \cdot \ldots \cdot g_m^{e_{i,m}}$$

for some exponents $e_{i,j} \in \mathbb{N}$ and coefficients $a_i \in \mathbb{R}_{>0}$.

Applied to our scenario, we obtain the following statement:

Corollary 5.31 If a polynomial $f \in \mathbb{R}[\boldsymbol{x}]$ is strictly positive on $[0, 1]^{\boldsymbol{x}}$, then f may be written as

$$f = \sum_{i=1}^{n} a_i \cdot h_i \qquad \text{with } h_i = \prod_{1 \leq j \leq k} x_j^{e_{i,j}} \cdot (1 - x_j)^{e'_{i,j}} \tag{5.3}$$

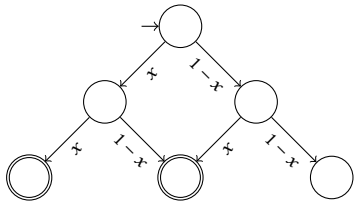


Figure 5.17: Construction of pMCs for Example 5.20

for some exponents $e_{i,j}, e'_{i,j} \in \mathbb{N}$ and coefficients $a_i \in \mathbb{R}_{>0}$.

Observe that the (original) proof to this theorem is not constructive. Using the Handelman representation from above, we may also avoid rescaling. Scaling is necessary in the construction from Proposition 5.29 as we need a distribution over the products h_i, and then have one path for each h_i. An alternative construction allows multiple paths for each h_i:

Example 5.20 Consider the (adequate) polynomial $f := 2x - x^2 = 2x + (1 - x)x + (1 - x) - 1 = 2x(1 - x) + x^2$. The coefficients in the latter form sum up to 3, and the construction of Proposition 5.29 requires rescaling, yet f is the solution function of the pMC in Figure 5.17.

A formal construction is (only briefly) outlined below.

Theorem 5.32 (Winkler's trick) Let $f \in \mathbb{Q}[x]$ be a univariate adequate polynomial. There exists a simple acyclic pMC $\mathcal{D}$ with a target state T such that $f = {}^{\mathbb{P}}\mathsf{sol}_T^{\mathcal{D}}$.

Example 5.21 Consider $f = 2x \cdot (1 - x) + \frac{1}{4}$ which is strictly positive on $[0, 1]$ and already in a Handelman representation. Following the proof of Lemma 5.33, f may be reformulated as:

$$f = \frac{1}{4}\binom{3}{0}x^3 + \frac{11}{12}\binom{3}{1}x^2(1 - x) + \frac{11}{12}\binom{3}{2}x(1 - x)^2 + \frac{1}{4}\binom{3}{3}(1 - x)^3.$$

The construction (described in the proof of Theorem 5.32) yields the pMC depicted in Figure 5.18.

The construction is based on the following result: Form (5.3) is called a *Handelman representation* of f w.r.t. $g_1, \ldots, g_m$. The next lemma states the existence of a specific

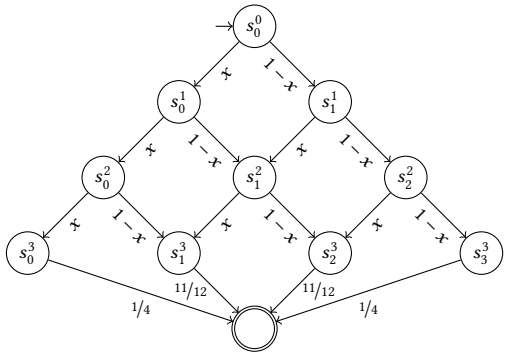


Figure 5.18: Construction of pMCs for Example 5.21

Handelman representation which we can map to a pMC. We refer to [19] for the rather technical proof.

Lemma 5.33 Let $f \in \mathbb{Q}[x]$ be an adequate polynomial with a Handelman representation. There exists an $n \geq 0$ such that

$$f = \sum_{i=0}^{n} a_i \cdot \binom{n}{i} \cdot x^{n-i} \cdot (1-x)^i \qquad \text{with } a_i \in [0,1] \text{ for all } 0 \leq i \leq n. \tag{5.4}$$

We prove Theorem 5.32 by reformulating the Handelman representation.

Proof (Theorem 5.32). We reformulate f as $x^e(1-x)^d f'$ for some integers $e, d \geq 0$ and a polynomial f' which is strictly positive on $[0,1]$. We then (magically) obtain the Handelman representation for f' by Corollary 5.31. Consequently, f is then also in Handelman representation. We apply Lemma 5.33 and obtain

$$f = \sum_{i=0}^{n} a_i \cdot \binom{n}{i} \cdot x^{n-i} \cdot (1-x)^i \quad \text{for some } n \geq 0 \text{ and } a_i \in [0,1] \text{ for all } 0 \leq i \leq n.$$

We now construct a pMC $\mathcal{D}_n(a_0, \ldots, a_n) := (S, \iota, X, \mathcal{P})$ with a pyramid like topology from Example 5.21. Formally, $S := \{s_i^j \mid 0 \leq i \leq j \leq n\} \cup \{t, \bot\}$, $\iota := s_0^0$, $X := \{x\}$, and

$$\mathcal{P}(s,s') := \begin{cases} x & \text{if } s = s_i^j \text{ and } s' = s_i^{j+1} \text{ for some } i,j, \\ 1-x & \text{if } s = s_i^j \text{ and } s' = s_{i+1}^{j+1} \text{ for some } i,j, \\ a_i & \text{if } s = s_i^n \text{ and } s' = t, \\ 1-a_i & \text{if } s = s_i^n \text{ and } s' = \bot, \\ 0 & \text{otherwise.} \end{cases}$$

Let $T \coloneqq \{t\}$. Figure 5.18 exemplifies the construction by showing $\mathcal{D}_3(1/4, 11/12, 11/12, 1/4)$. By induction on n and using the basic identity

$$\binom{n}{i} = \binom{n-1}{i-1} + \binom{n-1}{i} \text{ for all } 0 \leq i \leq n,$$

there are $\binom{n}{i}$ many paths, each with probability $x^{n-i}(1-x)^i$, from ι to s_i. Thus:

$${}^{\mathbb{P}}\mathsf{sol}^{\mathcal{D}_n(\ldots)}_{\{s_i^n\}} = \binom{n}{i} \cdot x^{n-i}(1-x)^i,$$

and by summing over all states at level n:

$${}^{\mathbb{P}}\mathsf{sol}^{\mathcal{D}_n(a_0 \ldots a_n)}_{T} = \sum_{i=0}^{n} \left(a_i \cdot \binom{n}{i} \cdot x^{n-i}(1-x)^i \right) = f.$$

■

A constructive version of the Handelman construction, a generalisation to multivariate polynomials, and complexity bounds for the procedure remain open for future work.

5.2.3 Accepting sets for pMCs

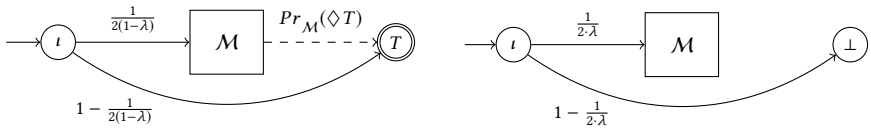
The essential result in this subsection is that polynomials and accepting sets of simple pMCs are in a one-to-one correspondence. Before we return to considering polynomials, we formulate the following trivial but important auxiliary statement.

Proposition 5.34 Let $\mathcal{M}$ be a pMDP and $\varphi = \mathbb{P}_{\bowtie\lambda}(\Diamond T)$ be a property with arbitrary target states T and $0 < \lambda < 1$, $\lambda \in \mathbb{Q}$. Now consider $\varphi' = \mathbb{P}_{\bowtie 1/2}(\Diamond T)$. There exists a constantly larger pMDP $\mathcal{M}'$ such that for any quantifier Q:

$$\mathsf{acc}^{\mathcal{M}}_{Q,\varphi} = \mathsf{acc}^{\mathcal{M}'}_{Q,\varphi'}.$$

Proof (sketch). The constructions are depicted in Figure 5.19: If $\lambda \leq 1/2$ then we prepend a transition with probability $p = 1/2\lambda$ to the initial state and with probability $1-p$ to a sink state. Otherwise, if $\lambda > 1/2$, we prepend a transition with probability $q = 1/2(1-\lambda)$ to the initial state and $1-q$ to the target state. Conversely, the $1/2$ threshold may analogously be reduced to an arbitrary threshold $0 < \lambda < 1$. ■

We first consider acyclic pMCs. Section 4.3.4 on page 100 shows that the accepting set can be described by a conjunction of the region constraints and the solution function. The region constraints of simple pMCs are independent of the particular pMC, the solution function is a polynomial (Corollary 5.19). Thus, this polynomial describes the accepting set of a simple acyclic pMC. Interestingly, for any polynomial, we can also construct a pMC such that the accepting set is described by this polynomial. The key ingredient is the pMC for scaled polynomials (Proposition 5.29). Scaling is not an issue on constraints, as the scaling can be applied on both sides of the inequality.



(a) $\lambda \leq 1/2$ (b) $\lambda > 1/2$

Figure 5.19: Reductions to reachability threshold $\lambda = 1/2$, cf. Proposition 5.34.

Example 5.22 Recall Example 5.19 and consider the inequality $-2x^2y + y \geqslant 5$. We reformulate this inequality to:

$$2 \cdot ((1-x)xy + (1-x)y + (1-y) - 1) + y \geqslant 5$$

and then to

$$2 \cdot (1-x)xy + 2 \cdot (1-x)y + 2 \cdot (1-y) + y \geqslant 7.$$

Observe that both sides of the inequality now only contain positive coefficients. Furthermore, observe that we wrote the left-hand side as sum of products over $\{x, 1-x, y, 1-y\}$. After rescaling (with $\frac{1}{8}$), we can construct the pMC $\mathcal{D}$ depicted in Figure 5.15 and set $\lambda = \frac{7}{8}$.

We formalise the construction as follows:

Theorem 5.35 (Chonev's trick) ([Cho17, Remark 7]) Let $f \in \mathbb{Q}[\boldsymbol{x}]$ be a polynomial, $\mu \in \mathbb{Q}$ and $0 < \lambda < 1$. There exists a simple acyclic pMC $\mathcal{M}$ with parameters $\boldsymbol{x}$ and target states T such that for all $\mathsf{val} \in \mathsf{Val}^{\mathrm{wd}}_{\mathcal{D}}$ and all comparison relations $\bowtie \in \{<, \leq, \geq, >\}$ it holds that

$$f[\mathsf{val}] \bowtie \mu \quad \text{iff} \quad {}^{\mathbb{P}}\mathsf{sol}^{\mathcal{D}}_{T}[\mathsf{val}] \bowtie \lambda.$$

Moreover, if d is the total degree of f, t the number of terms in f and κ a bound on the (bit-)size of the coefficients and the thresholds μ, λ, then $\mathcal{D}$ may be constructed in time $O(poly(d, t, \kappa))$.

We avoid the restriction to $\mathsf{Val}^{\mathrm{gp}}_{\mathcal{D}}$ by applying Corollary 5.19.

Proof. We apply Proposition 5.29 on f to construct $\tilde{f}$ using $B > \max\{\sum_{i=1}^{m} a_i, \mu - b\}$ and $A = -b$. We have that

$$f[\mathsf{val}] \bowtie \mu \quad \text{iff} \quad \tilde{f}[\mathsf{val}] \bowtie \frac{\mu + A}{B} =: \lambda'. \tag{5.5}$$

for all valuations $\mathsf{val} \in \mathbb{R}^X$ and all comparison relations $\bowtie$. The new threshold λ' is strictly between 0 and 1. We now modify $\tilde{\mathcal{D}}$ as outlined in Proposition 5.34 to obtain a pMC $\mathcal{D}$ such that for all $\mathsf{val} \in \mathsf{Val}^{\mathrm{wd}}_{\mathcal{D}}$ it holds that for the given threshold $0 < \lambda < 1$

$$f[\mathsf{val}] \bowtie \mu \quad \text{iff} \quad {}^{\mathbb{P}}\mathsf{sol}^{\mathcal{D}}_{T}[\mathsf{val}] \bowtie \lambda. \qquad \blacksquare$$

The tight correspondence between polynomials and accepting sets yields various interesting constructions. We showcase one such construction showing that *the restriction to acyclic pMCs is not a real restriction*:

Proposition 5.36 For any non-simple pMC $\mathcal{D}$ with $\mathsf{Val}^{\mathrm{gp}}_{\mathcal{D}} = (0,1)^X$, $\varphi = \mathbb{P}_{\bowtie\lambda}(\Diamond T)$ with target states T and $\lambda \in \mathbb{Q}$ there exists a simple acyclic pMC $\mathcal{D}'$ such that

$$\mathsf{acc}^{\mathcal{D}}_{\varphi} = \mathsf{acc}^{\mathcal{D}'}_{\varphi}.$$

In the fixed parameter case, $\mathcal{D}'$ may be computed in polynomial time.

Proof. Let $\mathcal{D}$ be a nonsimple pMC. Compute the solution function, which is a rational function, say $\frac{h}{g}$. Now fix some $\mathsf{val} \in \mathsf{Val}^{\mathrm{gp}}_{\mathsf{val}}$. W.l.o.g., assume $g[\mathsf{val}] > 0$, see Lemma 5.23. Let $f := h - (g-1) \cdot \lambda$. By construction

$${}^{\mathbb{P}}_{\mathrm{gp}}\mathsf{sol}^{\mathcal{D}}_{T}[\mathsf{val}] \bowtie \lambda \text{ iff } f[\mathsf{val}] \bowtie \lambda.$$

By applying Theorem 5.35 on $f \bowtie \lambda$, there exists a simple acyclic pMC $\mathcal{D}'$ and $\lambda' \in \mathbb{Q}$ such that

$${}^{\mathbb{P}}_{\mathrm{gp}}\mathsf{sol}^{\mathcal{D}'}_{T}[\mathsf{val}] \bowtie \lambda'.$$

Closer inspection yields that $\lambda' = \lambda$. The application of Definition 4.8 finishes the construction. For a fixed number of parameters, this transformation has polynomial time complexity, as the solution function is computable in polynomial time (Theorem 5.27) and the pMC from Theorem 5.35 is constructible in polynomial time too. $\blacksquare$

5.3 Somewhere Optimal Strategies

Optimal strategies are an essential concept in MDPs. For pMDPs, there is in general no universally optimal strategy. However, some strategies are optimal for some parameter valuation, while other strategies might be suboptimal for all valuations. Intuitively, to decide feasibility, it suffices to consider all pMCs induced by a somewhere optimal strategy. Therefore, we consider these strategies in some more detail.

Definition 5.19 Let $\mathcal{M}$ be a pMDP. A strategy $\sigma \in \mathfrak{S}^{\mathcal{M}}$ is *somewhere optimal*, if

·somewhere optimal

$$\text{there exists a } M \in \langle \mathcal{M} \rangle \text{ such that } \sigma \text{ is optimal on } M.$$

We adapt the Bellman characterisation for optimal strategies from the parameter-free

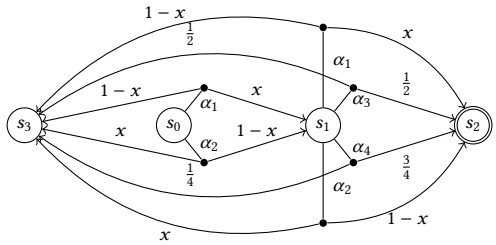


Figure 5.20: pMDP for Example 5.23

case, cf. Proposition 3.9. Such an adaption has been first given in [FA09]. Intuitively, the characterisation ensures that in no state, changing the strategy yields an improvement.

Example 5.23 Consider the pMDP in Figure 5.20. For $\mathsf{val} := \{x \mapsto \frac{3}{4}\}$, the strategy $\{s_0, s_1 \mapsto \alpha_2\}$ is minimal. For $\mathsf{val} := \{x \mapsto \frac{1}{2}\}$, the strategy $\{s_0, s_1 \mapsto \alpha_2\}$ is minimal, but also, e.g., the strategy $\{s_0 \mapsto \alpha_1, s_1 \mapsto \alpha_3\}$. In total, four strategies are somewhere optimal.

Lemma 5.37 (Somewhere minimal) Let $\mathcal{M}$ be a pMDP with region $R \subseteq \mathsf{Val}^{\mathrm{gp}}_{\mathcal{M}}$. Let $\sigma \in \mathfrak{S}^{\mathcal{M}}$ and let $f_s := {}^{\mathbb{P}}\mathsf{sol}^{\mathcal{M}[\sigma]}_{s \to T}$ for any $s \in S$. The strategy σ is somewhere minimal, iff there exists some $\mathsf{val} \in R$ such that for all $s \in S$ and $\alpha \in Act$:

$$f_s[\mathsf{val}] \leq \sum_{s' \in S} \mathcal{P}(s, \alpha, s') \cdot f_{s'}[\mathsf{val}].$$

The characterisation for somewhere maximal is analogous. For conciseness, we only consider graph-preserving regions here. However, both notions and characterisations can be extended. The characterisation gives rise to two sets of constraints: The first encoding avoids the solution functions. It is based on a standard ETR encoding for pMCs as in Constraints 4.5.

Constraints 5.1 Let $\mathcal{M}$ be a pMDP and $\sigma \in \Sigma^{\mathcal{M}}$ a strategy. Let $S_{=0} := \{s \in S \mid {}^{\mathbb{P}}_{\mathrm{gp}}\mathsf{sol}^{\sigma}_{\mathcal{M}} T = 0\}$. Consider the real-valued variables $\{p_s \mid s \in S\}$ and variables for the parameters:

$$
\begin{aligned}
p_s &= 1 && \text{for all } s \in T,\\
p_s &= 0 && \text{for all } s \in S_{=0},\\
p_s &= \sum_{s' \in S} \mathcal{P}(s, \sigma(\alpha), s') \cdot p_{s'} && \text{for all } s \in S \setminus (S_{=0} \cup T),\\
p_s &\leq \sum_{s' \in S} \mathcal{P}(s, \alpha, s') \cdot p_{s'} && \text{for all } s \in S \setminus (S_{=0} \cup T), \alpha \in \mathbb{A}\mathrm{ct}(s).
\end{aligned}
$$

We refer to the corresponding formula as $\Phi^{\mathrm{ETR}}_{\min}(\mathcal{M}, \sigma)$.

Analogously to the standard encoding, we can utilise solution functions and thereby avoid the large set of auxiliary variables. On the downside, such an encoding may be exponentially large in the number of parameters.

Constraints 5.2 Let $\mathcal{M}$ be a pMDP with region $R \subseteq \mathsf{Val}^{\mathrm{gp}}_{\mathcal{M}}$. Let $\sigma \in \Sigma^{\mathcal{M}}$ and let $h_s/g_s := {}^{\mathbb{P}}\mathsf{sol}^{\mathcal{M}[\sigma]}_{s \to T}$ for any $s \in S$. The constraints over variables for all parameters are:

$$
h_s[\mathsf{val}] \cdot \prod_{s'' \neq s} g_{s''}[\mathsf{val}] \;\leq\; \sum_{s' \in S} \mathcal{P}(s, \alpha, s') \cdot h_{s'}[\mathsf{val}] \cdot \prod_{s'' \neq s'} g_{s''}[\mathsf{val}] \text{ for all } s \in S, \alpha \in Act.
$$

We refer to the corresponding formula as $\Phi^{\mathsf{sol}}_{\min}(\mathcal{M}, \sigma)$.

Proposition 5.38 Let $\mathcal{M}$ be a pMDP. A strategy $\sigma \in \mathfrak{S}^{\mathcal{M}}$ is somewhere minimal on region $R \subseteq \mathsf{Val}^{\mathrm{gp}}_{\mathcal{M}}$ iff

$$
\mathsf{sat}(\psi_R \wedge \Phi^{\mathsf{sol}}_{\min}(\mathcal{M}, \sigma)) \neq \emptyset, \text{ or equivalently,} \quad \mathsf{sat}(\psi_R \wedge \Phi^{\mathrm{ETR}}_{\min}(\mathcal{M}, \sigma)) \neq \emptyset.
$$

Example 5.24 Consider the pMDP $\mathcal{M}$ in Figure 5.20 together with $\sigma := \{s_0, s_1 \mapsto \alpha_2\}$. Using the encoding from Constraints 5.2, we obtain:

$$
0 \lessdot x \lessdot 1 \wedge (1-x)^2 \leqslant x \cdot (1-x) \wedge 1 - x \leqslant x \wedge 1 - x \leqslant \frac{1}{2} \wedge 1 - x \leqslant \frac{3}{4}.
$$

The formula is satisfied by $\mathsf{val} := \{x \mapsto \frac{2}{3}\}$, indicating that the strategy is minimal for $\mathcal{M}[\mathsf{val}]$.

The byproduct of a method in [HHZ11a] is a set of somewhere minimal strategies. It is often interesting to find the *set* of somewhere optimal strategies covering the entire parameter space—a so called *optimal-strategy set* (*OSS*). In particular, (maxim- OSS

5

al/minimal) solution functions of pMDPs with a small OSS may be rather concisely represented in a straightforward fashion.

minimising strategy set

Definition 5.20 (Optimal strategy set) A set $\Omega \subseteq \Sigma^{\mathcal{M}}$ is called a *minimising strategy set* on $R \subseteq \mathsf{Val}^{\mathrm{wd}}_{\mathcal{M}}$ if

$$\text{for all } M \in \langle \mathcal{M} \mid R \rangle \text{ there exists a } \sigma \in \Omega \text{ such that } Pr^{\sigma}_{M}(\Diamond T) = {}^{\mathbb{P}}\mathsf{minsol}^{\mathcal{M}}_{T},$$

i.e. Ω contains a minimal strategy for every valuation in R. The notion may be analogously defined for maximal strategies.

MOSS

We consider maximising and minimising strategy sets as optimal strategy sets. An OSS of minimal cardinality is called a *minimal optimal strategy set* (*MOSS*).

Example 5.25 Reconsider the pMDP from Example 5.24. The set $\Sigma^{\mathcal{M}}$ is trivially an OSS, but not a MOSS. The set $\{\{s_0, s_1 \mapsto \alpha_1\}, \{s_0, s_1 \mapsto \alpha_2\}\}$ is a minimal minimising strategy set.

To check whether a set is indeed an OSS, one can check whether there is a valuation where none of the strategies in the OSS is optimal. If yes, then the set is not an OSS, otherwise it is.

Proposition 5.39 Let $\mathcal{M}$ be a pMDP and $R \subseteq \mathsf{Val}^{\mathrm{gp}}_{\mathcal{M}}$. Using $\Phi^{*}_{\min}(\mathcal{M}, \sigma)$ from either Constraints 5.1 or Constraints 5.2. A set $\Omega \subseteq \Sigma^{\mathcal{M}}$ is a minimising strategy set on R iff

$$\mathsf{sat}\big(\psi_R \wedge \bigwedge_{\sigma \in \Omega} \neg \Phi^{*}_{\min}(\mathcal{M}, \sigma)\big) = \emptyset.$$

In general, the set of strategies of a (p)MDP is exponential. The MOSS of a pMDP may be exponential in the number of parameters (and in the number of actions):

Proposition 5.40 There exists a family $(\mathcal{M}_n)_{n \in \mathbb{N}}$ of simple pMDPs with $n+2$ states s.t. $|\Omega| \geq 2^n$ for any OSS Ω on $\mathsf{Val}^{\mathrm{wd}}_{\mathcal{M}_n}$, i.e., the size of a MOSS can grow exponentially in the pMDP's size.

Proof. Define $\mathcal{M}_n := (S_n, \iota, Act, X_n, \mathcal{P}_n)$ with $S_n := \{s_1, \ldots, s_{n+1}\} \uplus \{\bot\}$, $\iota := s_1$, $X_n := \{x_1, \ldots, x_n\}$, $Act := \{\alpha, \beta\}$, and

$$\mathcal{P}_n(s_i, \alpha, s_{i+1}) := x_i, \quad \mathcal{P}_n(s_i, \beta, s_{i+1}) := 1 - x_i,$$

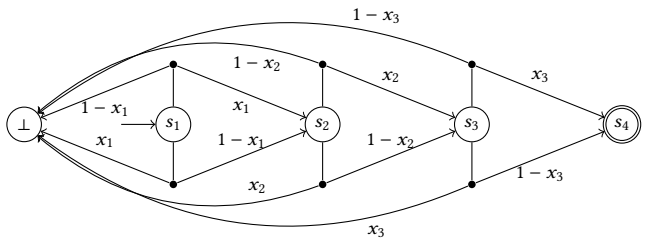


Figure 5.21: The instance $\mathcal{M}_3$ of the proof for Proposition 5.40 with MOSS $\Sigma^{\mathcal{M}_3}$.

for all $1 \leq i \leq n$, and $T = \{s_{n+1}\}$. The remaining unspecified probability mass is directed to the sink $\perp$. Let us fix a $\sigma \in \mathfrak{S}^{\mathcal{M}}$ and let $\mathsf{val} \in \mathsf{Val}^{\mathrm{wd}}_{\mathcal{M}}$ be such that

$$\mathsf{val}(x_i) := \begin{cases} 1 & \text{if } \sigma(s_i) = \alpha, \\ 0 & \text{if } \sigma(s_i) = \beta. \end{cases}$$

Then clearly $Pr^{\sigma}_{M_n[\mathsf{val}]}(\Diamond T) = 1$ and $Pr^{\sigma'}_{M_n[\mathsf{val}]}(\Diamond T) = 0 < 1$ for all $\sigma' \neq \sigma$. Thus we have shown that for all strategies σ, there is an instantiation for which σ is the unique maximal strategy, so no such σ may be excluded from a MOSS. The lemma follows as there are $|\mathfrak{S}^{\mathcal{M}}| = 2^n$ many strategies σ. ■

Further bounds on the MOSS are open. Overall, this topic deserves more attention: For example, it seems that there are strategies that are optimal on sets with measure zero, and these strategies are indeed also not part of any MOSSes.

> **Remark 5.3** In Example 5.23, there also exists a single strategy $\sigma \in \mathfrak{S}^{\mathcal{M}}$ such that
>
> for all $D \in \langle \mathcal{M}[\sigma] \rangle$ it holds $D \models \mathbb{P}_{\geq \frac{1}{2}}(\Diamond T)$.
>
> This strategy σ just uniformly chooses an action in every state. We consider such *robust* strategies in Section 5.5.

5.4 Computational Complexity

After considering various traits of feasibility analysis in pMCs and pMDPs, we are well-equipped to discuss the computational complexity of the considered problems. We consider simple pMDPs, it avoids that the complexity is governed by asking for some well-defined valuation, which in general is an ETR-hard problem, cf. Theorem 5.2.

Encoding of the input | Let a *simple* pMDP $\mathcal{M}$ with and a set T of target states be the given input. We analyse the decision problems according to whether the set X of parameters from M has bounded size—with a-priori fixed bound—or arbitrary size. It remains for us to fix an encoding for rational functions. Henceforth, we assume the coefficients, exponents and constants are all given as binary-encoded integer pairs.

Decision problems | The set of problems are the feasibility (and validity) problems of Section 4.2, i.e., the existence of so-called *robust parameter values* or lack thereof. More precisely, the question is whether some instantiation of $\mathcal{M}$ is such that its maximal or minimal probability of eventually reaching T compares with $\lambda \in \{0, \frac{1}{2}, 1\}$ in some desired way. In symbols, for $Q_1, Q_2 \in \{\exists, \forall\}$ and $\bowtie \in \{\leq, <, >, \geq\}$, we consider the decision problem

$$Q_1 \ \mathsf{val} \in \mathsf{Val}^{\mathrm{wd}}_{\mathcal{M}}, Q_2 \ \sigma \in \mathfrak{S}^{\mathcal{M}} : \ Pr^{\sigma}_{\mathcal{M}[\mathsf{val}]}(\Diamond T) \bowtie \lambda.$$

Structure. | The section is summarised by two tables: the complexity for the qualitative problems—where $\lambda \in \{0, 1\}$—in Table 5.1 and for the quantitative problems—where $\lambda = \frac{1}{2}$—in Table 5.2 on page 146. The results focus on feasibility, as validity problems are their dual. We follow the structure of the tables. Afterwards we give a brief interpretation of the results.

5.4.1 Qualitative analysis

We consider the existential quantification over the parameters, for either existential or universal quantification over the strategies. We consider different variants of $\bowtie$ and set the threshold λ either to 0 or to 1. Table 5.1 summarises the results for pMCs in dependence of $\bowtie$ and λ. We refer to the question whether there exists a pMC $D \in \langle \mathcal{D} \rangle$ with

unreachability
- an induced probability ≤ 0 as *unreachability*,

·positive
- an induced probability > 0 as *positive reachability*,

·almost sure
- an induced probability ≥ 1 as *almost sure reachability*

·unsure
- an induced probability < 1 as *unsure reachability*

Together, these problems are the existentially quantified qualitative reachability problems. The table lists for these four forms of reachability the complexity for a fixed number of parameters, and the complexity if the pMC $\mathcal{D}$ contains arbitrarily many parameters. For that case, we distinguish whether the parameter valuations range over the well-defined, graph-preserving or pure valuations.

We start with graph-preserving valuations. We already saw that in qualitative analysis, the graph-structure suffices to determine the feasibility (Proposition 4.7). If we only consider graph-preserving valuations this structure remains fixed, thus the

	Fixed # parameters	Arbitrary # parameters		
		graph-preserving	well-defined	pure
> 0	in P	in P Proposition 4.7	in P Theorem 4.8	NP-complete Proposition 5.42
< 1	”	”	NP-complete Proposition 5.44	”
≤ 0	”	”	NP-complete Theorem 5.43	”
≥ 1	”	”	”	”

Table 5.1: The complexity landscape for *qualitative reachability* in simple pMCs and pMDPs. Observe that the decision problems for pMCs and pMDPs (under either angelic or demonic nondeterminism) are different, but (with respect to the considered classes) the categorisation coincides. Unlisted combinations of comparison operators and thresholds yield trivial decision problems.

qualitative problems are essentially equivalent to the parameter-free counterparts and therefore in P, both for pMCs and for pMDPs. For a fixed number of parameters, even when ranging over all well-defined instantiations, there are only constantly many different graph-consistent regions when the number of parameters is fixed (Proposition 5.15). Consequently, the feasibility problem reduces to a constant number of problems in P, both for pMCs and for pMDPs. For positive reachability under well-defined instantiations, we have seen that a single path from source to target suffices. Intuitively, the more transitions exist, the better (Theorem 4.8). Therefore, this property can be decided by considering a graph-preserving instantiation (if that exists). Therefore, for well-defined instantiations, positive reachability is in P, both for pMCs and pMDPs. All other problems are in NP:

Proposition 5.41 The existentially quantified qualitative reachability problems for pMDPs are all decidable in NP.

Proof (sketch). Recall that we have simple pMDPs. Guess the graph structure by deciding whether a parameter is assigned 0, $\frac{1}{2}$ or 1. Then solve the qualitative problem on the MDP. ■

What remains in this section are NP-completeness proofs for pMCs. As pMCs are pMDPs with just one strategy, these hardness results carry over to the pMDPs.

Let us consider positive reachability for pure valuations. This hardness result depends crucially on the absence of graph-preserving instantiations and is inspired by a construction in [HBK17]. Indeed, we prove the following stronger statement.

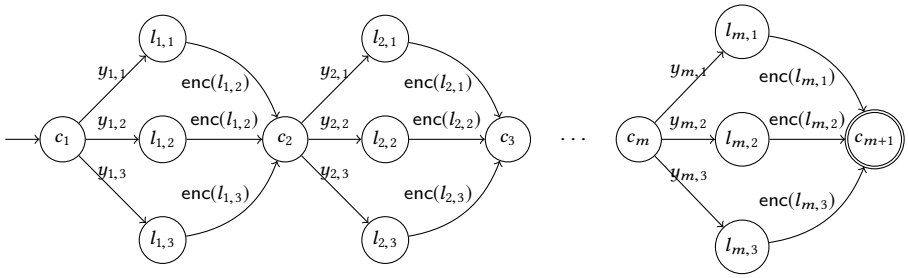


Figure 5.22: pMC construction for NP-hardness of positive reachability in pMCs.

Proposition 5.42 Given an acyclic simple pMC $\mathcal{D}$, deciding any existentially quantified qualitative reachability problems for pure valuations is NP-hard.

Proof. We show a reduction from 3SAT. We consider first positive reachability. Let

$$\psi := c_1 \wedge \ldots \wedge c_m$$

be a given 3SAT-formula, i.e. the clauses c_i are of the form

$$c_i = l_{i,1} \vee l_{i,2} \vee l_{i,3},$$

where the $l_{i,j}$ are *literals* (variables or negated variables). Let $\boldsymbol{x} = \{x_1, \ldots, x_k\}$ be the variables of ψ. The pMC for ψ is outlined in Figure 5.22. Formally, the pMC $\mathcal{D} := (S, \iota, X, \mathcal{P})$ is defined as follows: The $4m + 2$ many states:

$$S := \{c_i \mid 1 \leq i \leq m+1\} \cup \{l_{i,j} \mid 1 \leq i \leq m, 1 \leq j \leq 3\} \cup \{\bot\} with \iota := c_1,$$

$3m + k$ parameters

$$X := \{\tilde{x} \mid x \in \boldsymbol{x}\} \cup \{y_{i,j} \mid 1 \leq i \leq m, 1 \leq j \leq 3\},$$

and with transitions

$$\mathcal{P}(s, s') := \begin{cases} y_{i,j} & \text{if } s = c_i, s' = l_{i,j} \text{ for some } 1 \leq i \leq m, 1 \leq j \leq 3, \\ \mathsf{enc}(l_{i,j}) & \text{if } s = l_{i,j}, s' = c_{i+1} \text{ for some } 1 \leq i \leq m, 1 \leq j \leq 3, \\ 1 - \mathsf{enc}(l_{i,j}) & \text{if } s = l_{i,j}, s' = \bot \text{ for some } 1 \leq i \leq m, 1 \leq j \leq 3, \\ 0 & \text{otherwise,} \end{cases}$$

using

$$\mathsf{enc}(l_{i,j}) := \begin{cases} \tilde{x} & \text{if } l_{i,j} = x, \\ 1 - \tilde{x} & \text{if } \neg l_{i,j} = x. \end{cases}$$

The target states are $T = \{c_{m+1}\}$. This construction runs in polynomial time.

Let $\varphi := \mathbb{P}_{\geq 0}(\Diamond T)$. We show:

$$\mathsf{sat}(\psi) \neq \emptyset \quad \text{iff} \quad \exists D \in \langle \mathcal{D} \mid \text{pure}, \mathsf{acc}_\varphi \rangle.$$

Intuitively, the variables $l_{i,j}$ should represent the witness literal for each satisfied clause, i.e., the literal that makes the clause true. The parameters $\tilde{x}$ correspond to the x variables in the 3SAT-formula as follows: For a valuation val of variables $\boldsymbol{x}$ in ψ and a valuation val' of X such that $\mathsf{val}'(\tilde{x}) = 1$ iff $\mathsf{val}(x) = \texttt{true}$ it holds that:

$$\mathsf{enc}(l_{i,j})[\mathsf{val}] = 1 \quad \text{iff} \quad \mathsf{val}'(l_{i,j}) = \texttt{true}.$$

Formally, first assume that there exists an assignment $\mathsf{val} \in \mathsf{sat}(\psi)$. Then, this assignment makes at least one literal $l_{i,*}$ in every clause c_i true. We consider val' with the corresponding $y_{i,*}$ assigned to 1 and $\tilde{x}$ assigned 1 iff $\mathsf{val}(x) = \texttt{true}$. Then, in $\mathcal{D}[\mathsf{val}']$, there is a path from ι to T.

Now assume that there exists an MC $D \in \langle \mathcal{D} \mid \text{pure} \rangle$ with a path from ι to T. Observe that this path in D is the only path to the target. We construct a satisfying assignment $\mathsf{val} \in \mathsf{sat}(\psi)$. This path goes through a set of $l_{i,*}$. These become the witness literals that make all the clauses true. The assignment to the variables x are obtained from the occurrences of $\tilde{x}$ along the path, or equivalently, by lookup from the witness literals in ψ.

For unreachability, almost-sure and unsure reachability observe that the probability to reach c_{m+1} in $\mathcal{D}$ is either zero or one for any pure valuation. The proofs are straightforward adaptions. ■

Finally, we consider well-defined instantiations. We already argued that positive reachability is in P. We now show that all other qualitative problems are NP-complete.

Theorem 5.43 ([Cho17]) Given a simple pMC $\mathcal{D}$, deciding:

is there a $D \in \langle \mathcal{D} \rangle$ such that $Pr_D(\Diamond T) = 1$?

or deciding:

is there a $D \in \langle \mathcal{D} \rangle$ such that $Pr_D(\Diamond T) = 0$?

are both NP-hard.

We deliberately show the proof from [Cho17] rather than adapting the construction

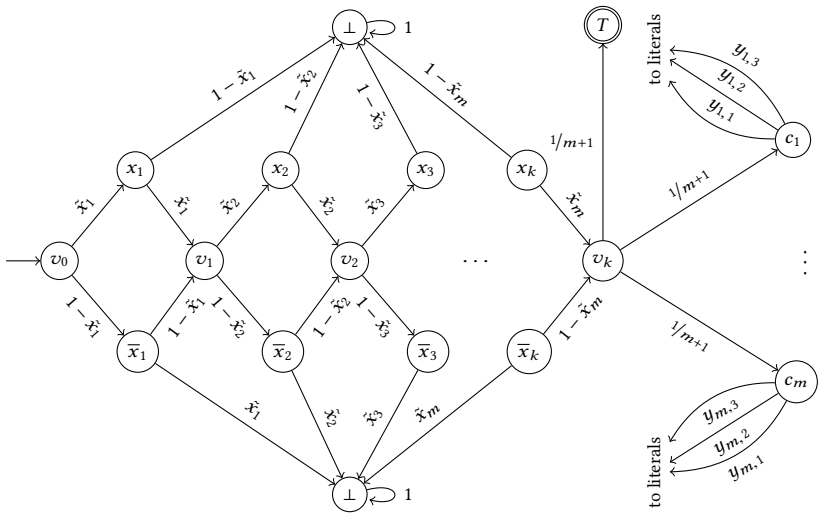


Figure 5.23: pMC construction for NP-hardness of almost-sure reachability in pMCs.

of Proposition 5.42, as this proof itself is a crucial step towards Proposition 5.52. We reformulate the construction in our notation. The essential idea here is to enforce pure valuations.

Proof. We reduce from 3SAT. Let

$$\psi = c_1 \wedge \ldots \wedge c_m$$

be a given 3SAT-formula, i.e. the c_i are of the form

$$c_i = l_{i,1} \vee l_{i,2} \vee l_{i,3},$$

where the $l_{i,j}$ are *literals* (variables or negated variables). Let $\boldsymbol{x} = \{x_1, \ldots, x_k\}$ be the variables of ψ. The pMC for ψ is outlined in Figure 5.23. Formally, the pMC $\mathcal{D} = (S, \iota, X, \mathcal{P})$ is defined as follows:

$$S \coloneqq \{v_i \mid 0 \leq i \leq k\} \uplus \{x_i, \overline{x_i} \mid 1 \leq i \leq k\} \uplus \{c_i \mid 1 \leq i \leq m\} \uplus \{T, \bot\}$$

are the $3m + k + 3$ states, $v_0 = \iota$ is the initial state, T and $\bot$ indicate target and sink respectively,

$$X \coloneqq \{\tilde{x} \mid x \in \boldsymbol{x}\} \cup \{y_{i,j} \mid 1 \leq i \leq m, 1 \leq j \leq 3\}$$

are the $k + 3m$ parameters, for all $1 \leq i \leq m$ and $1 \leq j \leq k$ we define the transition probabilities as

$$\begin{aligned}
&\mathcal{P}(v_{i-1}, x_i) := \tilde{x}_i, && \mathcal{P}(v_{i-1}, \overline{x_i}) := 1 - \tilde{x}_i,\\
&\mathcal{P}(x_i, v_i) := \tilde{x}_i, && \mathcal{P}(\overline{x_i}, v_i) := 1 - \tilde{x}_i,\\
&\mathcal{P}(x_i, \bot) := 1 - \tilde{x}_i, && \mathcal{P}(\overline{x_i}, \bot) := \tilde{x}_i,\\
&\mathcal{P}(v_m, c_i) := \frac{1}{k+1}, && \mathcal{P}(v_m, T) := \frac{1}{k+1},\\
&\mathcal{P}(c_j, x_i) := y_{j,r} \text{ if } l_{j,r} = x_i \text{ (in } \psi), && \mathcal{P}(c_j, \overline{x}_i) := y_{j,r} \text{ if } l_{j,r} = \overline{x}_i \text{ (in } \psi).
\end{aligned}$$

We let $\mathcal{P}(s, t) = 0$ for each pair (s, t) of states not specified above.

Observe that under any well-defined valuation, there are exactly two bottom strongly connected components, namely $\bot$ and T. As a consequence, it holds that:

$$\text{for all } D \in \langle \mathcal{D} \rangle : \quad Pr_D(\Diamond T) + Pr_D(\Diamond\{\bot\}) = 1. \tag{5.6}$$

Let $\varphi := \mathbb{P}_{\geq 1}(\Diamond T)$. We show:

$$\mathsf{sat}(\psi) \neq \emptyset \quad \text{iff} \quad \exists D \in \langle \mathcal{D} \mid \mathsf{acc}_\varphi \rangle.$$

First, assume ψ is satisfiable. Choose some satisfying assignment val for ψ. We construct $\mathsf{val}' \in \mathsf{Val}^{\mathrm{wd}}_{\mathcal{D}}$ in two steps. First, let $\mathsf{val}'(\tilde{x}_i) = \mathsf{val}(x_i) \in \{0, 1\}$ for all $1 \leq i \leq k$. Thus, $\bot$ is unreachable. Second, for each clause c_i, select one literal $l_{i,j}$ which makes c_i true, and set $\mathsf{val}'(y_{i,j}) = 1$. Set all other $y_{i,j}$ to 0. We conclude using (5.6) that $Pr_{\mathcal{D}[\mathsf{val}]}(\Diamond T) = 1$.

Now, assume there exists an MC satisfying φ. Consider a val' such that $\mathcal{D}[\mathsf{val}'] \models \varphi$. Then, using (5.6), no path leads to $\bot$. For $\tilde{x}_1$, that means that necessarily $\mathsf{val}'(\tilde{x}) \in \{0, 1\}$. For the concrete val', we can choose for each c_i a literal $l_{i,j}$ (a $y_{i,j}$ set to 1) such that we surely do reach v_m again. These $l_{i,j}$ are exactly the witness literals making every clause `true`. Then, the assignment for $\tilde{x}$ gives rise to a satisfying valuation $\mathsf{val} \in \mathsf{sat}(\psi)$. Observe that the proof immediately also shows the NP-hardness for unreachability.■

Proposition 5.44 Given a simplex pMC $\mathcal{D}$, deciding:

is there a $D \in \langle \mathcal{D} \rangle$ such that $Pr_D(\Diamond T) < 1$?

is NP-hard.

Proof (sketch). We omit a full technical proof, it uses some of the ideas similar as above. The essential trick is to reuse the pMC from Proposition 5.42 and extend it with a transition from c_{m+1} to c_1. Consider the bottom state as target. Only if c_{m+1} is reached with probability 1, no probability leaks to the bottom state, and the target is reached

		Fixed # parameters	Arbitrary # parameters	
			well-defined	graph-preserving
pMC	$\exists\textsc{reach}^{\geq/\leq}$	in P [HBK17] [Theorem 5.58]	— ETR-complete [Theorem 5.49] —	
	$\exists\textsc{reach}^{>}$	”	NP-hard [Proposition 5.52]	$\exists\textsc{reach}^{>}_{\text{wd}}$-complete [Lemmas 5.47 and 5.48]
	$\exists\textsc{reach}^{<}$	”	NP-hard [Proposition 5.52]	$\exists\textsc{reach}^{>}_{\text{wd}}$-complete [Lemma 5.46]
pMDP	$\exists\exists\textsc{reach}^{\geq/\leq}$	in NP [Proposition 5.59]	— ETR-complete — (trivial)	
	$\exists\exists\textsc{reach}^{>}$	”	— $\exists\textsc{reach}^{>}_{\text{wd}}$-complete [Proposition 5.53, Corollary 5.55] —	
	$\exists\exists\textsc{reach}^{<}$	”	$\exists\textsc{reach}^{<}_{\text{wd}}$-complete [Proposition 5.53]	$\exists\textsc{reach}^{>}_{\text{wd}}$-complete [Lemma 5.46]
	$\exists\forall\textsc{reach}^{\bowtie}$	in NP [Proposition 5.60]	— ETR-complete [Theorem 5.56] —	

Table 5.2: The complexity landscape for *quantitative* reachability in simple pMDPs. All problems are in ETR.

with probability 0. Otherwise, the target is the only SCC and the probability to reach that becomes 1. ■

5.4.2 Quantitative reachability

Table 5.2 surveys the results. We use the following notation for conciseness: For $Q_1, Q_2 \in \{\exists, \forall\}$ and $\bowtie \in \{\leq, <, >, \geq\}$, let

$$Q_1 Q_2 \textsc{reach}^{\bowtie}_{\text{wd}} \overset{\text{def}}{\iff} Q_1\ \mathsf{val} \in \mathsf{Val}^{\text{wd}}_{\mathcal{M}}, Q_2\ \sigma \in \mathfrak{S}^{\mathcal{M}} :\ Pr^{\sigma}_{\mathcal{M}[\mathsf{val}]}(\Diamond T) \bowtie \frac{1}{2}$$

be the problem of interest. We have fixed a threshold of $\frac{1}{2}$. This is without loss of generality as any *given* rational threshold may be reduced to $\frac{1}{2}$, cf. Proposition 5.34. We write $Q_1 Q_2 \textsc{reach}^{\bowtie}_{\text{gp}}$ whenever Q_1 quantifies over graph-preserving instantiations. We write $Q_1 Q_2 \textsc{reach}^{\bowtie}_{*}(\lambda)$ to denote both the wd and gp variants. Furthermore, if $\mathcal{M}$ is a pMC we omit the second quantifier, e.g. $\exists\textsc{reach}^{<}_{*}$.

Following the results from Section 4.3.2, we have an ETR upper bound.

Corollary 5.45 For every $Q_1, Q_2 \in \{\exists, \forall\}$ $Q_1 Q_2 \textsc{reach}^{\bowtie}_{*}$ are decidable in ETR.

Remark 5.4 For pure valuations, we even have an NP upper bound by guessing which parameters are assigned to one. Therefore, the lower bound for the qualitative and the upper bound for the quantitative case coincide.

We first reduce some entries in the table to each other. The remainder of this section then first considers hardness results for the case of arbitrary parameters, and then shows better upper bounds for the case where the number of parameters is fixed.

Considerations for the comparison relations. | The number of combinations that we need to consider is significantly reduced by a couple of reductions that follow from structural properties of pMCs. Intuitively, the first property is based on the duality of target and bad states:

Lemma 5.46 For every $Q_1, Q_2 \in \{\exists, \forall\}$, there are polynomial-time Karp reductions

- among the problems $Q_1Q_2\textsc{reach}^{>}_{\text{gp}}$ and $Q_1Q_2\textsc{reach}^{<}_{\text{gp}}$ and
- among the problems $Q_1Q_2\textsc{reach}^{\geq}_{\text{gp}}$ and $Q_1Q_2\textsc{reach}^{\leq}_{\text{gp}}$.

Proof. We prove only the first item for arbitrary Q_1 and $Q_2 = \exists$. First, we deduce from [BK08, Thm. 10.122 and Thm. 10.127] that in polynomial time, and without regarding the actual transition probabilities, we can compute from $\mathcal{M}$ and a target set T, a target set[1] T' such that for each $M \in \langle \mathcal{M} \mid \text{gp} \rangle$:

$$\max_{\sigma \in \Sigma^{\mathcal{M}}} Pr^{\sigma}_{M}(\Diamond T) = 1 - \min_{\sigma \in \Sigma^{\mathcal{M}}} Pr^{\sigma}_{M}(\Diamond T').$$

Please observe that the step above in general does not work without the restriction to graph-preserving instantiations. We combine this idea with a transition from a quantifier to maximising schedulers and obtain

$$\begin{aligned}
&Q_1\mathsf{val} \in \mathsf{Val}^{\text{gp}}_{\mathcal{M}},\ \exists \sigma \in \Sigma^{\mathcal{M}} :\ Pr^{\sigma}_{M[\mathsf{val}]}(\Diamond T) > \frac{1}{2} \\
&\text{iff } Q_1\mathsf{val} \in \mathsf{Val}^{\text{gp}}_{\mathcal{M}} :\ \max_{\sigma \in \Sigma^{\mathcal{M}}} Pr^{\sigma}_{M[\mathsf{val}]}(\Diamond T) > \frac{1}{2} \\
&\text{iff } Q_1\mathsf{val} \in \mathsf{Val}^{\text{gp}}_{\mathcal{M}} :\ \left(1 - \min_{\sigma \in \Sigma^{\mathcal{M}}} Pr^{\sigma}_{M[\mathsf{val}]}(\Diamond T')\right) > \frac{1}{2} \\
&\text{iff } Q_1\mathsf{val} \in \mathsf{Val}^{\text{gp}}_{\mathcal{M}} :\ \min_{\sigma \in \Sigma^{\mathcal{M}}} Pr^{\sigma}_{M[\mathsf{val}]}(\Diamond T') < \frac{1}{2} \\
&\text{iff } Q_1\mathsf{val} \in \mathsf{Val}^{\text{gp}}_{\mathcal{M}}, \exists \sigma \in \Sigma^{\mathcal{M}} :\ Pr^{\sigma}_{M[\mathsf{val}]}(\Diamond T') < \frac{1}{2}.
\end{aligned}$$

[1] Which is some adequate union of particular maximal end components in $\mathcal{M}$

5

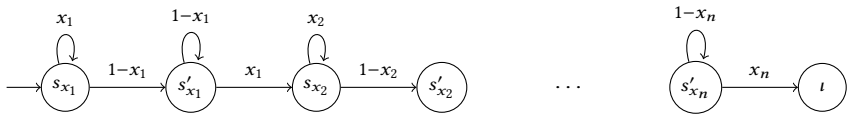


Figure 5.24: Gadget for the proof of Lemma 5.48

All other cases may be proven analogously. ■

For strict lower-bounded reachability, we can restrict our attention to graph-preserving parameter instantiations.

Lemma 5.47 $\exists\textsc{reach}^{>}_{\text{wd}}$ is polynomially reducible to $\exists\textsc{reach}^{>}_{\text{gp}}$.

The lemma is an immediate consequence of the semi-continuity of the solution function, cf. Theorem 5.18. The proof in Theorem 5.43 contains a construction that avoids graph-preserving valuations. Similar in spirit, we can also construct gadgets that avoid valuations which are not graph-preserving. The gadget in Figure 5.24 ensures that for any graph non-preserving instantiation, the probability to reach the target is 0, while it does not affect reachability probabilities for graph-preserving instantiations. Together with semi-continuity of the solution function, we deduce:

Lemma 5.48 $\exists\textsc{reach}^{>}_{\text{gp}}$ is polynomially reducible to $\exists\textsc{reach}^{>}_{\text{wd}}$, and $\exists\textsc{reach}^{\geq}_{\text{gp}}$ is polynomially reducible to $\exists\textsc{reach}^{\geq}_{\text{wd}}$.

Proof. Given a simple pMC $\mathcal{D}$. We extend the pMC with a gadget outlined in Figure 5.24. Formally, we construct a pMC $\mathcal{D}'$ with states $S' := S \cup \{s_x, s'_x \mid x \in X\}$, initial state s_{x_1} and

$$\mathcal{P}'(s,s') := \begin{cases} \mathcal{P}(s,s) & \text{if } s,s' \in S, \\ x & \text{if } s = s' = s_x, \\ 1-x & \text{if } s = s' = s'_x, \\ 1-x & \text{if } s = s_x \text{ and } s' = s'_x, \\ x & \text{if } s = s'_x \text{ and } s' = \text{next}(s'_x), \\ 0 & \text{otherwise.} \end{cases}$$

where $\text{next}(s'_x)$ is s_{x+1} if $x = x_i$ for some $i < |X|$, and ι if $i = |X|$. The pMC $\mathcal{D}'$ is only linearly larger than $\mathcal{D}$. Observe that the construction of the gadget may be adapted

for non-simple pMCs (with different well-defined parameter spaces). By construction, we have for every $\mathsf{val} \in \mathsf{Val}^{\mathrm{wd}}_{\mathcal{D}}$:

$$Pr_{\mathcal{D}'[\mathsf{val}]}(\Diamond T) = Pr_{\mathcal{D}'[\mathsf{val}]}(\Diamond\{\iota\}) \cdot Pr_{\mathcal{D}[\mathsf{val}]}(\Diamond T).$$

We observe the following:

$$\text{for all } \mathsf{val} \in \mathsf{Val}^{\mathrm{gp}}_{\mathcal{D}} : Pr_{\mathcal{D}'[\mathsf{val}]}(\Diamond\{\iota\}) = 1 \text{ and thus } Pr_{\mathcal{D}'[\mathsf{val}]}(\Diamond T) = Pr_{\mathcal{D}[\mathsf{val}]}(\Diamond T),$$

and

$$\text{for all } \mathsf{val} \in \mathsf{Val}^{\mathrm{gp}}_{\mathcal{M}} \setminus \mathsf{Val}^{\mathrm{wd}}_{\mathcal{M}} : Pr_{\mathcal{D}[\mathsf{val}]}(\Diamond\{\iota\}) = 0.$$

Together, we deduce:

$$\text{exists } \mathsf{val} \in \mathsf{Val}^{\mathrm{gp}}_{\mathcal{D}}\ Pr_{\mathcal{D}[\mathsf{val}]}(\Diamond T) \trianglerighteq \lambda \quad \text{iff} \quad \text{exists } \mathsf{val} \in \mathsf{Val}^{\mathrm{wd}}_{\mathcal{D}'} :\ Pr_{\mathcal{D}'[\mathsf{val}]}(\Diamond T) \trianglerighteq \lambda. \ \blacksquare$$

We are not aware of any such reductions for upper bounds.

5.4.2.1 pMCs with arbitrarily many parameters

We first consider the upper right part of Table 5.2: Feasibility in pMCs with an unbounded number of parameters.

Non-strict inequalities | This paragraph is devoted to the following theorem:

Theorem 5.49 $\exists\textsc{reach}^{\leq}_{*}$, $\exists\textsc{reach}^{\geq}_{*}$ are all ETR-complete (even for acyclic pMCs).

For this result, we reduce from an ETR-hard problem. We consider the following problem.

Definition 5.21 The decision problem *modified-closed-bounded-4-feasibility* (mb4FEAS-c) asks: Given a (non-negative) quartic polynomial f,

$$\text{exists } \mathsf{val} \colon X \to [0,1] \text{ s.t. } f[\mathsf{val}] \leq 0?$$

The *modified-open-bounded-4-feasibility* (mb4FEAS-o) is analogously defined with val ranging over $(0,1)$.

This problem easily reduces to its $\geq$-variant by multiplying f with -1.

Lemma 5.50 The problems mb4FEAS-c and mb4FEAS-o are ETR-hard.

We have included the proof in Appendix B.2 on page 332. Essentially, one reduces from the existence of common roots of quadratic polynomials lying in a unit ball, which is

known to be ETR-complete [Sch13, Lemma 3.9]. The reduction to mb4FEAS follows the reduction[1] between unconstrained variants (i.e., variants in which the position of the root is not constrained) of the same decision problems [SS17, Lemma 3.2].

Proof (Theorem 5.49). The reduction from mb4FEAS-c to $\exists\textsc{reach}^{\leq}_{\text{wd}}$ constructs for a polynomial a pMC by a straightforward application of Theorem 5.35 with $\mu = 0$ and $\lambda = \frac{1}{2}$. For $\exists\textsc{reach}^{\leq}_{\text{gp}}$, we reduce from the open variant and notice that as the construction in Theorem 5.35 preserves all satisfying instantiations $\mathsf{val}\colon X \to [0,1]$ it, in particular, also preserves them on the graph-preserving parameter space. For $\geq$, we apply Theorem 5.35 on $-f$. ■

Observe that there may be exactly one solution to mb4FEAS-o/c, and this solution may be irrational. In contrast, if there exists a solutions for a constraint $f \gtrdot 0$, then there exist infinitely many (rational) solutions. To the best of our knowledge, the complexity of a variant of mb4FEAS-o/c with strict bounds is open. Therefore, we have no ETR-hardness proof for $\exists\textsc{reach}$ with strict bounds. In general, *conjunctions* of strict inequalities are also ETR-complete [SS17]. We exploit this in the proof of Theorem 5.56 on page 153 .

Consequences | We established ETR completeness on simple acyclic pMCs. From these results, ETR completeness for feasibility problems of general pMCs follows. It is interesting to see that even though the solution function for cyclic models is significantly more complex, these problems all fall in the same complexity class.

Furthermore, a similar construction may be used for (sufficiently large[2], linear) subsets of the parameter space. In particular, that means that the same complexity bounds for well-defined valuations can be derived for ε-preserving valuations.

Strict inequalities | In this paragraph, the main result is:

Theorem 5.51 $\exists\textsc{reach}^{>}_{*}$ and $\exists\textsc{reach}^{<}_{*}$ are NP-hard.

Using Lemmas 5.47 and 5.48, we may restrict our attention to well-defined parameter spaces. Recall Theorem 5.43: Almost-sure reachability is NP-hard. A more refined analysis of the 3SAT-reduction yields:

Proposition 5.52 $\exists\textsc{reach}^{>}_{\text{wd}}$ and $\exists\textsc{reach}^{<}_{\text{wd}}$ are NP-hard.

We only sketch the important insight based on the proof of Theorem 5.43, a complete formulation is given in [19].

[1] Essentially the polynomial f in mb4FEAS is constructed by taking the sum-of-squares of the quadratic polynomials, and further operations are adequately shifting the polynomial.

[2] The bounds should be at least $\delta < 1$ apart, with δ encoded by at most exponentially many bits.

Proof (sketch). Reconsider the construction in Figure 5.23 on page 144. We first show the following claim to simplify our proof afterwards:
Auxiliary claim: If ψ is unsatisfiable, then for all $\mathsf{val} \in \mathsf{Val}^{\mathrm{wd}}_{\mathcal{D}}$ there exists some clause c_{i^*}, such that $\mathcal{P}(l_{i^*,j}, \bot)[\mathsf{val}] \geq \frac{1}{2}$ for all $j \in \{1,2,3\}$, or more formally

$$\begin{aligned} &\mathsf{sat}(\psi) = \emptyset \text{ iff} \\ &\forall \mathsf{val} \in \mathsf{Val}^{\mathrm{wd}}_{\mathcal{D}},\ \exists i^* \in \{1,\dots,m\}, \forall j \in \{1,2,3\} :\ \mathcal{P}(l_{i^*,j}, \bot)[\mathsf{val}] \geq \frac{1}{2}. \end{aligned} \tag{5.7}$$

Proof of the auxiliary claim: By contraposition. Suppose for some val and for every clause c_i there is a 'witness' literal $l_{i,j}$ with $\mathcal{P}(l_{i,j}, \bot)[\mathsf{val}] < \frac{1}{2}$. Together with the definition of $\mathcal{P}$, we conclude either

1. $l_{i,j}$ is a variable x, and $\mathsf{val}(\tilde{x}) > \frac{1}{2}$ or
2. $l_{i,j}$ is a negated variable $\overline{x}$, and $\mathsf{val}(\tilde{x}) < \frac{1}{2}$.

We construct a satisfying assignment for ψ: Consider an assignment val_ψ for ψ, with

$$\mathsf{val}_\psi(x) := \begin{cases} \texttt{true}, & \text{if } \mathsf{val}(\tilde{x}) > \frac{1}{2}, \\ \texttt{false}, & \text{if } \mathsf{val}(\tilde{x}) < \frac{1}{2}, \\ \text{arbitrary}, & \text{if } \mathsf{val}(\tilde{x}) = \frac{1}{2}. \end{cases}$$

In both case 1 and 2 above, val_ψ satisfies clause c_i. Thus ψ is satisfiable. By contraposition, statement (5.7) holds.

Proof for correctness of reduction: We only show:

$$\mathsf{sat}(\psi) = \emptyset \text{ iff for all } \mathsf{val} \in \mathsf{Val}^{\mathrm{wd}}_{\mathcal{D}} : Pr_{\mathcal{D}[\mathsf{val}]}(\Diamond T) \leq \frac{2}{3}, \tag{5.8}$$

which is equivalent to:

$$\mathsf{sat}(\psi) \neq \emptyset \text{ iff exists } \mathsf{val} \in \mathsf{Val}^{\mathrm{wd}}_{\mathcal{D}} : Pr_{\mathcal{D}[\mathsf{val}]}(\Diamond T) > \frac{2}{3}.$$

Again let ψ be unsat and fix a parameter valuation val and set $D := \mathcal{D}[\mathsf{val}]$. We show $D \models \mathbb{P}_{\leq \frac{2}{3}}(\Diamond T)$. Let i^* be like in the auxiliary claim (5.7). The idea here is that c_{i^*} is the (potentially only) unsatisfied clause. By construction of $\mathcal{D}$ and the auxiliary claim,

$$Pr_D(s \models \Diamond T) \leq 1 - \mathcal{P}(l_{i*,j}, \bot)[\mathsf{val}] \leq \frac{1}{2}$$

for all $j \in \{1,2,3\}$ and hence

$$Pr_D(s \models \Diamond T) = \sum_{j=1}^{3} \mathsf{val}(y_{i^*,j}) \cdot Pr_D(s \models \Diamond T) \leq \frac{1}{2} \sum_{j=1}^{3} \mathsf{val}(y_{i^*,j}) = \frac{1}{2}.$$

Consequently, for $Pr_D(s \models \Diamond T)$ it holds that

$$\begin{aligned} Pr_D(s \models \Diamond T) &= \mathcal{P}(v_m, c_{i^*}) \cdot Pr_D(s \models \Diamond T) + \sum_{i \neq i^*} \mathcal{P}(v_m, c_i) \cdot Pr_D(s \models \Diamond T) \\ &\leq \frac{1}{m+1} \cdot \frac{1}{2} + \frac{m-1}{m+1} = \frac{2m-1}{2(m+1)}. \end{aligned}$$

Plugging this into the equation

$$Pr_D(s \models \Diamond T) = \frac{1}{m+1} + Pr_D(s \models \Diamond T) \cdot Pr_D(s \models \Diamond T)$$

yields $Pr_D(s \models \Diamond T) \leq \frac{2}{3}$. All paths from v_0 to T go through v_m, thus:

$$Pr_D(\Diamond T) = Pr_D(s \models \Diamond T) \cdot Pr_D(s \models \Diamond T) \leq Pr_D(s \models \Diamond T) \leq \frac{2}{3}.$$

The remainder of the proof is analagous to the proof of Theorem 5.43. The proof for threshold 1/2 follows by application of Proposition 5.34. ■

5.4.2.2 pMDPs with arbitrarily many parameters

We now move to the lower-right corner of Table 5.2, and consider pMDPs without bound on the number of parameters. For the results, we have to distinguish whether the quantifier over the strategies is existential (leading to angelic nondeterminism) or universal (leading to demonic nondeterminism). We consider these cases separately.

Angelic nondeterminism | Recall from Definition 5.13 on page 116 that feasability in pMDPs with angelic nondeterminism can be reduced to pMCs with additional variables for the nondeterminism[1]. This reduction however requires an arbitrary range for the parameters. More formally, we obtain:

Proposition 5.53 There are polynomial-time Karp reductions among $\exists\textsc{reach}^{\bowtie}_{\text{wd}}$ and $\exists\exists\textsc{reach}^{\bowtie}_{\text{wd}}$.

The proposition merely applies the following corollary to Proposition 5.10:

[1] And, as it is the complement, region verification in pMDPs with demonic nondeterminism may be reduced likewise.

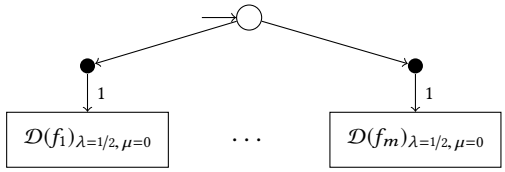


Figure 5.25: Construction for the proof of Theorem 5.56

Corollary 5.54 For simple pMDPs $\mathcal{M}$ and simple pMC $\mathcal{D} := \mathsf{simple}(\mathsf{ntp}(\mathcal{M}))$ we have:

$$\left(\exists M \in \langle M \rangle, \exists \sigma \in \Sigma^M : \; Pr_M^\sigma(\Diamond T) \bowtie \frac{1}{2}\right) \text{ iff } \left(\exists D \in \langle \mathcal{D} \rangle : \; Pr_D(\Diamond T) \bowtie \frac{1}{2}\right).$$

Minor adaptions to Proposition 5.10 and Lemma 5.48 yield:

Corollary 5.55 There are polynomial-time Karp reductions among $\exists\exists\textsc{reach}^{>}_{\text{gp}}$ and $\exists\textsc{reach}^{>}_{\text{wd}}$.

Demonic nondeterminism | We now consider demonic nondeterminism, in which we universally quantify over the strategies. Contrary to pMCs, we obtain ETR-completeness for pMDPs and any comparison relation:

Theorem 5.56 $\exists\forall\textsc{reach}^{\bowtie}_{*}$ are all ETR-complete (even for acyclic pMDPs with a single nondeterministic state).

Non-strict relations are already trivially ETR-hard via Theorem 5.49. For the strict relations, we reduce from the following problem.

Definition 5.22 The decision problem *bounded-conjunction-of-inequalities* (bcon4INEQ-c) asks: Given a family of quartic polynomials $f_1, \ldots, f_m$:

$$\text{exists } \mathsf{val} : X \to [0,1] \text{ s.t. } \bigwedge_{i=1}^{m} f_i[\mathsf{val}] < 0?$$

The open variant (bcon4INEQ-o) may be defined analogously.

By a reduction from mb4FEAS (adapted from [SS17, Thm 4.1]):

Lemma 5.57 The bcon4INEQ-o/c problems are ETR-hard.

The proof may be found in Appendix B.2 on page 332.

Proof (Theorem 5.56). We show the reduction from bcon4INEQ-o/c to $\exists\forall\textsc{reach}^{>}_{\text{wd}}$. For given $f_1, \ldots, f_m$, we construct pMCs

$$\mathcal{D}(f_1)_{\lambda=1/2,\,\mu=0}, \ldots, \mathcal{D}(f_m)_{\lambda=1/2,\,\mu=0} \text{ with target states } T_i$$

by applying Theorem 5.35 to f_i (with $\lambda = \frac{1}{2}$ and $\mu = 0$). Then, we construct a pMDP as outlined in Figure 5.25. We take the disjoint union of the pMCs and adding a fresh initial state, with nondeterministic actions into each pMC. Formally, let $\mathcal{D}(f_i)_{\lambda=1/2,\,\mu=0} = (S_i, \iota^i, X, \mathcal{P}_i)$. We construct a pMDP $\mathcal{M} := (S, \iota, Act, X, \mathcal{P})$ with

$$S := \bigcup S_i \cup \{s_0\}, \iota := s_0, Act := \{\alpha_i \mid 1 \leq i \leq m\},$$

and $\mathcal{P}$ given by:

$$\mathcal{P}(s, \alpha, s') := \begin{cases} \mathcal{P}_i(s, s') & \text{if } s, s' \in S_i, \alpha = \alpha_i \text{ for some } i, \\ 1 & \text{if } s = s_0, s' = \iota^i, \alpha = \alpha_i \text{ for some } i, \\ 0 & \text{otherwise.} \end{cases}$$

We consider target states $T := \bigcup T_i$. The construction is in polynomial time. The pMDP $\mathcal{M}$ has m strategies $\sigma_1, \ldots, \sigma_m$ with $\sigma_i := \{s_0 \mapsto \alpha_i\}$ (all other states have trivial nondeterminism).
By construction, there exists $\mathsf{val} \in \mathsf{Val}^{\text{wd}}_{\mathcal{M}}$ such that:

$$Pr^{\sigma_i}_{M[\mathsf{val}]}(\Diamond T) < \frac{1}{2} \quad \text{iff} \quad f_i[\mathsf{val}] < 0.$$

Then,

$$\exists \mathsf{val} \in \mathsf{Val}^{\text{wd}}_{\mathcal{M}} : \bigwedge_i Pr^{\sigma_i}_{\mathcal{M}[\mathsf{val}]}(\Diamond T) < \frac{1}{2} \quad \text{iff} \quad \exists \mathsf{val} \in [0,1]^X \bigwedge_i f_i[\mathsf{val}] < 0,$$

or equivalently,

$$\exists \mathsf{val} \in \mathsf{Val}^{\text{wd}}_{\mathcal{M}}\ \forall \sigma \in \Sigma^{\mathcal{M}} : Pr^{\sigma_i}_{\mathcal{M}[\mathsf{val}]}(\Diamond T) < \frac{1}{2} \quad \text{iff} \quad \exists \mathsf{val} \in [0,1]^X \bigwedge_i f_i[\mathsf{val}] < 0. \qquad \blacksquare$$

5.4.2.3 pMCs and pMDPs with a fixed bound on the number of parameters

While the ETR-completeness may be considered bad news, as it renders the problem intractable in general, there is also good news. In particular, for any fixed number of parameters, the (parametric) complexity is lower.

In our considerations, we focus ourselves on graph-preserving instantiations, as the analysis of pMDP $\mathcal{M}$ and $\mathsf{Val}^{\text{wd}}_{\mathcal{M}}$ corresponds to analysing constantly many pMDPs M' on $\mathsf{Val}^{\text{gp}}_{\mathcal{M}'}$, cf. Corollary 5.14 on page 119.

pMCs | First, recall that the standard ETR encoding (e.g., for Theorem 4.12 on page 89) requires $|S| + |X|$ many variables. However, by using the solution function, a formula with only $|X|$ many variables can be constructed (Theorem 4.22). For a fixed number of parameters, the size of the formula is polynomially bounded (Proposition 5.25) and can be computed in polynomial time (Theorem 5.27). Solving ETR formulae is only exponential in the number of variables, i.e., the formulae above can be solved in polynomial time (Proposition 2.4). Together, this yields:

Theorem 5.58 ([HBK17; 20]) For any fixed number K, given a pMC $\mathcal{D}$ with at most K parameters, deciding:

$$\text{is there a } D \in \langle \mathcal{D} \rangle \text{ such that } Pr_D(\Diamond T) \bowtie \lambda?$$

is in P.

That is, in the fixed parameter case, $\exists\textsc{reach}_*^{\bowtie}$ is in P.

Remark 5.5 That reachability in pMCs with, say, four parameters is in P—just like reachability in parameter-free MCs—should not be mistaken as stating that both problems are equally tractable. Algorithms handling ETR require a lot of symbolic computations. We will see in Section 8.1 that merely generating the solution function with four parameters is a challenge.

5

pMDPs | With the positive result for pMCs in place, we turn our attention to pMDPs. However, we can no longer simply eliminate all state variables in the ETR encoding, as also discussed in Section 4.3.4. Observe that a reduction from pMDPs with angelic nondeterminism to pMCs does not work: it requires the introduction of additional parameters (depending on the number of states). Indeed, the precise complexity for the problem remains open. Below, we establish NP-membership for all variants. We also establish coNP-membership for pMDPs with polynomially sized MOSSes. A characterisation of the set of pMDPs with polynomially sized MOSSes is also open. For pMDPs with angelic nondeterminism, NP-membership is straightforward.

Proposition 5.59 In the fixed parameter case, $\exists\exists\textsc{reach}_*^{\bowtie}$ is in NP.

Proof. Guess a memoryless strategy. The strategy can be stored using polynomially many bits[1]. Construct the induced pMC, and verify it in P. ■

For pMDPs with demonic nondeterminism, NP-membership is slightly more involved.

Proposition 5.60 In the fixed parameter case, $\exists\forall\textsc{reach}_*^{\bowtie}$ is in NP.

The following lemma is not strictly necessary, but hints at the solution.

[1] contrary to guessing parameter values, as they are real numbers.

Lemma 5.61 In the fixed parameter case, checking whether a given strategy is somewhere optimal on $\mathsf{Val}^{\mathrm{wd}}_{\mathcal{M}}$ is in P.

Proof. The ETR formula in Constraints 5.2 only has $|X|$ many variables (and is polynomially larger than the pMDP). ■

The essential trick for NP-membership for demonic nondeterminism is guessing an optimal strategy and verifying the induced pMC together with checking that the strategy is indeed optimal.

Proof (Proposition 5.60). We only give the proof for the $\geq$-relation, the other cases are analogous. Observe that

$$\begin{aligned} &\text{exists } \mathsf{val} \in \mathsf{Val}^{\mathrm{wd}}_{\mathcal{M}}, \text{ for all } \sigma \in \mathfrak{S}^{\mathcal{M}} : Pr^{\sigma}_{\mathcal{M}[\mathsf{val}]}(\Diamond T) \geq 1/2 \\ \text{iff } &\text{exists } \mathsf{val} \in \mathsf{Val}^{\mathrm{wd}}_{\mathcal{M}} : \min_{\sigma \in \Sigma^{\mathcal{M}}} Pr^{\sigma}_{M[\mathsf{val}]}(\Diamond T) \geq 1/2, \end{aligned}$$

which means that it is sufficient and necessary for the answer to the problem to be positive that there is a somewhere optimal strategy which (for the valuation for which it is minimal, it) simultaneously induces a reachability probability at least $1/2$. Hence we may guess a somewhere minimal strategy and check its minimality using Constraints 5.2 with a conjunction that the initial state satisfies the threshold. This conjunction only has parameters X, and thus can be checked in P. ■

We neither have any NP-hardness results, nor any evidence that this is unlikely: In particular, a polynomial bound on the size of MOSSes (for a fixed number of variables) would yield strong evidence (see below) that the problem is not NP-hard, but we do not have such a result.

Proposition 5.62 If the size of a MOSS on $\mathsf{Val}^{\mathrm{wd}}_{\mathcal{M}}$ is polynomially bounded for fixed-parameter pMDPs, then $\exists\exists\textsc{reach}^{\bowtie}_{*}$ and $\exists\forall\textsc{reach}^{\bowtie}_{*}$ are in coNP.

The following lemma again hints at the right idea:

Proposition 5.63 In the fixed parameter case, checking whether a given $\Omega \subseteq \Sigma^{\mathcal{M}}$ constitutes an OSS on region R may be done in time polynomial in the size of M, Ω.

Proof. The ETR formula in Proposition 5.39 only has $|X|$ many variables (and is polynomially larger than the pMDP). ■

Proof (Proposition 5.62). Consider the complement of $\exists\forall\textsc{reach}^{\bowtie}_{*}$, that is $\forall\exists\textsc{reach}^{\overline{\bowtie}}_{*}$. It quantifies over all strategies. If a minimal optimal strategy set is only polynomially large, then we guess such a set in polynomial time, verify that it is an OSS (using Thm. 5.63) and then check the induced pMC under each policy (which is in P). ■

5.5 Robust Strategies

We briefly consider obtaining *robust strategies* in pMDPs, rather than (robust) parameter values: We swap the quantification order from the Q_1Q_2REACH problem and thus ask whether some strategy gives guarantees on the maximal or minimal probability of all instantiations of the pMDP eventually reaching T. Formally, for $Q_1, Q_2 \in \{\exists, \forall\}$, let ·robust

$$Q_1Q_2\textsc{robreach}^{\bowtie}_{\text{wd}} \overset{\text{def}}{\iff} Q_1\sigma \in \Sigma^{\mathcal{M}}, Q_2\mathsf{val} \in \mathsf{Val}^{\text{wd}}_{\mathcal{M}} : Pr^{\sigma}_{\mathcal{M}[\mathsf{val}]}(\Diamond T) \bowtie 1/2.$$

We adopt the same conventions as for the Q_1Q_2REACH problem. Variants which use the same quantifier twice, or consider pMCs yield the same results as for REACH$^{\bowtie}$.

Robust strategies have been widely studied in the field of operations research [NG05; WKR13] and control engineering [ZS06; WTM12; DFT16], recently drawn some attention in the verification community [HHHL+17; ABCK+18] and are considered in the scope of *reinforcement learning* [SB98] approaches [MMX12; LXM16; TPCZ18]. It is known that the robust-reachability problem as defined above is not the most general question one can ask. Indeed, we restrict our attention to *memoryless strategies* while, in general, optimal robust strategies require memory and randomisation [ABCK+18]. Moreover, robust reachability is strongly related to risk-sensitive measures [Oso12], which recently gained more attention [KM18; CT18]. Our interest in the robust-reachability problem defined as above is twofold. First, it naturally corresponds to the quantifier-swapped version of the reachability problem. Second, memoryless strategies are desirable for their comprehensibility and ease of implementation.

Proposition 5.64 In the fixed parameter case, $\exists\forall\textsc{robreach}^{</>}_{*}$ are NP-complete. NP-hardness holds even for acyclic pMDPs with a single parameter.

Proof (sketch). Membership in NP is analogous to Proposition 5.59. NP-hardness is based on a reduction from 3SAT, with a construction similar to Figure 5.25. A full proof may be found in [19]. ■

Theorem 5.65 The decision problems $\exists\forall\textsc{robreach}^{\bowtie}_{*}$ are NP-hard and coNP-hard, and in PSPACE. For non-strict inequalities, the problems are coETR-hard.

Proof. NP-hardness follows from Proposition 5.64, coNP/coETR-hardness follows from Theorems 5.49 and 5.51 and Proposition 5.52, respectively. Iterating over all (finitely many) strategies, check each strategy in ETR or in coETR (and thus in PSPACE). ■

Consequently, it is unlikely that either of the problems are in ETR or coETR, as then ETR and coETR would coincide (which is not impossible, but unlikely [SS17]).

We have investigated feasibility in pMDPs. We started with a variety of subclasses of pMDPs and their valuations, which reduce technicalities and are motivated by practical considerations. For simple pMCs, we investigated the solution function and its relation to general polynomials. For pMDPs, we considered an adaption of optimal strategies. These are ingredients for the complexity results presented near the end of this chapter. First and foremost, even feasibility in simple acyclic pMCs (with non-strict inequalities) is ETR-complete. Consequently, assuming simplicity does not make the problem more tractable. We finished the section with some implications regarding robust strategies.

6 Markov Chain Families

Modelling discrete choices

Summary | We consider the model of family MCs (fMCs). Like their parametric counterparts, they are best thought of as representing a set of MCs. However, the focus for fMCs is on concisely representing finite sets of MCs with varying topology. We formally introduce the model, show their relation with pMCs with pure valuations, and formulate the most prominent problem statements on fMCs. We conclude the section by considering how to model fMCs using sketches. Sketches are an extension to probabilistic programs describing MCs. They allow to concisely represent a set of programs and are thus a natural choice to model fMCs.

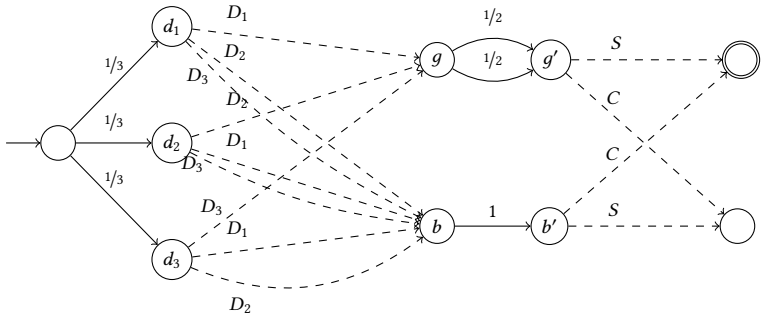
Origins | This chapter is novel in its presentation, but built around an adaption of the notion of families, introduced in [15], and probabilistic program sketches, as in [14].

Background | This chapter considers the analysis of (finite) sets of Markov chains, and consequently requires knowledge about MCs and their analysis, as introduced in Chapter 3. The second part of this chapter lifts the notion to the analysis of finite sets of probabilistic programs (describing finite MCs) and requires a marginal understanding of such programs as summarised in Section 3.5. Various ideas, notions and problems coincide with pMCs, therefore, an understanding of pMCs (Sections 4.1 and 4.2) and their subclasses (Section 5.1.1) is helpful. Except for the section making the connection to pMCs explicit, no knowledge of pMCs is required, but for conciseness, we keep the explanation of some shared notions very brief.

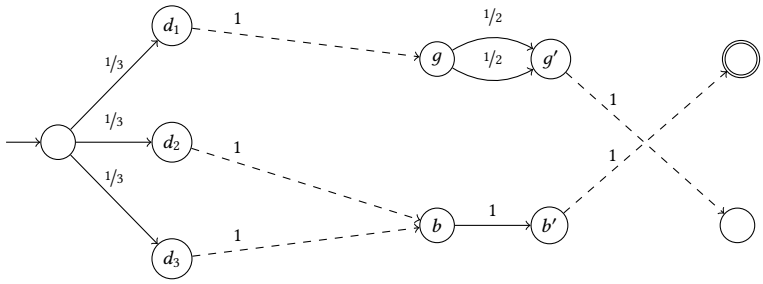
6.1 Family Markov Chains

In this section, we consider sets of MCs with a varying topology. Such models occur when aggregating various alternative designs. Before we formalise such families, we give two typical examples.

Example 6.1 We consider the development of a communication system. For that, we can buy two different types of antennas, and configure both the number of re-



(a) The Monty-Hall MDP



(b) A Monty-Hall MC for selecting door D_1 and changing doors after revealing a goat.

Figure 6.1: The Monty-Hall MDP and an MC for a concrete strategy.

> transmissions of each packet and the time between retransmissions. Consequently, large parts of the protocol description are shared among all these instances, but the instances vary. Importantly, a different number of retransmissions clearly leads to a different topology of the underlying MC. We might now be interested in either proving that all protocol combinations yield satisfactory performance, which is interesting if all products are to be released to the market, or finding a product with suitable performance, if we want to buy either of the products, or if we are designing such a system.

An example as above is typical for the development of *software product lines*. We discuss them a bit further in Section 7.3. Uncertain topologies may also be considered as underlying model when making decisions under restrictions, i.e., when considering MDPs where (memoryless deterministic) strategies have to make consistent choices in multiple states.

Example 6.2 Consider the famous Monty-hall problem[a], which traces back to a probability paradox in [Ber89]. We include only a brief explanation. Alice plays the finale of a game show. She may open one of three doors. Opening one of the doors means she wins a car, opening the other doors provide her with a goat. Alice wants to win the car. We denote this door as the good door, the other doors are bad doors. A-priori, all doors have the same probability of being good, i.e., each door is good with probability $1/3$. Alice selects a door d. Now the host, Bob, opens randomly (and fair) one bad door different from d. Such a door always exists. Alice now has the opportunity to change doors. Indeed, if she does, she doubles her winning chances.
In Figure 6.1a, we show an MDP (with dashed arrows to denote nondeterministic choices). The MDP concisely models the problem. In a first transition, the car and goats are distributed. The good door is thus d_i. Then, Alice picks a door D_i. Due to symmetry, our states only store whether she picks a good door (g) or not (b). After Bob opens one door, we are in g', b', respectively. Now Alice may change doors (C) or stick to her choice (S). She wins the car if she sticks after picking the good door, and changes after picking a bad door.
While the MDP is an adequate transition model, not all strategies in the MDP are realisable in the real world, as Alice *does not know behind which door the car is located.* Thus, if she selects a door D_1 in state d_1, she would also select D_1 in d_2. However, we can consider the finitely many induced Markov chains that adhere to these restrictions. An example is the MC in Figure 6.1b, which reflects selecting door D_1 and changing doors at the end. There are six MCs that reflect the (memoryless, deterministic) strategies.

[a] https://en.wikipedia.org/wiki/Monty_Hall_problem

Partially observable MDPs (POMDPs) are a model in which, like above, we are interested in strategies mapping (sequences of) state labels (so called observations) to actions, rather than general strategies that map (sequences of) states to actions. We discuss POMDPs and their relation to pMCs (and by transitivity, their relation to fMCs) in Section 7.1. For now, the important message is that MDP strategies are often subject to restrictions and that such restricted strategies may be analysed by considering the set of induced MCs.

Related work | Besides software product lines and various types of POMDPs discussed in Chapter 7, alternative designs are prominently modelled as modal transition systems [LT88; AHLN+08; Kre17]. In particular, *parametric* modal transition systems [BKLM+15] and synthesis therein [BKLM+12] allow for similar dependencies that occur in program-level sketches. Probabilistic extensions are considered in, e.g. [DKLL+13], but not in conjunction with synthesis.

6.1.1 Dedicated model for finite sets of MCs

We introduce a dedicated formalism to capture MCs with an uncertain topology.

fMC

Definition 6.1 (family MC) A *family MC* (*fMC*) is a tuple $\mathfrak{D} = (S, \iota, Y, \mathsf{Dom}_Y, \mathbf{P}, \mathsf{rew})$ with a finite set of states $S := S_1 \times S_2$ where each state is a tuple $\langle s_1, s_2 \rangle$, an initial state ι, a finite set of parameters Y, where the domain for each parameter is given by $\mathsf{Dom}_Y \colon Y \to 2^{S_1}$, a parameterised transition probability function $\mathbf{P} \colon S \to \mathit{Distr}\big((Y \cup S_1) \times S_2\big)$, and rewards $\mathsf{rew} \colon S \to \mathbb{R}_{\geq 0}$.

As before, we omit the rewards if $\mathsf{rew}(s) := 0$ for all $s \in S$. The notation of these fMCs is affected by simplifications in Remarks 6.2 and 6.3 on the next page and on page 164. Family MDPs are an interesting object for future work.

Remark 6.1 The tuple structure of states may look peculiar on first sight, but corresponds to the semantics of a probabilistic program, as we argue in Section 6.2.2. In particular, states are then tuples that encode program-variable valuations.

The transition probability function of MCs maps states to distributions over states. In fMCs, this function maps states to distributions over states, with their first component parameterised. Assigning every parameter a concrete state yields an MC.

assignment

Definition 6.2 (Assignment) An *assignment* of an fMC $\mathfrak{D} = (S_1 \times S_2, \iota, Y, \mathsf{Dom}_Y, \mathbf{P})$ is a function $\nu \colon Y \to S_1$ where for all $y \in Y$ it holds that $\nu(y) \in \mathsf{Dom}_Y(y)$. The set $\mathcal{F}_{\mathfrak{D}}$ contains all assignments for $\mathfrak{D}$.

instantiation

Definition 6.3 (Instantiation) Let $\mathfrak{D} = (S := S_1 \times S_2, \iota, Y, \mathsf{Dom}_Y, \mathbf{P}, \mathsf{rew})$ be an fMC and ν an assignment for $\mathfrak{D}$. The *instantiation* $\mathfrak{D}[\nu]$ of $\mathfrak{D}$ and ν is an MC with $\mathfrak{D}[\nu] := (S, \iota, P, \mathsf{rew})$, where for all $s, s' \in S$, with $s' := \langle s_1', s_2' \rangle$ and

$$P(s, s') := \mathbf{P}[\nu](s, s') := \sum_{\substack{y \in Y \\ \nu(y) = s_1'}} \mathbf{P}(s)(\langle y, s_2' \rangle) + \mathbf{P}(s)(\langle s_1', s_2' \rangle).$$

family

We refer to sets of assignments as (sub)*families*. An fMC and a family may be seen as a generator for the set of all instantiations:

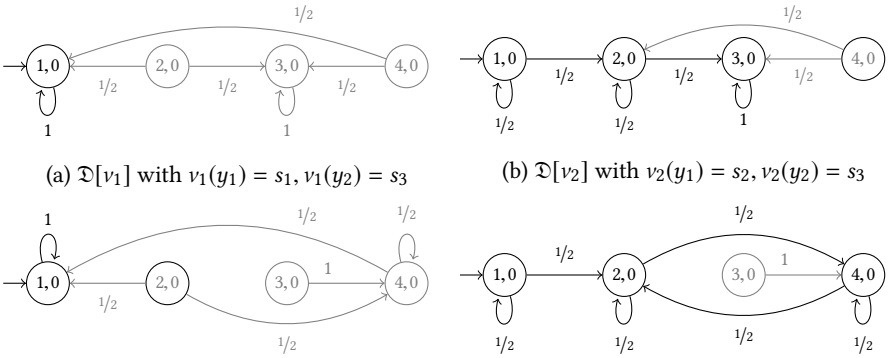
generator

Definition 6.4 (Generator for fMCs) The *generator* $\langle \mathfrak{D} \rangle$ for fMC $\mathfrak{D}$ and $F \subseteq \mathcal{F}_{\mathfrak{D}}$ is:

$$\langle \mathfrak{D} \mid F \rangle := \{\mathfrak{D}[\nu] \mid \nu \in F\}.$$

We write $\langle \mathfrak{D} \rangle$ to denote $\langle \mathfrak{D} \mid \mathcal{F}_{\mathfrak{D}} \rangle$.

The set $\mathcal{F}_{\mathfrak{D}}$ is finite, but exponential in $|Y|$.



(a) $\mathfrak{D}[v_1]$ with $v_1(y_1) = s_1, v_1(y_2) = s_3$

(b) $\mathfrak{D}[v_2]$ with $v_2(y_1) = s_2, v_2(y_2) = s_3$

(c) $\mathfrak{D}[v_3]$ with $v_3(y_1) = s_1, v_3(y_2) = s_4$

(d) $\mathfrak{D}[v_4]$ with $v_4(y_1) = s_2, v_4(y_2) = s_4$

Figure 6.2: Family for the fMC from Example 6.3. States $\langle s_i, s_j \rangle$ are labelled as i,j.

Example 6.3 Consider a family of MCs $\mathfrak{D} = (S_1 \times S_2, \iota, Y, \mathsf{Dom}_Y, \mathbf{P})$ where

$$S_1 := \{s_1, s_2, s_3, s_4\} \text{ and } S_2 := \{s_0\},$$

$\iota := \langle s_1, s_0 \rangle$, and $Y := \{y_1, y_2\}$ with

$$\mathsf{Dom}_Y(y_1) := \{s_1, s_2\}, \mathsf{Dom}_Y(y_2) := \{s_3, s_4\}.$$

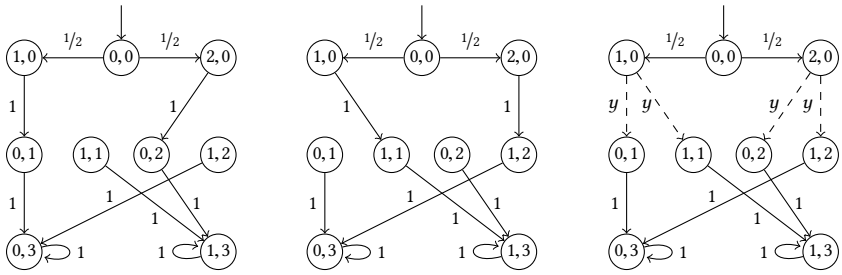
The parametric transition function $\mathbf{P}$ is defined by:

$$\begin{aligned}
\mathbf{P}(\langle s_1, s_0 \rangle) &:= 1/2\colon \langle s_1, s_0 \rangle + 1/2\colon \langle y_1, s_0 \rangle,\\
\mathbf{P}(\langle s_2, s_0 \rangle) &:= 1/2\colon \langle y_1, s_0 \rangle + 1/2\colon \langle y_2, s_0 \rangle,\\
\mathbf{P}(\langle s_3, s_0 \rangle) &:= 1\colon \langle y_2, s_0 \rangle,\\
\mathbf{P}(\langle s_4, s_0 \rangle) &:= 1/2\colon \langle y_1, s_0 \rangle + 1/2\colon \langle y_2, s_0 \rangle.
\end{aligned}$$

We use the distribution syntax from Appendix A. Figure 6.2 shows the four MCs that result from the assignments $\{v_1, v_2, v_3, v_4\}$. States that are unreachable from the initial state are greyed out.

We make two remarks about notation.

Remark 6.2 To simplify the formal notation, we may restrict the signature of the transition function to $\mathbf{P}\colon S \to \mathit{Distr}(Y \times S_2)$. Therefore, we need a parameter for every state $s_1 \in S_1$. Thus, the more verbose signature given in Definition 6.1 makes the notation of fMCs more concise.



(a) MC $\mathfrak{D}_2[\{y \mapsto s_0\}]$ (b) MC $\mathfrak{D}_2[\{y \mapsto s_1\}]$ (c) fMC $\mathfrak{D}_2$

Figure 6.3: Instantiations of the fMC from Example 6.4. States $\langle s_i, s_j \rangle$ are labelled as i,j.

Remark 6.3 The fMCs in [14; 15] correspond to fMCs with $|S_2| = 1$, as does the example above. Their main rationale is their simplicity. Observe that whenever S_2 is trivial, we may omit explicitly denoting all tuples. We illustrate this based on the example from above. We write s_i to denote $\langle s_i, s_j \rangle$ and assume $\mathbf{P}$ maps $S_1 \to \mathit{Distr}(Y)$:

$$\mathbf{P}(s_1) := 1/2 \colon y_0 + 1/2 \colon y_1, \qquad \mathbf{P}(s_2) := 1/2 \colon y_1 + 1/2 \colon y_2,$$
$$\mathbf{P}(s_3) := 1 \colon y_2, \qquad \mathbf{P}(s_4) := 1/2 \colon y_1 + 1/2 \colon y_2.$$

In Section 6.1.4, we show that the class with $|S_2| = 1$ is not expressive enough for most practical purposes: Indeed, because all parametric transitions lead to a single state, taking such a transition amounts to forgetting everything but this transition, which severely limits the applicability. Yet, most notions may be explained on this more concise notion of families. The next example shows a family with $|S_2| > 1$.

Example 6.4 Consider the fMC $\mathfrak{D} = (S_1 \times S_2, \iota, Y, \mathsf{Dom}_Y, \mathbf{P})$ where

$$S_1 := \{s_0, s_1, s_2\}, S_2 := \{t_0, t_1, t_2, t_3\}, \iota := \langle s_0, t_0 \rangle, \text{and}$$
$$Y := \{y\} \text{ with } \mathsf{Dom}_Y(y) := \{s_0, s_1\}.$$

The parametric transition function $\mathbf{P}$ is defined by:

$$\mathbf{P}(\langle s_0, t_0 \rangle) := 1/2 \colon \langle s_1, t_0 \rangle + 1/2 \colon \langle s_2, t_0 \rangle,$$
$$\mathbf{P}(\langle s_1, t_0 \rangle) := 1 \colon \langle y, t_1 \rangle, \qquad \mathbf{P}(\langle s_2, t_0 \rangle) := 1 \colon \langle y, t_2 \rangle,$$
$$\mathbf{P}(\langle s_0, t_1 \rangle) := 1 \colon \langle s_0, t_3 \rangle, \qquad \mathbf{P}(\langle s_1, t_1 \rangle) := 1 \colon \langle s_1, t_3 \rangle,$$
$$\mathbf{P}(\langle s_0, t_2 \rangle) := 1 \colon \langle s_1, t_3 \rangle, \qquad \mathbf{P}(\langle s_1, t_2 \rangle) := 1 \colon \langle s_0, t_3 \rangle,$$

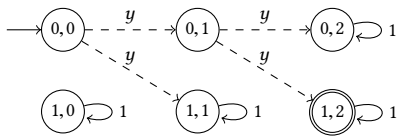


Figure 6.4: fMC with a reachable state that is not reachable in any instantiation.

and for all other pairs $\langle s, t\rangle \in S_1 \times S_2$: $\mathbf{P}(\langle s, t\rangle) \coloneqq 1\colon \langle s, t\rangle$. Figure 6.3 shows the two MCs corresponding to $\mathcal{F}_{\mathfrak{D}}$ and a graphical representation of the fMC.

We may depict the latter fMC concisely, as it is a *simplex* fMC. The following notion is helpful, e.g., when defining simplex fMCs: ·simplex

Definition 6.5 (Occurs) The set $\mathsf{occurs}(s)$ of parameters occurring in state s is:

$$\mathsf{occurs}(s) \coloneqq \{y \in Y \mid \text{exists } s_2 \in S_2 \text{ s.t. } \langle y, s_2\rangle \in \mathit{supp}(\mathbf{P}(s))\}.$$

Definition 6.6 (Simplex fMCs) An fMC is called *simplex*, if for all $s \in S$ either:
- there is one outgoing (parametric) transition, i.e. $|\mathit{supp}(\mathbf{P}(s))| = 1$, or
- no parameters occur, i.e., $\mathsf{occurs}(s) = \emptyset$.

Let us clarify the graphical representation of a simplex fMC as in Figure 6.3c. Transitions without occurring parameters are depicted with a solid line and labelled with their probability, as usual. Any transition from state s with a parameter is indicated by dashed lines labelled with the parameter and a state $s' \in S_1$. The lines lead to any state $\langle s_1', s_2'\rangle$ that are reached by substituting the parameter with s_1'.

Focussing on simplex fMCs is not a restriction.

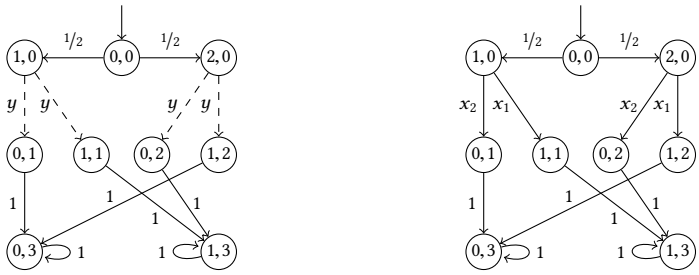
Lemma 6.1 For every fMC $\mathfrak{D}$ there exists a simplex fMC $\mathfrak{D}'$ such that $\langle\mathfrak{D}\rangle \simeq \langle\mathfrak{D}'\rangle$.

The construction introduces at most $|S|$ auxiliary states, analogous to the construction of alternating (p)MDPs in Definition 5.10 on page 113, and runs in linear time. In that correspondence, parametric transitions in the fMC match nondeterminism in the MDP.

Finally, graph notions are trivial to adapt and helpful in the remainder of this thesis.

Definition 6.7 (Path) A *path* of an fMC $\mathfrak{D} \coloneqq (S_1 \times S_2, \iota, Y, \mathsf{Dom}_Y, \mathbf{P}, \mathsf{rew})$ is an (in)finite sequence $\pi = s_0 \xrightarrow{a} s_1 \xrightarrow{\alpha_1} \cdots$, where $s_i \in S$, and there exists $\nu \in \mathcal{F}_{\mathfrak{D}}$ such that $\mathbf{P}[\nu](s_i, s_{i+1}) \neq 0$ for all $i \in \mathbb{N}$. ·path

We stress that a path to an fMC, in general, does not reflect a path in any instantiation. Similarly, a path from the initial state of $\mathfrak{D}$ to some state s does not mean that s is reachable in any instantiation of $\mathfrak{D}$.



(a) simplex fMC (b) simplex pMC

Figure 6.5: a simplex fMC and its corresponding simplex pMC

Example 6.5 Consider the fMC in Figure 6.4. Although there is a path from the initial state to $\langle s_1, t_2 \rangle$, it is unreachable in any instantiation. The only incoming transition requires an assignment $\{y \mapsto s_1\}$ in $\langle s_0, t_1 \rangle$, but to reach $\langle s_0, t_1 \rangle$, the assignment has to be $\{y \mapsto s_0\}$.

We also lift the set of successors in an fMC, interpreting this set as *potential* successors[1]

$$\mathsf{succ}(s) \coloneqq \{s' \in S \mid \text{exists } \nu \in \mathcal{F}_{\mathfrak{D}} \text{ s.t. } \mathbf{P}[\nu](s, s') \neq 0\}.$$

As before, we write $\mathsf{succ}(S') \coloneqq \bigcup_{s \in S'} \mathsf{succ}(s)$.

6.1.2 Relation to pMCs

There is a strong connection between simplex fMCs and simplex pMCs. Indeed, the following example illustrates that constructing a pMC from an fMC is straightforward.

·to pMC

Example 6.6 Recall the fMC in Example 6.4, also depicted in Figure 6.5a. The simplex pMC in Figure 6.5b has the same two instantiations from pure valuations. In particular, we introduce a parameter group for each parameter in the fMC, and the group consists of a parameter for each value in the parameter domain.

We formalise the construction from the example:

Proposition 6.2 Let $\mathfrak{D}$ be a simplex fMC. We can (in linear time) construct a simplex pMC $\mathcal{D}$ such that

$$\langle \mathfrak{D} \rangle = \langle \mathcal{D} \mid \text{pure} \rangle.$$

[1] This definition is in line with the definition for pMCs.

Proof. Let $\mathfrak{D} = (S_1 \times S_2, \iota, Y, \mathsf{Dom}_Y, \mathbf{P}, \mathsf{rew})$ be a simplex fMC. We construct an expanded simplex pMC $\mathcal{D} := (S, \iota, X, \mathcal{P}, \mathsf{rew})$ with $S := S_1 \times S_2$, parameters $X := Y \times S_1$ with groups $\{y \in Y\} \times S_1$, and

$$\mathcal{P}(s, \langle s_1', s_2' \rangle) := \begin{cases} \mathbf{P}(s)(\langle s_1', s_2' \rangle) & \text{if } \mathsf{occurs}(s) = \emptyset, \\ \langle y, s_1' \rangle & \text{if } \mathit{supp}(\mathbf{P}(s)) = \{\langle y, s_2' \rangle\}, \\ 0 & \text{otherwise.} \end{cases}$$

Thus, the parameters x_1 and x_2 in Figure 6.5b would—in this construction—be denoted $\langle y, s_1 \rangle$ and $\langle y, s_0 \rangle$, respectively. The transformation clearly runs in linear time (when only creating the parameters that are used in the transitions). With this translation, we obtain $\langle \mathfrak{D} \rangle = \langle \mathcal{D} \mid \text{pure} \rangle$ as argued below:

- For any assignment $\nu \in \mathcal{F}_{\mathfrak{D}}$, we construct $\mathsf{val} \in \mathsf{Val}_{\mathcal{D}}^{\text{pure}}$ such that $\mathcal{D}[\mathsf{val}] \simeq \mathfrak{D}[\nu]$: We set

  $$\mathsf{val}(\langle y, s_1 \rangle) := \begin{cases} 1 & \text{if } \nu(y) = s_1, \\ 0 & \text{otherwise.} \end{cases}$$

  Clearly, $\mathfrak{D}[\nu] = \mathcal{D}[\mathsf{val}]$.

- For any valuation $\mathsf{val} \in \mathsf{Val}_{\mathcal{D}}^{\text{pure}}$, we construct $\nu \in \mathcal{F}_{\mathfrak{D}}$ such that $\mathcal{D}[\mathsf{val}] \simeq \mathfrak{D}[\nu]$: We uniquely set $\nu(y) := s_1$ if $\mathsf{val}(\langle y, s_1 \rangle) = 1$ for some $y \in Y$, $s_1 \in \mathsf{Dom}_Y(y)$. As val is well-defined, $\mathsf{val}(\langle y, s_1' \rangle) = 0$ for all $s_1' \in \mathsf{Dom}_Y(y) \setminus \{s_1\}$. Clearly, $\mathfrak{D}[\nu] = \mathcal{D}[\mathsf{val}]$. ■

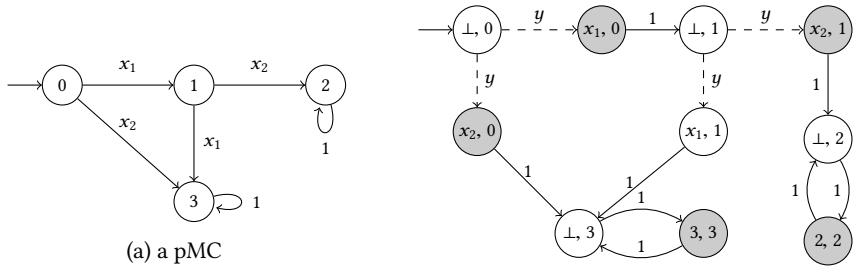
The other direction, that is, transforming (expanded) simplex pMCs (for pure valuations) into fMCs is also possible. Intuitively, the family parameters are encoding to which of the probability parameters the probability mass should be attached. The formal construction is slightly technical, the main reason for the technicality is that pMCs do not impose the state-tuple structure as required for fMCs. We first give an example. ·from pMC

6

Example 6.7 Consider the pMC in Figure 6.6a. We copy the pMC into the fMC in Figure 6.6b and introduce an auxiliary state for every outgoing transition. The only interesting transitions, however, are the parameterised transitions. With the introduction of an auxiliary state, we first take a parameterised transition that stores according to which (pMC) parameter we move, and then use the second transition to move accordingly.

We formalise the idea from the example[1]:

[1] auxiliary states after non-parameterised transitions ease the presentation of the construction.



(a) a pMC

(b) an fMC

Figure 6.6: a simplex pMC and its corresponding simplex fMC

Proposition 6.3 Let $\mathcal{D}$ be a simplex pMC with constant rewards. We may (in quadratic time) construct a simplex fMC $\mathfrak{D}$ such that

$$\langle \mathfrak{D} \rangle \simeq \langle \mathcal{D} \mid \text{pure} \rangle.$$

Proof. Let $\mathcal{D} = (S, \iota, X, \mathcal{P}, \mathsf{rew})$ be an expanded simplex pMC with parameter groups $X_1, \ldots, X_m$. We use $\mathsf{group}\colon S \to \{0, \ldots, m\}$ with

$$\mathsf{group}(s) := \begin{cases} i & \text{if } \mathsf{occurs}(s) = X_i, \\ 0 & \text{otherwise.} \end{cases}$$

We construct an fMC $\mathfrak{D}$. Intuitively, the fMC alternates between original states s, which are reflected by $\langle \bot, s \rangle$, and auxiliary states. There are two types of auxiliary states:

- If s is a parametric state in the pMC, we have a parametric transition to any $\langle x, s \rangle$ state, with parameter $x \in \mathsf{occurs}(s)$, and
- otherwise, we move to a state $\langle s, s' \rangle$ with the (concrete) probability $\mathcal{P}(s, s')$.

From either of those states, we move back to a $\langle \bot, s'' \rangle$ state.
Formally, we construct the fMC

$$\mathfrak{D} := \big((X \cup S \cup \{\bot\}) \times S, \iota', Y, \mathsf{Dom}_Y, \mathbf{P}, \mathsf{rew}'\big)$$

with $\iota' := \langle \bot, \iota \rangle$, $Y := \{y_1, \ldots, y_m\}$, Dom_Y defined by $\mathsf{Dom}_Y(y_i) := X_i$. We create three groups of transitions, using $s, s' \in S$, $x \in X$:

$$\mathbf{P}(\langle \bot, s \rangle) := \begin{cases} \{\langle y_i, s \rangle \mapsto 1\} & \text{if } i = \mathsf{group}(s) \neq 0, \\ \{\langle s, s' \rangle \mapsto \mathcal{P}(s, s') \mid s' \in S\} & \text{otherwise.} \end{cases}$$

$$\mathbf{P}(\langle s, s' \rangle) := \{\langle \bot, s' \rangle \mapsto 1\}$$

$$\mathbf{P}(\langle x, s' \rangle) := \{\langle \bot, s \rangle \mapsto 1\} \text{ with } s \text{ the unique state s.t. } \mathcal{P}(s', s) = x,$$

and rew defined by

$$\mathsf{rew}'(\langle s_1, s_2\rangle) := \begin{cases} \mathsf{rew}(s_2) & \text{if } s_1 = \bot, \\ 0 & \text{otherwise.} \end{cases}$$

The number of states is at most quadratic. The correctness follows from the following two constructions:

- For any assignment $\nu \in \mathcal{F}_{\mathfrak{D}}$, we let $\mathsf{val} \in \mathsf{Val}^{\text{pure}}_{\mathcal{D}}$ such that $\mathcal{D}[\mathsf{val}] \simeq \mathfrak{D}[\nu]$: We set

  $$\mathsf{val}(x) := \begin{cases} 1 & \text{if } \nu(y_i) = x, \text{ for some } x \in X_i, 1 \leq i \leq m \\ 0 & \text{otherwise.} \end{cases}$$

  Then $\mathfrak{D}[\nu] \simeq \mathcal{D}[\mathsf{val}]$, as $Pr_{\mathfrak{D}[\nu]}(\langle \bot, s\rangle \models \Diamond\{\langle \bot, s'\rangle\}) = \mathcal{P}(s, s')[\mathsf{val}]$ for any $s, s' \in S$.

- For any valuation $\mathsf{val} \in \mathsf{Val}^{\text{pure}}_{\mathcal{D}}$, we let $\nu \in \mathcal{F}_{\mathfrak{D}}$ such that $\mathcal{D}[\mathsf{val}] \simeq \mathfrak{D}[\nu]$. Observe that for any pure valuation, for each parameter group X_i there is exactly one $x \in X_i$ s.t. $\mathsf{val}(x) = 1$. We uniquely set $\nu(y_i) := x$ if $\mathsf{val}(x) = 1$.

  Then $\mathfrak{D}[\nu] \simeq \mathcal{D}[\mathsf{val}]$, as $Pr_{\mathfrak{D}[\nu]}(\langle \bot, s\rangle \models \Diamond\{\langle \bot, s'\rangle\}) = \mathcal{P}(s, s')[\mathsf{val}]$ for any $s, s' \in S$. ■

Remark 6.4 We observe a gap in adequate modelling support. pMCs are typically used for considering (uncountable) parameter ranges which are graph-preserving, e.g., parameter x between $1/4$ and $1/3$. fMCs target discrete choices, thus either x is $1/4$ or $1/3$, but the methods are mostly targeted towards scenarios where the graph is *not* preserved. While fMCs have no problem in modelling that the probability is either $1/4$ or $1/3$ (using auxiliary states), we do not consider such scenarios in detail as fMCs feel too liberal to capture or exploit this relevant graph-preserving aspect.

6.1.3 Problem statements

We consider specifications Φ, i.e., finite sets of properties φ as in Section 3.2.2. Recall that $D \models \Phi$ iff $D \models \varphi$ for all $\varphi \in \Phi$. We are interested in the assignments—and the corresponding instantiations—that ensure a given specification is satisfied.

Remark 6.5 This subsection deliberately strongly resembles Section 4.2. Readers that have studied pMCs before will find no surprises. Therefore, we also refrain from giving motivating examples. The differences are as follows: We allow specifications rather than single properties, as the methods for only finitely many MCs (and no nondeterminism) are a bit more powerful. Consequently, the problem definition for *maximum restricted probability* is not mirroring any problem statement for pMCs. We do not introduce solution functions as we are not aware of any structural results about such functions.

·accepting
·rejecting

Definition 6.8 Let $\mathfrak{D}$ be an fMC and Φ a specification: An assignment ν is *accepting* Φ, if $\mathfrak{D}[\nu] \models \Phi$, and *rejecting* Φ, if $\mathfrak{D}[\nu] \not\models \Phi$. The sets $\mathsf{acc}_\Phi^\mathfrak{D}$ and $\mathsf{rej}_\Phi^\mathfrak{D}$ denote all accepting and rejecting assignments, respectively.

Clearly, it holds that $\mathsf{acc}_\Phi^\mathfrak{D} \cup \mathsf{rej}_\Phi^\mathfrak{D} = \mathcal{F}_\mathfrak{D}$. For singleton specifications $\Phi = \{\varphi\}$, we may omit the set-braces, e.g. use $\mathsf{acc}_\varphi^\mathfrak{D}$ to denote $\mathsf{acc}_{\{\varphi\}}^\mathfrak{D}$.

Lemma 6.4 For an fMC and a specification Φ, it holds that

$$\mathsf{acc}_\Phi^\mathfrak{D} = \bigcap_{\varphi \in \Phi} \mathsf{acc}_\varphi^\mathfrak{D} \quad \text{and} \quad \mathsf{rej}_\Phi^\mathfrak{D} = \bigcup_{\varphi \in \Phi} \mathsf{rej}_\varphi^\mathfrak{D}.$$

With the definition of accepting and rejecting, we are ready to formulate the problems that we want to (algorithmically) solve. Some essential problems that we address are: Given an fMC $\mathfrak{D}$, a family F, and a specification Φ:

- (**feasibility**)
  does there exist an MC $D \in \langle \mathfrak{D} \mid F \rangle$ s.t. $D \models \Phi$, or equivalently, is $\mathsf{acc}_\Phi^\mathfrak{D} \cap F \neq \emptyset$?
- (**validity**)
  for all MCs $D \in \langle \mathfrak{D} \mid F \rangle$, does $D \models \Phi$ hold, or equivalently, is $F \subseteq \mathsf{acc}_\Phi^\mathfrak{D}$?

Feasibility and validity problems are dual. Validity problems may be considered verification problems, in which we want to certify that certain behaviour is unlikely irrespectively of precise topology. Consequently, we often refer to validity as *family verification*. Feasibility problems are typical instances of a synthesis problem and we are mostly interested in obtaining an instantiation accepting Φ.

family verification

optimal feasibility

We extend the feasibility problems to *optimal feasibility*, as we are interested in not just any, but preferably a good or even optimal instantiation. For example, we want to optimise the reachability probability: Given an fMC $\mathfrak{D}$, a family F, and a target set T,

- (**maximum probability**)
  find $\max_{\nu \in F} Pr_{\mathfrak{D}[\nu]}(\Diamond T)$,

or the analogue formulation with expected rewards, and/or minimisation. Typically, we are also interested in finding this maximising assignment. Variants of the problem that search for an almost optimal assignment are a straightforward extension.

We may put additional constraints on the assignments. These constraints may be formulated over the assignments themselves: Given an $\mathfrak{D}$, a family F and property φ, and additionally a *parameter-cost function*

parameter-cost function

$$\mathsf{cost}_\mathfrak{D} \colon \mathcal{F}_\mathfrak{D} \to \mathbb{R}_{\geq 0}$$

and a budget κ. The constraints may also be formulated by additional properties. Together, that yields: Given an fMC $\mathfrak{D}$, a family F, a target set T, a specification Φ, and a budget κ:

- (**maximum restricted probability**)
  find $\max_{\nu \in F} Pr_{\mathfrak{D}[\nu]}(\Diamond T)$ such that $\nu \in \mathsf{acc}_{\Phi}^{\mathfrak{D}}$ and $\mathsf{cost}_{\mathfrak{D}}(\nu) \leq \kappa$.

Ultimately, it may be interesting to analyse all instantiations and see which satisfy the specification and which do not. This idea is captured by the *partitioning* (aka: threshold) problem: given an fMC $\mathfrak{D}$, a family F and a specification Φ, partitioning

- (**complete partitioning**)
  find $F \cap \mathsf{acc}_{\Phi}^{\mathfrak{D}}$ and $F \setminus \mathsf{acc}_{\Phi}^{\mathfrak{D}}$ (and represent them concisely).

Just as for pMDPs, the list above is not meant to be complete.

6.1.4 Aspects of family MCs

We finish the treatment of fMCs with some theoretical aspects. In particular, we discuss that the extension proposed in this thesis is more powerful than the original notion of an fMC and we present some complexity results.

Expressivity of fMCs with $|S_2| = 1$ | Recall from Example 6.5 that a path in the fMC may not be reflected by a path in any of its instantiations. This statement also holds for some paths in an fMC with $|S_2| = 1$. However, if $|S_2| = 1$, then every state s with a path in the fMC from the initial state to s is reachable in at least one instantiation.

Proposition 6.5 Let $\mathfrak{D}$ be an fMC with $|S_2| = 1$ and state $s \in S$. If there is a path in $\mathfrak{D}$ from ι to s, then there is an MC $D \in \langle \mathfrak{D} \rangle$ such that $s \in \mathsf{reach}_D$.

Proof. Let π be a path in $\mathfrak{D}$ from ι to $s_n := s$. First, assume that along the path each parameter occurs at most once, i.e., after suitable renaming,

$$\pi := \iota y_0 s_1 y_1 s_2 \ldots y_{i-1} s_n \text{ with } y_i \neq y_j \text{ for all } i \neq j.$$

However, the assignment $\nu := \{y_i \mapsto s_{i+1}\}$ yields a path from ι to s_n in $\mathfrak{D}[\nu]$.

Let us now assume that some parameter y^* occurs multiple times, the first time at s_l and the last time at s_m, i.e.,

$$\pi' := \iota y_0 s_1 y_1 s_2 \ldots s_l y^* s_{l+1} \ldots s_m y^* s_{m+1} y_{m+1} \ldots s_n,$$

with $y_i \neq y^*$ for all $i < l$ and for all $i > m$. But then, there is a path

$$\hat{\pi} := \iota y_0 s_1 y_1 \ldots s_l y^* s_{m+1} y_{m+1} \ldots s_{n-1} y_{n-1} s_n$$

where parameter y^* occurs only once. By an inductive argument, there is always a path π as above, where each parameter occurs at most once. Consequently, for MC $D := \mathfrak{D}[\nu]$ it holds that $s \in \mathsf{reach}_D$. ■

Likewise, "mutual exclusion" as in Example 6.4 is impossible with $|S_2| = 1$.

Lemma 6.6 There is no fMC $\mathfrak{D}$ with $|S_2| = 1$ and states s, s', t such that

- for all $D \in \langle \mathfrak{D} \rangle$, $t \notin \mathsf{reach}_D(s) \cap \mathsf{reach}_D(s')$.
- exists a $D \in \langle \mathfrak{D} \rangle$ such that $t \in \mathsf{reach}_D(s) \setminus \mathsf{reach}_D(s')$,
- exists a $D \in \langle \mathfrak{D} \rangle$ such that $t \in \mathsf{reach}_D(s') \setminus \mathsf{reach}_D(s)$.

For conciseness we leave out this technical proof. The essential idea is that there must be the same parameter on paths from both states, and after taking this parameter, we are always in the same state.

These result show that fMCs with $|S_2| = 1$ are a restricted class of fMCs. These restrictions are also reflected by the complexity results below. Nevertheless, their concise representation is beneficial for presenting various ideas.

Complexity | Contrary to pMCs, the discrete nature of fMCs yields NP upper bounds for the complexity.

Proposition 6.7 Given an fMC $\mathfrak{D}$ and, deciding whether there is an MC $D \in \langle \mathfrak{D} \rangle$ such that $Pr_D(\Diamond T) \geq \frac{1}{2}$ is in NP.

Proof. We simply guess the assignment in polynomial time, obtain an MC, and apply standard model checking, again in polynomial time. ■

The hardness results carry over from the results for qualitative model checking on pMCs under pure valuations in Proposition 5.42 on page 142.

Theorem 6.8 Given an fMC $\mathfrak{D}$, deciding whether there is an MC $D \in \langle \mathfrak{D} \rangle$ such that $Pr_D(\Diamond T) > 0$ is NP-hard.

Likewise, the other qualitative problem statements (cf. Section 5.4.1) pose also NP-hard problems. Following Proposition 6.5, deciding (positive) reachability for fMCs with $|S_2| = 1$ may be done by polynomial-time graph algorithms[1] and is thus tractable.

6.2 Sketching Probabilistic Programs

Sketches may be either seen as an application of, or an extension to fMCs. In this section, we use sketches to simultaneously describe fMCs at a symbolic level, extended with some convenience features for modellers, and to formalise information to be exploited by algorithms. A view on sketching as an application is part of Section 7.3.

[1] This statement contradicts a statement in [15], where we claimed that this problem was NP-hard. Indeed, its proof relied on the connection to pMCs and POMDPs, which is only valid with $|S_2| > 1$.

6.2.1 What are sketches?

A program *sketch* [Sol13], a syntactic template, defines a high-level structure of an underlying operational model and allows for a concise representation of a-priori knowledge about the system under development. It effectively restricts the size of the design space and also allows to concisely add constraints and costs to various designs. Program sketching has been successfully applied to e.g., scientific programs and concurrent data structures [ASFS18; GPS17]. We propose to extend sketches to probabilistic programs and in particular, to a language which is easily supported by model checkers. Therefore, we focus our presentation on PRISM program sketches. sketch

A sketch is a program that contains *holes*. Holes are the program's open parts and may be replaced by one of finitely many *options*. Fundamentally, they are declared as: holes option

$$\texttt{hole}\ h\ \texttt{either}\ \{\texttt{expr}_1, \ldots, \texttt{expr}_k\}$$

where h is the hole identifier and $\texttt{expr}_i$ is an expression over the program variables describing the option. Each option may be named and associated with a cost:

$$\texttt{hole}\ h\ \texttt{either}\ \{x_1\ \texttt{is}\ \texttt{expr}_1\ \texttt{cost}\ c_1, \ldots, n_k\ \texttt{is}\ \texttt{expr}_k\ \texttt{cost}\ c_k\}$$

where n_i is the option name and c_i is the cost given as expressions over natural numbers. A hole h may be used in the updates of a command, in a similar way as a constant, and may occur multiple times within one or multiple commands. Extensions to such occurrences are discussed in Section 7.3.2.1. The option names may be used to describe constraints on realisations. These propositional formulae over option names restrict realisations, e.g.,

$$\texttt{constraint}\ (x_1 \lor x_2) \implies x_3$$

requires that whenever the options x_1 or x_2 are taken for some (potentially different) holes, option x_3 is also to be taken.

Definition 6.9 (Program sketch) A *(PRISM program) sketch* is a tuple $\mathcal{S} := (\mathcal{L}_{\mathcal{H}}, \mathbf{O}_{\mathcal{H}}, \Gamma, \mathsf{cost})$ where $\mathcal{L}_{\mathcal{H}}$ is a program with a set $\mathcal{H}$ of holes, and the holes have options $\mathbf{O}_{\mathcal{H}} := \bigcup_{h \in \mathcal{H}} \mathbf{O}_h$, Γ are Boolean expressions over $\mathbf{O}_{\mathcal{H}}$, and $\mathsf{cost}\colon \mathbf{O}_{\mathcal{H}} \to \mathbb{N}$ option-costs. sketch

Example 6.8 We consider a small running example to illustrate the main concepts. Figure 6.7a depicts the program sketch $\mathcal{S}$ with holes $\mathcal{H} = \{\mathrm{HX}, \mathrm{HY}, \mathrm{HZ}\}$. For HX, the options are $\mathbf{O}_{\mathrm{HX}} = \{1, 2\}$. The constraint forbids XA and YA both being one; it ensures a non-trivial random choice in state `s=0`.

```
hole HX either { XA is 1 cost 3, 2}
hole HY either { YA is 1, 3 }
hole HZ either { 1, 2 }
constraint  !(XA && YA);
module rex
s : [0..3] init 0;
s = 0 -> 0.5: s'=HX + 0.5: s'=HY;
s = 1 -> 1: s'=s+HZ;
s >= 2 -> 1: s'=s;
endmodule
```

(a) Program sketch $\mathcal{S}$

```
module rex
s : [0..3] init 0;
s = 0 -> 0.5: s'=1 + 0.5: s'=3;
s = 1 -> 1: s'=s+2;
s >= 2 -> 1: s'=s;
endmodule
```

(b) Instantiation $\mathcal{S}(\{\mathrm{HX} \mapsto 1, \mathrm{HY} \mapsto 3, \mathrm{HZ} \mapsto 2\})$

Figure 6.7: Running example

Remark 6.6 Below, we formalise notions previously used on families. We deem this necessary as there is not a one-to-one correspondence between families and sketches. The relation between sketch and family is further explored in Section 6.2.2. Nevertheless: holes and parameters are similar, the set of options are parameter domains, and realisations and assignments both yield concrete instantiations from a sketch or family. The synthesis problems carry over naturally.

·realisation

instantiation

Definition 6.10 (Realisations of sketches) Let $\mathcal{S} := (\mathcal{L}_{\mathcal{H}}, \mathbf{O}_{\mathcal{H}}, \Gamma, \mathsf{cost})$ be a sketch, a *sketch realisation* on holes $\mathcal{H}$ is a function

$$\mathbf{v} \colon \mathcal{H} \to \mathbf{O}_{\mathcal{H}} \text{ such that for all } h \in \mathcal{H} : \mathbf{v}(h) \in \mathbf{O}_h,$$

and that satisfies all constraints in Γ. The *sketch instantiation* $\mathcal{S}[\mathbf{v}]$ for realisation $\mathbf{v}$ is the program (without holes) $\mathcal{L}_{\mathcal{H}}[\mathcal{H} \leftarrow \mathbf{v}(\mathcal{H})]$ in which each hole $h \in \mathcal{H}$ in $\mathcal{L}_{\mathcal{H}}$ is replaced by $\mathbf{v}(h)$. The cost $\mathbf{c}(\mathbf{v})$ is the sum of the cost of the selected options, $\mathbf{c}(\mathbf{v}) := \sum_{h \in \mathcal{H}} \mathsf{cost}(\mathbf{v}(h))$.

Example 6.9 We continue Example 6.8. The program in Figure 6.7b reflects $\mathcal{S}[\mathbf{v}]$ for realisation $\mathbf{v} := \{\mathrm{HX} \mapsto 1, \mathrm{HY} \mapsto 3, \mathrm{HZ} \mapsto 2\}$, with $\mathbf{c}(\mathbf{v}) = 3$ as $\mathsf{cost}(\mathbf{v}(\mathrm{HX})) = 3$ and all other options have cost zero. For realisation $\mathbf{v}' = \{\mathrm{HX} \mapsto 2, \mathrm{HY}, \mathrm{HZ} \mapsto 1\}$, $\mathbf{c}(\mathbf{v}') = 0$. The assignment $\{\mathrm{HX}, \mathrm{HY}, \mathrm{HZ} \mapsto 1\}$ violates the constraint and is not a realisation. In total, $\mathcal{S}$ represents $6 = 2^3 - 2$ programs and their underlying MCs.

Sketches are well-defined, if all their realisations are well-defined programs.

Related Work | In the domain of quantitative reasoning, sketches and likelihood computation are used to find probabilistic programs that best match available data [NORV15]. An extension to handle parameters affecting transition rates has been integrated into the evolutionary-driven synthesis [CCGK+18] and is available in Rodes [CCGK+17].

```
hole HY either { 0, 1 }
module fam
s : [0..2] init 0;
t : [0..3] init 0;
s = 0 & t = 0 -> 0.5: s'=1 + 0.5: s'=2;
s > 0 & t = 0 -> 1: s'=HY & t'=s;
t = 1 | t = 3 -> 1: t'=3;
t = 2 -> 1: s'=1-s & t'=3;
endmodule
```

(a) Sketching an fMC

```
hole HX either { 1, 2, 4 }
hole HY either { 1, 3, 5 }
module rex
s : [0..10] init 0;
t : [0..10] init 0;
s > t -> 0.5: s'=HX+HY + 0.5: s'=HY & t'=HX;
s < t -> 1: s'=s+HX & t'=HX;
s = t -> 1: s'=s;
endmodule
```

(b) Sketching beyond an fMC

Figure 6.8: Sketches versus fMCs

These sketches model continuous-time Markov chains. ProFeat [CDKB18] is closest to the sketching language proposed here. It is a meta-language on top of PRISM that helps to describe sets of PRISM programs. It contains constructs specifying the structure of the program, rather than a program with some parts left open. Due to the complex structure of the ProFeat language, analysis typically iterates over all instantiations.

6.2.2 Families versus sketches

The following example clarifies how sketches are a straightforward way to model fMCs.

> **Example 6.10** The sketch in Figure 6.8a reflects the fMC from Example 6.4 as depicted in Figure 6.3c on page 164. Although the construction here is slightly compressed, the idea is simple. The program contains two variables to reflect S_1 and S_2, and one hole to reflect the parameter.

The constraints on sketches allow to concisely describe a subfamily directly in the modelling language. However, giving semantics to sketches in terms of fMC is not straightforward. We illustrate the challenge with the following example.

> **Example 6.11** The sketch in Figure 6.8b goes beyond the sketch in Figure 6.8a. First, both variables are affected by the holes. Second, the same hole appears in different variables, thereby encoding implicit constraints on the possible combinations of commands in different instantiations of the sketch, and also between the assignments to the different variables in the program.

All additional dependencies between transitions may be expressed by some combination of constraints. More variables in the program may be reflected by fMCs with state space tuples with an entry for every variable. Naturally, families could be extended to formally support such constructions, but this would significantly clutter their

definition[1]. Still, we are interested in even extending sketches. We allow the parallel composition of modules and further extensions to ease modelling. Therefore, we refrain from giving formal fMC semantics to sketches. In Section 7.3, we show extensions to the sketching language proposed here and discuss a less rigorous handling of sketches that are not well-formed. Within this thesis, fMCs are thus a construct that helps to formalise ideas and algorithms that we want to apply on sketches.

fMCs are an adequate model to concisely capture finite sets of similar MCs with a (slightly) varying topology. Sketches are programs with holes and, like fMCs, may be used to concisely define sets of programs (that describe MCs).

[1]Tool support for fMCs actually covers these more complex models implicitly.

7 Employing Parameter Synthesis

Solving other problems with parameter synthesis

Summary | We collect and discuss three different active research areas closely connected to parameter synthesis. We clarify how parameter synthesis may be seen as a fundamental underlying problem. The first area is small-memory strategies for partially observable MDPs (POMDPs). In POMDPs, finding strategies satisfying an indefinite horizon specification is an undecidable problem in general, due to the complexity of optimal strategies. It is natural to search for simpler strategies, and this search for simpler strategies coincides with feasibility for pMCs. The second area is concurrent stochastic games, which are an adequate model for distributed processes. The connection is similar to the relation between POMDPs and pMCs, and the additional source of nondeterminism naturally leads to pMDPs. The third area considers software product lines and their connection to sketching of probabilistic programs. We illustrate typical problems and the connection to synthesis in fMCs and sketches. It particularly addresses some requirements that go beyond the analysis aspects of parameter synthesis.

Origins | The contents of this chapter are collected from various sources. The first section is extracted from [12], and the second section is extracted from [19]. The last section is based on [13; 14].

Background | For the first two sections, pMCs and their subclasses (Chapter 4 and Section 5.1) are essential. For the third section, families and sketches (Chapter 6) are essential.

7.1 Partially Observable MDPs

MDPs are an adequate model to describe sequential decision processes, where the decision made in some state affects the evolution of the system, and the decisions to be made. MDPs allow to make a decision in every state independently, i.e., a strategy maps states to actions. Such a map requires that we are fully aware of the system state. Many applications, however, allow only for *partial observability* of the current system state. Examples include robotics [TBF05] software verification [CCHR+11], controls [WF12]

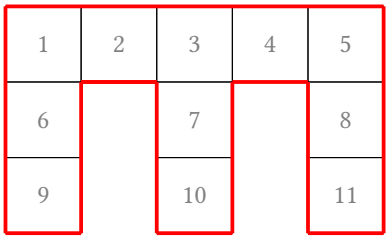


Figure 7.1: The maze

and planning [KLC98; Koc15]—e.g., to save endangered species [CCMN+12]. For such applications, MDPs are extended to *partially observable Markov decision processes* (POMDPs) [Åst65; KLC98; RN10] by labelling states with some *observation*. While acting within the environment, we encounter certain *observations*, according to which we infer the likelihood of the system being in a certain state. This likelihood is called the *belief state*. Executing an action leads to an update of the belief state according to new observations. The belief states together with an update function form a typically uncountably infinite MDP, referred to as the *belief MDP* [Åst65; Son78].

observations

belief MDP

A strategy for a POMDP has to base its decisions on the sequence of observations, i.e., it has to be *observation-based*. Observation-based memoryless strategies do not suffice to optimise the probability of reaching a target, or to minimise the expected steps to a target. Optimal observation-based strategies may be computed on the (fully observable) belief MDP [SC05; SPK13; WS17; NPZ17], but when the reachable fragment of that MDP is uncountably large, such strategies are also infinitely large. Indeed, deciding whether there exists an observation-based strategy satisfying an unbounded reachability property is undecidable [MHC03].

POMDP strategies from an uncountable belief MDP are *infinite-state controllers*. For computational tractability, strategies are often restricted to finite memory; this amounts to using *randomised finite-state controllers* (*FSCs*) [MKKC99]. We often refer to such strategies as FSCs. The computation of a memoryless strategy adhering to a specification has been shown to be NP-hard, SQRT-SUM-hard, and in PSPACE [VLB12]. While optimal values cannot be guaranteed, a small amount of memory in combination with *randomisation* may suffice to find near-optimal strategies [CAH04]. Standard algorithms are based on optimisation and search techniques [BB04; ABZ10].

FSC

Example 7.1 Consider a POMDP for the maze (the so-called cheese maze, e.g. in [McC92; LCK95; CCGK16]) as in Figure 7.1. The maze has 11 rooms. Three rooms are special: the bottom centre room (nr. 10) is the goal room and the two other bottom rooms contain traps (nr. 9, 11). We start in some room r, as specified

below, and try to move to the goal room. From the bottom rooms (i.e., 9, 10, 11), leaving the location is impossible. In all other rooms (i.e., ≤ 8), we may select a direction either north/east/south/west which is not blocked by a wall, i.e., we may only select directions in which there is an adjacent room r'. Taking the action has a deterministic outcome; we move to the room r'. All non-bottom rooms look similar except for the present walls, thus we cannot observe in which exact room we currently are, but we do observe the blocked directions. Consequently, rooms $\{2, 4\}$ and rooms $\{6, 7, 8\}$ are indistinguishable. The POMDP consists of 12 states: an initial state and 11 states corresponding to the rooms in the maze. We have 9 different observations in the POMDP. Below, we consider two scenarios.

Scenario 1. We start (uniformly distributed) in rooms 1 or 5. Technically, the initial state has one outgoing action which leads to room 1 or 5, both with probability $1/2$. Then, there is no deterministic memoryless observation-based strategy such that we almost surely reach the goal room: In rooms 2 and 4, we either go west or east. As the strategy has to be observation-based, in both rooms, we make the same decision. Assume we go west. The following (fragment of a) strategy ensures that we reach the target from state 3:

$$\{1 \mapsto \text{east}, \quad 2, 4, 5 \mapsto \text{west}, \quad 3, 6, 7, 8 \mapsto \text{south}\}.$$

However, from room 1, we cannot reach the target, and thus, from the initial state, we reach the target with probability $1/2$. To reach the target from room 1, we need to go east in room 2, but this symmetrically prevents reaching the target from room 5. However, there is a randomised memoryless observation-based strategy such that we reach the goal room almost surely:

$$\{2, 4 \mapsto \left(\frac{1}{2} : \text{west} + \frac{1}{2} : \text{east}\right) \quad 1 \mapsto \text{east}, \quad 5 \mapsto \text{west}, \quad 3, 6, 7, 8 \mapsto \text{south}\}.$$

Indeed, this strategy ensures that we eventually reach room 3 and from their the target when we start in room 1 or 5.

Scenario 2. We start (uniformly distributed) in rooms 6 or 8. There is no memoryless observation-based strategy such that we almost surely reach the goal room: In rooms 6 and 8, it is clear that we need to go north with probability 1, as otherwise we immediately reach a trap room with some positive probability. However, if we go north in rooms 6 or 8 (with probability one), we also go north in state 7, thereby cutting off the only path to the goal room. A strategy with one bit of memory suffices to reach the target with probability one: In the first step, we go north and switch the memory bit. Then, we behave as if we start in rooms 1 and 5, discussed above.

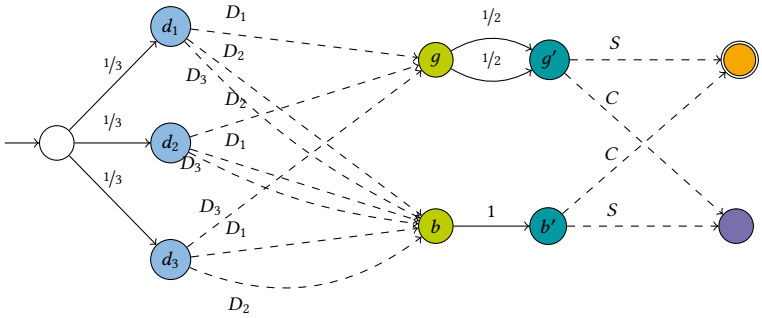


Figure 7.2: Monty-Hall POMDP

In this section, we show how to employ parameter synthesis to construct correct-by-construction FSCs for POMDPs subject to reachability or expected reward properties. We show that the pMC parameter synthesis problem and the POMDP strategy synthesis problem are equally hard. This insight yields new complexity results (via Section 5.4) and enables using a plethora of methods for parameter synthesis.

We first formalise POMDPs and then detail the relation to parameter synthesis. We end the section with a comparison with two state-of-the-art POMDP tools.

7.1.1 What are partially observable MDPs?

POMDP
·underlying MDP
observation function

Definition 7.1 (POMDP) A *partially observable MDP* (*POMDP*) is a tuple $\Omega = (M, Z, \mathsf{Obs})$, with $M = (S, \iota, Act, P, \mathsf{rew})$ the *underlying MDP* of Ω, Z a finite set of observations and $\mathsf{Obs}\colon S \to Z$ the *observation function*.

We require that states with the same observations have the same set of available actions, i.e., $\mathsf{Obs}(s) = \mathsf{Obs}(s')$ implies $\mathbb{A}\mathrm{ct}(s) = \mathbb{A}\mathrm{ct}(s')$ for all $s, s' \in S$. We define $\mathbb{A}\mathrm{ct}(z) = \mathbb{A}\mathrm{ct}(s)$ if $\mathsf{Obs}(s) = z$. More general observation functions [RGT05; SPK13] take the last action into account and provide a distribution over the observations Z. There is a transformation of the general observation functions to our POMDP definition that blows up the state space polynomially [CCGK15].

Example 7.2 We reconsider the Monty-hall problem as also discussed in Example 6.2 and depict the matching (concise) POMDP in Figure 7.2. We depict nondeterministic choices with Dirac distributions by dashed arrows and indicate the observations with colours. The POMDP is acyclic. The paths merely vary in

the door that has been opened, which can be stored with two bits of memory. Closer analysis yields that memoryless strategies are optimal for this POMDP.

We lift the observation function to paths: For a path

$$\pi = s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} \cdots s_n \in \Pi^M_{\mathsf{fin}},$$

the associated *observation sequence* is

$$\mathsf{Obs}(\pi) := \mathsf{Obs}(s_0) \xrightarrow{\alpha_0} \mathsf{Obs}(s_1) \xrightarrow{\alpha_1} \cdots \mathsf{Obs}(s_n).$$

Several paths in the underlying MDP may yield the same observation sequence. Strategies have to take this restricted observability into account.

Definition 7.2 An *observation-based strategy* $\sigma \in \mathfrak{S}^M$ for a POMDP Ω is a strategy for the underlying MDP M such that $\sigma(\pi) = \sigma(\pi')$ for all $\pi, \pi' \in \Pi^M_{\mathsf{fin}}$ with $\mathsf{Obs}(\pi) = \mathsf{Obs}(\pi')$. The set $\mathfrak{S}^\Omega_o$ contains all observation-based strategies for Ω. ·observation-based

An observation-based strategy selects actions based on the observation sequence along a path (and the past actions). Applying the strategy to a POMDP yields an induced MC as in Definition 3.5 on page 40, resolving all nondeterminism and partial observability. We denote the set of all memoryless observation-based strategies with $\mathfrak{S}^\Omega_{o,1}$.

7.1.2 Relation to parameter synthesis

We show that finding observation-based strategies corresponds to feasibility synthesis in pMCs. We start with the memoryless case, formalised below, to give the intuition. We then sketch how to apply this to (small) finite memory strategies in Section 7.1.3.

Definition 7.3 Given a POMDP Ω and a property φ, the problem $\exists$-MEMLESS-STRAT asks

is there a memoryless observation-based strategy σ with $\Omega[\sigma] \models \varphi$?

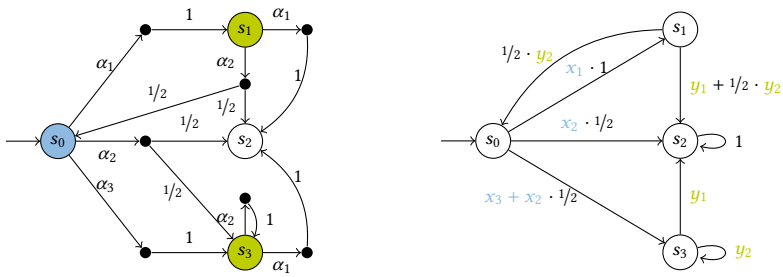
7.1.2.1 From POMDP to pMC: memoryless strategies

We show the following theorem:

Theorem 7.1 Let Ω be a POMDP. There exists a simplex pMC $\mathcal{D}_\Omega$ such that

$$\{\Omega[\sigma] \mid \sigma \in \mathfrak{S}^\Omega_{o,1}\} = \langle \mathcal{D} \rangle.$$

The idea behind this theorem is a straightforward adaption of Definition 5.13 on page 116. There, we show that we may interpret nondeterministic choices as the



(a) POMDP (b) induced pMC

Figure 7.3: From POMDPs to pMCs

choice of parameter values. In POMDPs, we have an additional constraint that for two states s, s' with $\mathsf{Obs}(s) = \mathsf{Obs}(s')$, the distributions $\sigma(s) = \sigma(s')$ coincide, i.e., there are additional dependencies. These dependencies are covered by using the same parameters. We first illustrate this idea:

Example 7.3 Consider the POMDP in Figure 7.3a. When observing z_1 in state s_1 or s_3, a strategy selects a distribution over $\mathbb{A}\mathrm{ct}(z_1)$. Equivalently, the strategy σ takes α_1 with some probability q_1 and α_2 with some probability q_2. Similarly, probabilities p_1, p_2, p_3 reflect the strategy in state s_0. Thus, an induced MC $\Omega[\sigma]$ contains these values q_1, q_2, p_1, p_2, p_3. In general, for each strategy these values are assignments to some placeholders or parameters x_1, x_2, y_1, y_2, y_3, e.g., $\mathsf{val}(y_1) := q_1$. Substituting these values in a template of the induced MC that is inferred from the POMDP structure, yields the induced MC for the particular strategy. More precisely, fixing the values for the parameters by a valuation val yields a concrete MC—the template immediately corresponds to a pMC, as given in Figure 7.3b.

·corresponding pMC

We formalise the construction of the *corresponding pMC*:

Definition 7.4 (Corresponding pMC) For POMDP $\Omega = (M, Z, \mathsf{Obs})$ with $M = (S, \iota, Act, P, \mathsf{rew})$, we define the *corresponding pMC* $\mathcal{D}_\Omega := (S, \iota, X, \mathcal{P}', \mathsf{rew})$ with

$$X' := \{x_{z,\alpha} \mid z \in Z, \alpha \in \mathbb{A}\mathrm{ct}(z)\}, \quad \text{and}$$

$$\mathcal{P}'(s, s') := \begin{cases} P(s, \alpha, s') & \text{if } \mathbb{A}\mathrm{ct}(s) = \{\alpha\}, \\ \sum_{\alpha \in \mathbb{A}\mathrm{ct}(\mathsf{Obs}(s))} x_{\mathsf{Obs}(s),\alpha} \cdot P(s, \alpha, s') & \text{otherwise.} \end{cases}$$

Observe that the construction yields expanded simplex pMCs if the MDP was al-

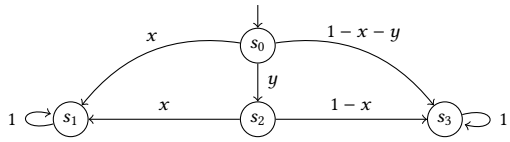


Figure 7.4: A non-simple pMC

ternating, cf. Definition 5.10[1]. They may be transformed in simple(x) pMCs by the constructions from Section 5.1.

Corollary 7.2 Given a corresponding pMC $\mathcal{D}_\Omega$ and a property φ. For every $\sigma \in \mathfrak{S}^{\Omega}_{o,1}$ and the corresponding $\mathsf{val} \in \mathsf{Val}^{\mathrm{wd}}_{\mathcal{D}_\Omega}$:

$$\Omega[\sigma] = \mathcal{D}_\Omega[\mathsf{val}].$$

From this corollary, Theorem 7.1 follows immediately. Furthermore, we have seen that the number of parameters is the main indicator for the complexity of parameter synthesis. The following formalises the number of parameters in a pMC corresponding to some POMDP.

Lemma 7.3 (Number of Parameters) Given a POMDP Ω. The number of parameters in the pMC $\mathcal{D}_\Omega$ is in $\mathcal{O}\big(|Z| \cdot \max_{z \in Z} |\mathbb{A}\mathrm{ct}(z)|\big)$.

7.1.2.2 From pMC to POMDP

The following theorem is the reverse of Theorem 7.1.

Theorem 7.4 Let $\mathcal{D}$ be a simplex pMC (with constant rewards). There exists a POMDP $\Omega_{\mathcal{D}}$ such that

$$\langle \mathcal{D} \rangle = \{\Omega_{\mathcal{D}}[\sigma] \mid \sigma \in \mathfrak{S}^{\Omega_{\mathcal{D}}}_{o,1}\}.$$

States in the pMC with the same occurring parameter group get the same observation in the POMDP. This intuition motivates the restriction to simplex pMC:

Example 7.4 Consider the pMC in Figure 7.4. Observe that the parameters are not grouped: the parameter x occurs in two different distributions (at s_0 and s_2). To define a reduction where the resulting POMDP has the same set of states, there are two options for the observation function at the states s_0 and s_2: Either $\mathsf{Obs}(s_0) = \mathsf{Obs}(s_2)$ or $\mathsf{Obs}(s_0) \neq \mathsf{Obs}(s_2)$. Intuitively, every (parametric)

[1] When making a POMDP alternating, we may give all new states the same observation.

transition in the pMC corresponds to an action choice in a POMDP. Then $\mathsf{Obs}(s_0) = \mathsf{Obs}(s_2)$ is impossible as s_0 and s_2 have a different number of outgoing transitions (outdegree). Adding a self-loop to s_2 does not alleviate the problem. Moreover, $\mathsf{Obs}(s_0) \neq \mathsf{Obs}(s_2)$ is impossible, as a strategy could distinguish s_0 and s_2 and assign different probabilities to p.

The following construction is an exact inverse of Definition 7.4:

·corresponding POMDP

Definition 7.5 Let $\mathcal{D} = (S, \iota, X, \mathcal{P}, \mathsf{rew})$ be a simplex pMC with parameter groups $X_1, \ldots, X_m$. We define the *corresponding POMDP* $\Omega_{\mathcal{D}} := (M, Z, \mathsf{Obs})$ with underlying MDP $M := (S, \iota, Act, P, \mathsf{rew})$, where $Act := \{\alpha_x \mid x \in X\} \cup \{\bot\}$,

$$P(s, \alpha, s') := \begin{cases} 1 & \text{if } \mathcal{P}(s, s') = x \in \mathsf{occurs}(s), \alpha = \alpha_x, \\ \mathcal{P}(s, s') & \text{if } \mathsf{occurs}(s) = \emptyset, \alpha = \bot, \\ 0 & \text{otherwise,} \end{cases}$$

and $Z := \{X_1, \ldots, X_m\}$ with $\mathsf{Obs}(s) = \mathsf{occurs}(s)$.

Corollary 7.5 For every pMC $\mathcal{D}$ there exists a POMDP Ω such that $\mathcal{D}$ is the corresponding pMC for Ω, i.e. such that $\mathcal{D} = \mathcal{D}_\Omega$.

7.1.2.3 Consequences for complexity

As immediate corollary to the (polynomial-time) constructions above, we obtain:

Corollary 7.6 The decision problems $\exists$-MEMLESS-STRAT and $\exists$REACH are polynomial-time equivalent.

Consequently, using Theorem 5.49, we obtain the following new complexity result:

Theorem 7.7 Given a POMDP Ω and a quantitative property φ with a non-strict comparison relation, then deciding

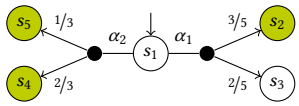
exists a memoryless observation-based strategy σ such that $\Omega[\sigma] \models \varphi$?

is ETR-complete.

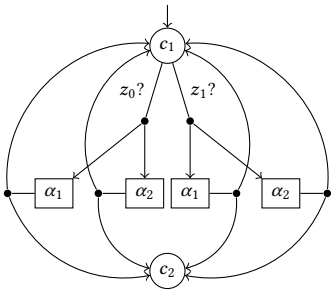
7.1.3 Adding memory

As illustrated in Example 7.1, memoryless strategies are often too restricted, and already adding a bit[1] of memory may help to drastically improve the strategy.

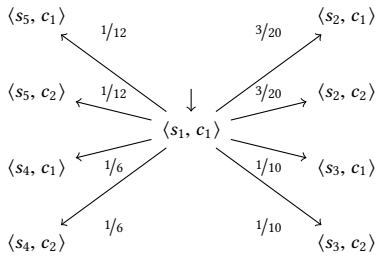
[1] no pun intended.



(a) The POMDP Ω with observations $z_0 := \mathsf{Obs}(s_1) = \mathsf{Obs}(s_3)$ and $z_1 := \mathsf{Obs}(s_2) = \mathsf{Obs}(s_4) = \mathsf{Obs}(s_5)$.



(b) (partial) FSC $\mathcal{A}$ with control nodes c_1, c_2



(c) A part of MC $\Omega[\sigma_{\mathcal{A}}]$ induced by Ω and $\mathcal{A}$.

Figure 7.5: Illustrating FSCs for POMDPs

7.1.3.1 How to represent finite memory strategies

To represent observation-based strategies with finite memory, we define *finite-state controllers* (FSCs). A randomised observation-based strategy for a POMDP Ω with (finite) m memory is represented by an FSC $\mathcal{A}$ with m control nodes. If $m = 1$, the FSC describes a *memoryless strategy*. We use FSCs to refer to the corresponding strategies.

finite-state controllers

Definition 7.6 (FSC) A *finite-state controller* (*FSC*) for a POMDP Ω is a tuple $\mathcal{A} = (C, c_0, \theta, \delta)$, where C is a finite set of *control nodes*, $c_0 \in C$ is the *initial control node*, the *action mapping*

FSC

·action mapping

$$\theta\colon C \times Z \to \mathit{Distr}(\mathit{Act}) \text{ with } \mathit{supp}(\theta(c, z)) \subseteq \mathbb{A}\mathrm{ct}(z) \text{ for all } c \in C, z \in Z,$$

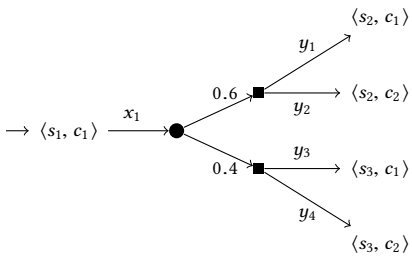
and the *control update*

·control update

$$\delta\colon C \times Z \times \mathit{Act} \to \mathit{Distr}(C).$$

The set FSC_m^{Ω} denotes the set of FSCs with m control nodes, called *m-FSC*s.

A $\sigma_{\mathcal{A}} \in \mathfrak{S}_o^{\Omega}$ denotes the observation-based strategy represented by $\mathcal{A}$. The set $\mathfrak{S}_{o,m}^{\Omega}$ denotes the observation-based strategies representable by an m-FSC. From a control node c and the observation z in the current state of the POMDP, the next action α is chosen from $\mathbb{A}\mathrm{ct}(z)$ randomly as given by $\theta(c, z)$. The successor control node of the FSC is then determined randomly via $\delta(c, z, \alpha)$.



(a) Induced pMC (fragment)

Act	*P*	Node	Result
$\alpha_1 : x_1$	$3/5$	$c_1 : y_1$	$3/5 \cdot x_1 \cdot y_1$
		$c_2 : y_2$	$3/5 \cdot x_1 \cdot y_2$
	$2/5$	$c_1 : y_1$	$2/5 \cdot x_1 \cdot y_1$
		$c_2 : y_2$	$2/5 \cdot x_1 \cdot y_2$
$\alpha_2 : x_2$	$2/3$	$c_1 : y_3$	$2/3 \cdot x_2 \cdot y_3$
		$c_2 : y_4$	$2/3 \cdot x_2 \cdot y_4$
	$1/3$	$c_1 : y_3$	$1/3 \cdot x_2 \cdot y_3$
		$c_2 : y_4$	$1/3 \cdot x_2 \cdot y_4$

(b) Parameterised transition probabilities

Figure 7.6: Induced parametric Markov chain for FSCs.

Example 7.5 We use a fragment of a POMDP Ω as depicted in Figure 7.5a, with two observations z_0 and z_1. Figure 7.5b shows an excerpt of an FSC $\mathcal{A}$ with two control nodes. From node c_1, the action mapping distinguishes observations z_0 and z_1. The solid dots indicate a probability distribution from $Distr(Act)$. For readability, all distributions are uniform, probabilities are omitted and we omit the action mapping for c_2. A fragment of the induced MC $\Omega[\sigma_{\mathcal{A}}]$ is shown in Figure 7.5c. Assume Ω is in state s_1 and $\mathcal{A}$ in node c_1. Based on the observation $z_0 := \mathsf{Obs}(s_1)$, $\sigma_{\mathcal{A}}$ chooses action α_1 with probability $\theta(c_1, z_0)(\alpha_1) = 1/2$ leading to the probabilistic branching in the POMDP. With probability $3/5$, Ω evolves to state s_2. Next, the FSC $\mathcal{A}$ updates its control node; with probability $\delta(c_1, z_0, \alpha_1)(c_1) = 1/2$, $\mathcal{A}$ stays in c_1. The corresponding transition from $\langle s_1, c_1\rangle$ to $\langle s_2, c_1\rangle$ in $\Omega[\sigma_{\mathcal{A}}]$ has probability $1/2 \cdot 3/5 \cdot 1/2 = 3/20$.

7.1.3.2 The idea

Next, we define the pMC that results from applying an m-FSC to a POMDP. The idea is to assign parameters as arbitrary probabilities to action and control node choices.

Example 7.6 Recall Figure 7.5 and Example 7.5. The action mapping θ and the control update δ have arbitrary but fixed probability distributions. As in the memoryless case, we describe them with parametric transitions. For α_1, we represent the probability $\theta(c_1, z_0)(\alpha_1)$ by parameter x_1. The control update yields $\delta(c_1, z_0, \alpha_1)(c_1)$, represented by y_1, and $\delta(c_1, z_0, \alpha_1)(c_2)$, represented by y_2, respectively. Figure 7.6a shows the induced pMC for action choice α_1. For instance, the transition from $\langle s_1, c_1\rangle$ to $\langle s_2, c_1\rangle$ is labelled with polynomial $x_1 \cdot 3/5 \cdot y_1$.

We collect all polynomials for observation z_0 in Figure 7.6b. The *result* column describes a *parameterised distribution* over tuples of states and control nodes. Valuations for these parameters need to sum up to one.

Definition 7.7 Let $\Omega = (M, Z, \mathsf{Obs})$ be a POMDP with underlying MDP $M = (S, \iota, Act, P, \mathsf{rew})$ and $m > 0$ a memory bound. Let $C \coloneqq \{1, \ldots, m\}$. We define the pMC $\mathcal{D}_{\Omega,m} \coloneqq (S \times C, \langle \iota, 1\rangle, X, \mathcal{P}, \mathsf{rew}')$ with

$$X \coloneqq \{x_{c,z,\alpha} \mid c \in C, z \in Z, \alpha \in Act\} \cup \{x_{c,z,\alpha,c'} \mid c, c' \in C, z \in Z, \alpha \in Act\},$$

rew' defined by $\mathsf{rew}(\langle s', c\rangle) = \mathsf{rew}(s)$ and, using $z \coloneqq \mathsf{Obs}(s)$:

$$\mathcal{P}(\langle s, c\rangle, \langle s', c'\rangle) \coloneqq \begin{cases} P(s, \alpha, s') \cdot x_{c,z,\alpha,c'} & \text{if } \mathbb{A}\mathrm{ct}(s) = \{\alpha\}, \\ \sum_{\alpha \in \mathbb{A}\mathrm{ct}(z)} x_{c,z,\alpha} \cdot x_{c,z,\alpha,c'} \cdot P(s, \alpha, s') & \text{otherwise.} \end{cases}$$

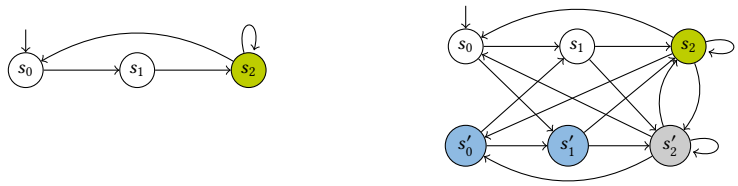
The following formulation is equivalent and allows to reuse earlier results: For a POMDP Ω and memory bound $m > 1$, we construct a POMDP Ω_m such that Ω satisfies a specification φ under some m-FSC iff Ω_m satisfies φ under some 1-FSC.

Definition 7.8 (*m*-Unfolding) Let $\Omega = (M, Z, \mathsf{Obs})$ be a POMDP with $M = (S, \iota, Act, P, \mathsf{rew})$ and $m > 1$. The *m-unfolding* of Ω is the POMDP $\Omega_m \coloneqq (M_m, Z_m, \mathsf{Obs}_m)$ with $M_m \coloneqq (S_m, \iota_m, Act_m, P_m, \mathsf{rew}_m)$ defined by: ·unfolding

- $S_m \coloneqq S \times \{0, \ldots, m{-}1\}$,
- $\iota_m \coloneqq \langle \iota, 0\rangle$,
- $Act_m \coloneqq Act \times \{0, \ldots, m{-}1\}$
- $P_m\big(\langle s, i\rangle, \langle \alpha, j\rangle, \langle s', i'\rangle\big) \coloneqq \begin{cases} P(s, \alpha, s') & \text{if } i' = j, \\ 0 & \text{otherwise.} \end{cases}$
- $\mathsf{rew}_m(\langle s, i\rangle \coloneqq \mathsf{rew}(s)$,

and $Z_m = Z \times \{0, \ldots, m{-}1\}$, $\mathsf{Obs}_m\big(\langle s, i\rangle\big) = \big\langle \mathsf{Obs}(s), i\big\rangle$.

Intuitively, Ω_m stores the current control node into its state space. At state $\langle s, i\rangle$ of Ω_m, a 1-FSC may not only choose between the available actions $\mathbb{A}\mathrm{ct}(s)$ in Ω, but also between different successor control nodes. Figure 7.7 shows this process for $m = 2$. All states of the POMDP are copied once. Different observations allow to determine in which copy of a state—and therefore, which control node—we currently are. Additionally, all actions are duplicated to model the option for a strategy to switch the control node. The corresponding pMC $\mathcal{D}_{\Omega_m}$ of the m-unfolding of Ω has the same topology as the induced pMC $\mathcal{D}_{\Omega,m}$ of Ω with memory bound m. In fact, both pMCs have the same instantiations. We obtain:



(a) POMDP

(b) 2-unfolding

Figure 7.7: Unfolding a POMDP for two control nodes: We omitted action labels and probabilities to avoid clutter.

Proposition 7.8 For POMDP Ω and memory bound m:

$$\langle \mathcal{D}_{\Omega_m} \rangle = \langle \mathcal{D}_{\Omega,m} \rangle$$

Lemma 7.9 (Number of Parameters) The number of parameters in the induced pMC $\mathcal{D}_{\Omega,m}$ is in $\mathcal{O}\big(m^2 \cdot |Z| \cdot \max_{z \in Z} |\mathbb{A}\mathrm{ct}(z)|\big)$.

The intuition is that in both pMCs the parameter instantiations reflect arbitrary probability distributions over the same set of successor states. Details are given in [12].

7.1.4 Outlook

Naturally, the ideas presented in this section may be extended to restricted classes of FSCs, e.g., FSCs that encode variants of counters, or FSCs that do not use randomisation. In the latter case, the connection between FSCs and pMCs and the connection between pMCs and fMCs immediately yields that feasibility in fMCs exactly reflects finding small-memory deterministic strategies. Other types of FSCs are particularly interesting when there are domain-specific restrictions or knowledge that help determine the type of FSC that is to be synthesised. Such restrictions may be formulated as sketches, cf. Section 7.3. We want to stress that the literature also contains various variants of FSC definitions. One should be careful to transfer results for other types of FSCs to this setting. Further details may be found in [12].

7.1.5 Empirical evaluation

Many state-of-the-art POMDP solvers mainly consider expected discounted reward measures [WS17], which are a subclass of indefinite horizon properties [KMW12]. A notable exception is an extension to PRISM [NPZ17]. The goal of this section is to briefly investigate the impact that parameter space partitioning may have on the state of the

art in solving POMDPs, not to evaluate pMC methods. We delay a discussion of the selected approaches to the respective sections. We distinguish three key problems:

1. *Find a correct-by-construction strategy for a POMDP and a specification.* To construct such a strategy, we use a method for feasibility. Most solution techniques focused on pMCs with a few parameters, rendering the problem at hand infeasible. We employ PSO (Section 8.3) and CCP (Chapter 9) for our evaluation.

2. *Prove that no m-FSC exists for a POMDP and a specification.* Proving the absence of an FSC with the given memory bound allows us to show ε-optimality of a previously synthesised strategy (w.r.t. the memory bound). We use *parameter lifting* (Chapter 12) and an ETR encoding (Section 8.2).

3. *Provide a closed-form solution for the underlying measure of a specification* in form of a function over the induced parameters of an FSC. The function may be used for further analysis, e.g., of the sensitivity of decisions or parameter values, respectively. To compute this function, we employ *state elimination* (Section 8.1).

7.1.5.1 Set-up

Implementation | We extended STORM [9] to parse and store POMDPs, and implemented several transformation options to pMCs. Most notably, STORM supports m-unfolding, the product with several restricted FSCs such as counters that can be incremented at will, and several types of transformation to (simple) pMCs.

Environment | We evaluated on a HP BL685C G7 with 48 2 GHz cores, a 16 GB memory limit, and 1800 seconds time limit. The compared methods are single-threaded.

Benchmarks and baseline | We took the POMDPs from PRISM-POMDP [NPZ17], additional maze (as in Example 7.1), load/unload examples from [MKKC99] and a slippery gridworld with traps inspired by [RN10]. Table 7.1 gives details. The specifications (Tp.) either ask to minimise expected costs from an initial state until reaching a specified target set ($\mathbb{E}$) or maximising the probability of reaching from an initial state a target set without hitting a bad state before ($\mathbb{P}$). We list the number of states, branches ($\sum |\mathbb{A}\mathrm{ct}(s)|$) and observations in each POMDP. As a baseline, we provide the results and run time of the model-checking tool PRISM-POMDP and a state-of-the-art point-based solver SolvePOMDP [WS17], obtained with default settings. Both tools compute optimal memory-unbounded strategies and are prototypes. The last column contains the result on the underlying fully observable MDP. The experiments contain minimal expected rewards, which are analysed by a straightforward extension of maximal reachability probabilities. All pMCs computed are transformed into simple pMCs, as parameter synthesis typically benefits from the simpler structure. We invoked all instances with the same (default) settings of the respective methods.

Table 7.1: Benchmarks

			POMDP Ω			PRISM-POMDP		SolvePOMDP		MDP
Id	Name	Tp.	States	Bran.	Obs.	Result	Time (s)	Result	Time (s)	Res
1	NRP (8)	$\mathbb{P}$	125	161	41	[.125, .24]	20		TO	1.0
2	Grid (4)	$\mathbb{E}$	17	62	3	[3.97, 4.13]	1038	4.13	0.4	3.2
3	Netw (3,4,8)	$\mathbb{E}$	2729	4937	361		TO		TO	0.83
4	Crypt (5)	$\mathbb{P}$	4885	11733	890		MO		TO	1.0
5	Maze (2)	$\mathbb{E}$	16	58	8	[5.11, 5.23]	3.9	5.23	16	4.0
6	Load (8)	$\mathbb{E}$	16	28	5	[10.5, 10.5]	1356	10.5	7.6	10.5
7	Slippery (4)	$\mathbb{P}$	17	59	4		TO	0.93	95	1.0

7.1.5.2 Results & Evaluation

We list results for the three questions separately below. We finish the section by a (preliminary) comparison to the baseline tools.

Finding strategies | We evaluate the time to synthesise a strategy that satisfies the property. We vary the threshold used in the property and the structure of the FSC.

Results: We summarise the obtained results in Table 7.2. For each instance (Id), we define three thresholds (λ's), ordered from challenging—close to the optimum—to less challenging. λ_1 is chosen to be nearly optimal for all benchmarks. For different types of FSCs (FSC, F=full, C=counter) and memory bounds (m), we obtain simple pMCs with the given number of states, transitions and parameters. Full FSCs are fully connected and alike to the FSCs discussed in the theory section. In counter-FSCs a control node c is succeeded by either c or $c + 1$. For each threshold ($\lambda_1, \lambda_2, \lambda_3$), we report the run time in seconds of the methods PSO and CCP, respectively. A dash indicates a combination of memory and threshold for which no FSC exists, according to the results in Table 7.4a. TO/MO denote violations of the time/memory limit, respectively.

Evaluation: Strategies for thresholds which are suboptimal (λ_3) are synthesised faster. If the memory bound is increased, the number of parameters quickly grows and the performance of the methods degrades. Trivially, an m-FSC is also an $(m{+}i)$-FSC for each $i \in \mathbb{N}$. Yet, the larger number of parameters make searching for $(m{+}i)$-FSCs significantly more difficult. The simpler FSC topology for a counter alleviates the blow-up of the pMC and is successfully utilised to find strategies for larger instances. Further experiments show that the number of POMDP states has only a minor effect on the performance.

Summarising, many of the POMDPs in the benchmarks allow good performance via FSCs with small memory. We find nearly-optimal—and small—FSCs for POMDP benchmarks with thousands of states within a minute. We furthermore observe that the performance of PSO and CCP is incomparable and both methods have their merits.

Table 7.2: Synthesing FSC strategies by parameter synthesis

Id	λ's	FSC/m	States	Trans	Pars	λ_1		λ_2		λ_3	
						PSO	CCP	PSO	CCP	PSO	CCP
1	.124/.11/.09	F/1	75	118	8	<1	<1	<1	<1	<1	<1
		F/2	205	420	47	2	<1	2	<1	2	<1
		F/4	921	1864	215	9	2	9	2	10	2
		F/8	3889	7824	911	43	15	42	14	42	14
2	4.15/4.5/5.5	F/1	47	106	3	–	–	–	–	Err	<1
		F/2	183	390	15	7.4	11	4	9	2	<1
		F/4	719	1486	63	TO	64	39	91	14	8
		F/8	2845	5788	255	TO	700	TO	946	254	69
3	9/10/15	F/1	3268	13094	276	TO	TO	TO	43	22	4
		F/2	16004	46153	1783	TO	TO	TO	877	152	28
		C/2	11270	36171	1168	TO	TO	TO	358	100	62
		C/4	27183	82145	2940	TO	MO	TO	MO	476	MO
4	.249/.2/.15	F/1	3366	6534	364	18	25	18	15	18	12
		F/2	25713	51608	3907	330	MO	350	MO	326	MO
5	5.2/15/25	F/1	30	64	8	–	–	TO	TO	<1	TO
		F/2	137	294	49	TO	TO	14	TO	2	TO
		F/4	587	1214	219	93	TO	TO	TO	26	TO
		F/8	2421	4924	919	TO	TO	1034	TO	115	TO
		C/2	99	212	33	TO	TO	3.7	TO	<1	TO
		C/4	231	476	81	7	TO	6	TO	3	TO
6	10.6/10.9/82.5	F/1	16	33	1	–	–	–	–	<1	TO
		F/2	77	160	11	9	TO	6	TO	<1	TO
		F/4	354	721	63	20	TO	21	63	3	TO
7	.929/.928/.927	F/1	87	184	3	TO	TO	<1	1	<1	<1
		F/2	285	592	15	4	TO	4	20	3	22
		F/4	1017	2080	63	76	767	71	205	67	187
		F/8	3825	7744	255	TO	TO	TO	TO	TO	TO

Absence of small FSCs | We evaluate how fast pMC techniques prove the absence of a small strategy satisfying the property. In particular, we consider proving that for a specific threshold, there is no strategy with at most m memory inducing a better value than the threshold λ. Such a proof allows us to draw conclusions about the (ε-)optimality of the FSCs synthesised as above.

Results: Table 7.4a shows the run times to check that for the POMDP in column Id, there exists no strategy of type FSC with m nodes that performs better than threshold λ. The row indicated by * is obtained solving the ETR encoding, other results are obtained with parameter lifting. *Evaluation:* The methods generally prove tight bounds for $m = 1$. For $m > 1$, the high number of parameters yields a mixed impression, the performance depends on the benchmark. Still, we may find proofs for non-trivial bounds up to $m = 8$, even if the pMC has hundreds of parameters.

Table 7.3: ε-optimality and closed-form computation

(a) Proving absence

Id	FSC/m	λ	time (s)
2	F/1	5	<1
3	F/1	5	8
3	C/4	5	183
4	F/1	0.25	2*
5	F/1	10	3
5	F/2	5	TO
6	F/1	82	<1
6	F/8	10.5	1
7	F/1	0.94	5

(b) Closed-form solution

Id	FSC/m	time (s)
1	F/1	<1
1	F/2	97
2	F/1	155
3	F/1	464
4	F/1	<1
5	F/1	116
6	F/1	<1
7	F/1	TO

Closed-form solutions | Finally, we consider the computation of solution functions of the corresponding pMC using state elimination in STORM (with the default setting).

Results: Table 7.4b indicates run times to compute a solution function.

Evaluation: Closed-form computation is limited to small memory bounds. The rational functions obtained vary wildly in their structure. For benchmark (4), the result is a constant function, which is trivial to analyse, while for benchmark (3), we obtained rational functions with roughly one million terms, rendering further evaluation expensive.

Comparison to state-of-the-art | We stress that to some extent, we are comparing apples and oranges. Finding an (almost) optimal deterministic history-dependent observation-based strategy is an undecidable problem in general. However, much research has been put into developing good algorithms and engineering good tools to tackle that problem, making these tools appear more mature than parameter synthesis is. Nevertheless, these tools sometimes struggle significantly with larger POMDPs. On the other hand, parameter synthesis successfully finds strategies that might not be optimal, but still perform rather well and may be easily understood and implemented. Still, finding small strategies in POMDPs with many observations and large action spaces seems to be intractable. However, the performance and ideas of finding small strategies seem to be largely orthogonal to the performance of finding almost optimal history-dependent strategies, rising hopes for a symbiosis of both approaches.

7.2 Concurrent Stochastic Reachability Games

We want to consider the idea of using parameters to describe strategies further. In particular, we are interested in finding a natural model with two sources of nondeterminism

corresponding to pMDPs once the strategy for one source of nondeterminism is represented by parameters. Concretely, we look for an extension to both POMDPs and SGs. Partially observable SGs [HBZ04; CD14] require memory for both players, so we prefer to restrict the partial observation. Therefore, we study *concurrent stochastic reachability games* (*CSRGs*) [AHK07]. One way to interpret CSRGs is as an extension to (alternating turn-based) SGs, cf. Section 3.4, in which player I moves first, but player II cannot observe what action player I selects. The player observes all other information, in particular, the outcome of its actions. This interpretation yields a restricted class of partially observable SGs. CSRGs do not fully correspond to pMDPs, but they are an interesting subclass with open complexity results. CSRGs may also be regarded as a variant of MDPs. Instead of (one player) selecting an action at a state, two players both select an action independently and simultaneously. The combination of the actions determines the dynamics, i.e., the distributions over successor states describing how the CSRG evolves. CSRG

Example 7.7 We reconsider a variant of rock-paper-scissors, inspired by [KNPS18] and a straightforward adaption of Example 4.2. We play against Eve and both select *simultaneously* Rock (R), Paper (P) or Scissors (S). There are nine possible outcomes, all reflected by the actions in the CSRG. We are interested in minimising the probability that we ever loose a round. Luckily, we thus either (1) win by playing (draws) forever, or (2) Eve gives up if we win twice in a row. We are interested in the following questions: *What is the best strategy for us to avoid reaching the target state in the case Eve plays perfectly?*, *What is the minimal probability of reaching the target state?*, and *How can we play if we merely want to ensure we loose with a probability less than 90%?*

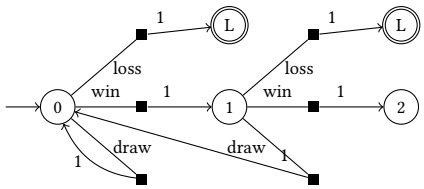
CSRGs extend matrix games [Sha53; NM47] which have been studied for decades. CSRGs are an adequate model for many applications [FV97], e.g., the synthesis problem in synchronous reactive synthesis [AHM00]. They have been studied in theory [CAH13; CHI17] and have recently been integrated in probabilistic model checkers [KNPS18].

7.2.1 What are concurrent stochastic reachability games?

We briefly formalise the main concepts.

Definition 7.9 A *concurrent stochastic reachability game* (*CSRG*) is a tuple $\mathcal{G} = (S, \iota, Act \coloneqq Act_{\mathrm{I}} \times Act_{\mathrm{II}}, P)$ with a finite set of states S, initial state ι, finitely many actions generated by the Cartesian product of player-I actions Act_{I} and player-II actions Act_{II}, and a probabilistic transition function $P\colon S \times Act \to Distr(S)$. CSRG

CSRGs syntactically are MDPs without rewards and with a special action-structure. The selected action is the outcome of the independent action choices of player I and



(a) CSRG

		player II		
		Rock	Paper	Scissors
player I	Rock	0	$i+1$	L
	Paper	L	0	$i+1$
	Scissors	$i+1$	L	0

(b) (determinstic) outcomes in state i

Figure 7.8: (a) The CSRG from our running example. Winning/loosing/draws are from the perspective of player II, and each reflects three actions in the CSRG, as in the table in (b).

player II. The objective of player I (aka: *reachability* player) is to reach a target state in $T \subseteq S$ while player II (aka: *safety player*) has to avoid ever reaching a state in T.

·play A CSRG is a two-player game $\mathcal{G}$ played on a finite set S of states. A *play* of $\mathcal{G}$ begins in the initial state ι and proceeds as follows: In state s, both players I and II *concurrently* select an action $\alpha_{\mathrm{I}} \in Act_{\mathrm{I}}$ (resp. $\alpha_{\mathrm{II}} \in Act_{\mathrm{II}}$). The game then picks a successor state s' according to a fixed probability distribution $P(s, \langle \alpha_{\mathrm{I}}, \alpha_{\mathrm{II}} \rangle)$ over S, and the play continues in s'. The transition from s to s' is called a *round* of $\mathcal{G}$. Player I wins $\mathcal{G}$ once a state in T is reached. Otherwise, if a target is never reached, then II wins.

·round

·strategy

Definition 7.10 (CSRG Strategy) Let $i \in \{\mathrm{I}, \mathrm{II}\}$ be a player of a CSRG. A player-i *strategy* $\sigma_i \colon \Pi^{\mathcal{G}}_{\mathsf{fin}} \to Distr(Act_i)$ selects actions for player i.

Deterministic and memoryless player-i strategies are defined analogously to MDP strategies. Let $\mathfrak{S}^{i,\mathcal{G}}_m$ denote the set of memoryless strategies for player $i \in \{\mathrm{I}, \mathrm{II}\}$.

Example 7.8 In Figure 7.8 we define the CSRG for rock-paper-scissors, as outlined in Example 7.7. The nine actions may be partitioned into three sets. The target states are those in which we, player II, have lost a round (duplicated to avoid clutter). Otherwise, if we win, we move to the next round, and if we draw, we restart the game. In the second round, we may choose a different distribution over the actions. After two wins, we reach a sink state.

·induced MDP The *induced MDP* $\mathcal{G}[\sigma_{\mathrm{I}}]$ of CSRG $\mathcal{G}$ with a memoryless strategy $\sigma_{\mathrm{I}} \in \mathfrak{S}^{i,\mathcal{G}}_m$ is a finite MDP M with probability transition function P' with

$$P'(s, \beta, s') := \sum_{\alpha \in \mathbb{A}\mathrm{ct}_{\mathrm{I}}(s)} \sigma_{\mathrm{I}}(s, \alpha) \cdot P'(s, \langle \alpha, \beta \rangle, s')$$

for all $s, s' \in S$ and actions $\beta \in \mathbb{A}\mathrm{ct}_{\mathrm{II}}$ of player II. (Induced MDPs are defined completely symmetrically for strategies of player II.) Conversely, every MDP may be viewed as a

CSRG where $|\mathbb{A}\mathrm{ct}_{\mathrm{I}}(s)| = 1$ (or $|\mathbb{A}\mathrm{ct}_{\mathrm{II}}(s)| = 1$) for all $s \in S$, i.e. an MDP is a CSRG where one of the players never has any choice.

Let $\Pr_{\mathcal{G}}^{\sigma,\tau}(\Diamond T)$ be the probability that T is reached if player I plays according to σ and player II according to τ. The *value* of $\mathcal{G}$ is defined as follows: The player-I-value (resp. player-II-value) is defined as ·value

$$V_{\mathrm{I}}(\mathcal{G}) := \sup_{\sigma} \inf_{\tau} Pr_{\mathcal{G}}^{\sigma,\tau}(\Diamond T), \quad \text{and} \quad V_{\mathrm{II}}(\mathcal{G}) := 1 - \inf_{\tau} \sup_{\sigma} Pr_{\mathcal{G}}^{\sigma,\tau}(\Diamond T),$$

where the sup and inf range over all strategies of both players, respectively. Intuitively, it is the maximal winning probability of player I that may be guaranteed against all strategies of player II.

Lemma 7.10 ([Par71; HPRV76]) CSRGs enjoy the following properties:

- It holds that
$$\sup_{\sigma} \inf_{\tau} Pr_{\mathcal{G}}^{\sigma,\tau}(\Diamond T) = \inf_{\tau} \sup_{\sigma} Pr_{\mathcal{G}}^{\sigma,\tau}(\Diamond T),$$
and thus $V_{\mathrm{I}}(\mathcal{G}) + V_{\mathrm{II}}(\mathcal{G}) = 1$ (i.e. CSRGs are determined).
- There is a memoryless strategy τ^* for II such that
$$\inf_{\tau} \sup_{\sigma} Pr_{\mathcal{G}}^{\sigma,\tau}(\Diamond T) = \min_{\tau \in \mathfrak{S}_m^{\mathrm{II},\mathcal{G}}} \max_{\sigma \in \mathfrak{S}_m^{\mathrm{I},\mathcal{G}}} Pr_{\mathcal{G}}^{\sigma,\tau}(\Diamond T) = \max_{\sigma \in \mathfrak{S}_m^{\mathrm{I},\mathcal{G}}} Pr_{\mathcal{G}}^{\sigma,\tau^*}(\Diamond T).$$

7.2.2 Relation to parameter synthesis

After fixing one strategy, the CSRG reduces to its induced MDP. The challenge for the player II is that they do not know the strategy of player I. Much alike constructing pMCs corresponding to POMDPs, we now represent memoryless strategies for player II by parameters, and obtain a pMDP (which coincides with the pMDP from Example 4.2.)

Example 7.9 We construct the corresponding pMDP of the CSRG from Figure 7.8 in Figure 7.9. The actions refer to player I moves, the parameters to the probability that the player II strategy chooses a particular action. For example, x_S refers to the probability that player II picks scissors. Thus, if we follow the action α_R and then the transition for scissors, we immediately loose, in line with the CSRG. Indeed, there are nine outgoing transitions from the initial state, corresponding to the nine action combinations.

The existence of stationary optimal strategies for player II allows us to always reduce a CSRG into a pMDP. Please bear in mind that we do not have a result allowing to encode pMDPs into CSRGs.

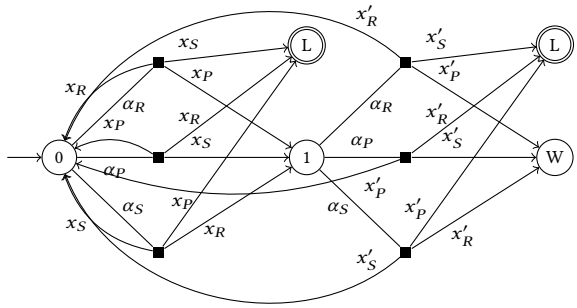


Figure 7.9: The pMDP corresponding to the CSRG from Figure 7.8.

·corresponding pMDP

Definition 7.11 (Corresponding pMDP) Let $\mathcal{G} = (S, \iota, Act_{\mathrm{I}} \times Act_{\mathrm{II}}, P)$ be a CSRG. We define the *corresponding pMDP* $\mathcal{M}_{\mathcal{G}} := (S, \iota, Act_{\mathrm{I}}, X, \mathcal{P})$ with $X := S \times Act_{\mathrm{II}}$ and

$$\mathcal{P}(s, \alpha, s') := \sum_{\beta \in Act_{\mathrm{II}}} P(s, \langle \alpha, \beta \rangle, s') \cdot x_{\langle s, \beta \rangle},$$

where $x_{\langle s, \beta \rangle} \in X$.

The intuition is that there is a one-to-one correspondence between memoryless strategies τ for player II in $\mathcal{G}$ and parameter valuations $\mathsf{val} \in \mathsf{Val}^{\mathrm{wd}}_{\mathcal{M}}$: A parameter $x_{\langle s, \beta \rangle} \in X$ corresponds to a player-II action $\beta \in \mathbb{A}\mathrm{ct}_{\mathrm{II}}(s)$ for some $s \in S$. Using τ, we may define the valuation $\mathsf{val} := \mathsf{enc}(\tau) \in \mathsf{Val}^{\mathrm{wd}}_{\mathcal{M}}$ with enc such that $\mathsf{val}(x_{\langle s, \beta \rangle}) = \tau(s)(\beta)$. This construction is invertible. Formally, we obtain:

Theorem 7.11 For any given CSRG $\mathcal{G}$, there exists a corresponding simplex pMDP $\mathcal{M}_{\mathcal{G}}$ such that

$$V_{\mathrm{I}}(\mathcal{G}) = \min_{\tau \in \Sigma^{\mathrm{II}}} \max_{\sigma \in \Sigma^{\mathrm{I}}} Pr^{\sigma,\tau}_{\mathcal{G}}(\Diamond T) = \min_{M \in \langle \mathcal{M} \rangle} \max_{\sigma \in \Sigma^{M}} Pr^{\sigma}_{M}(\Diamond T)$$

and $\mathcal{M}_{\mathcal{G}}$ may be computed in polynomial time (in the size of $\mathcal{G}$).

The proof follows the construction in Section 7.1 and is omitted here. For a full proof, we refer to the technical report of [19], which uses additional auxiliary states to obtain a simplex pMDP.

7.3 Sketching and Software Product Lines

The idea of program *sketching* [Sol13] is to start with a program sketch, a partial program in which difficult expressions, guards, and statements are left unspecified. The hypothesis of program sketching is that programmers often have an idea about the main control flow of the program but filling in all low-level details is laborious and error prone. Completing these low-level details is left to an automated synthesiser. The synthesised program has to satisfy the specification Φ. sketching

A software product line (*SPL*) is (according to Wikipedia[1]) "a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way." Products in a software product line have different features which may be understood as functionalities changing the behaviours of a core software system. They thus provide an elegant way to specify families of systems: every member of the family comprises the core system and a combination of features. SPL

Both sketching and SPLs consider a program with some behaviour unspecified: In sketching, the goal is to concretise the unspecified behaviour such that the overall system satisfies a specification, in software product lines, the goal is to classify which of the available behaviours yield a system that satisfies the specification. It is thus natural to consider these two areas together and to consider a sketch as the description of a software product line. We have already seen a technical view of sketches in Section 6.2. Here, we focus on the more practical side of using sketches to solve problems.

Related work | Sketching probabilistic programs to match available data has been considered in [NORV15]. The goals and methods are significantly different from our notion of sketches. Sketching with fixed topology but parametric transitions has been considered in [GTC15]. The quantitative analysis of software product lines has been studied extensively. In [GS13; RANL+15], standard pMCs are used for this analysis. The downside is that features are discrete choices, whereas standard pMC analysis assumes continuous parameters. Moreover, most tools for pMC analysis do not soundly support changing the underlying graph in pMC, which is important for these applications. In [VK13], model checking families for full PCTL is introduced in the context of software product lines. These families are much alike pMCs. Tool support in [CDKB18; VBLL18] focusses on modelling aspects and uses a probabilistic model checker or a discrete event simulator as a backend. Dynamic SPLs extend SPLs with configuration changes during runtime, and probabilistic model checking is considered in, e.g., [DKB14].

[1] https://en.wikipedia.org/wiki/Software_product_line

7.3.1 Illustrative challenges

We show how parameter synthesis may be applied to sketching.

7.3.1.1 A sketching challenge

Many sketching challenges that are interesting in combination with model checking originate from some form of controller synthesis: Often, the sketches describe a partial strategy for POMDPs, decentralised MDPs, or other extensions to MDPs with restricted classes of strategies.

Concrete challenge | Our program sketch is describing a *dynamic power manager (DPM)*, a key component in dynamically optimising energy consumption [BBPM99]. A DPM controls changing the system's power states at run time. Depending on workload and performance constraints, it issues commands (e.g., go into sleep mode, wake up) to the system. We consider a DPM system with two request priorities, low and high; these priorities typically depend on the time-criticality. Requests are placed in finite buffers (based on their priority), provided the buffers are not full. Otherwise, the requests are lost. A similar DPM has been analysed by probabilistic model checking [SDM08].

Problem statement | Our goal is to *synthesise a DPM program* that decides to switch power state based on the current workload expressed in terms of the occupancy of the low-priority and high-priority request buffers. It should not depend on other information. We are interested in finding a DPM program that satisfies a specification (inspired by [GTC15]) that constrains the expected number of lost low- and high-priority requests, and the expected energy consumption.

Approach | The starting point is a program sketch that includes partially specified commands of the form:

$$g_H \ \& \ g_L \quad \rightarrow \quad 1: \ \mathtt{pm}' = y$$

where g_H and g_L are guards concerning the low-priority and high-priority request buffer, respectively, and hole y represents an unknown update of the DPM state. We clarify the structure in Figure 7.10. Every time the program comes to a point where a sched command is executed in the DPM module, the current state of the two queues is inspected, and based on (only) the value for these queues and the corresponding hole, the power mode is selected. We synthesise a realisation such that the resulting DPM control program meets the specification outlined above.

```
const int ACTIVE_HIGH = 0; const int ACTIVE_LOW = 1; const int SLEEP = 3, const int OFF = 4;
hole pm11 either { ACTIVE_HIGH, ACTIVE_LOW, SLEEP, OFF }
hole pm12 either { ACTIVE_HIGH, ACTIVE_LOW, SLEEP, OFF }
...
hole pm99 either { ACTIVE_HIGH, ACTIVE_LOW, SLEEP, OFF }
module DPM
pm : [0..1] init 0;
[sched] qH = 1 & qL = 1 -> 1: pm'=pm11;
[sched] qH = 1 & qL = 2 -> 1: pm'=pm12;
..
[sched] qH = 9 & qL = 9 -> 1: pm'=pm99;
endmodule
module SYSTEM
qH : [1..10] init 1;
qL : [1..10] init 1;
...
endmodule
...
```

Figure 7.10: Part of the DPM Sketch

Results | The MC family has over $3 \cdot 10^5$ realisations, i.e., control programs. The average realisation has more than 5000 states. We consider an unsatisfiable conjunction of 3 properties describing a possible DPM specification. Within 20 minutes, the conjunction is shown to be unsatisfiable (although each property alone is satisfiable)—using a technique outlined in Chapter 10. An enumerative approach takes more than 20 hours to show this. For a satisfiable conjunction, we find a realisation within minutes.

7.3.1.2 A software product line challenge

For software product lines that are tightly integrated with their environment, quantifying (and minimising) the probability of encountering an anomaly or unexpected behaviour becomes crucial, just as quantifying the energy consumption of battery-based (software) systems. This insight has led to a growing interest in model checking probabilistic models in the software engineering community [Ros16].

Concrete challenge. | We consider the BSN (Body Sensor Network) software product line benchmark from [RANL+15], the largest benchmark analysed by probabilistic model checking with the ProFeat tool [CDKB18]. BSN describes a network of connected sensors that send measurements to a unit identifying health-critical situations. The family contains the various configurations of 10 binary features, that is, whether a sensor is available or not. We are interested in the reliability of the system, that is, in the probability that the system behaves as described.

Problem statement. | Our goal is to find all feature combinations where the induced system does not meet a certain reliability.

```
const double pFA;
const double pFB;
module encode
s : [0..1] init 0;
FA : [0..1] init 0;
FB : [0..1] init 0;
s = 0 -> pFA  * pFB: s'=1 & FA'=1 & FB'=1  +
         (1-pFA)  * pFB: s'=1 & FA'=0 & FB'=1 +
         pFA  * (1-pFB): s'=1 & FA'=1 & FB'=0  +
         (1-pFA)  * (1-pFB): s'=1 & FA'=0 & FB'=0;
..
endmodule
```

(a) A pMC description with features.

```
hole fa either { 0, 1 }
hole fb either { 0, 1 }
module encode
s : [0..1] init 0;
FA : [0..1] init 0;
FB : [0..1] init 0;
s = 0 -> 1: s'=1 & FA'=fa & FB'=fb;
..
endmodule
```

(b) A sketch depending on features.

Figure 7.11: Describing a SPL as pMC or sketch.

Approach. | The formulation in [RANL+15] yields a pMC where each feature is captured by a parameter, but we use a sketch as a more concise representation, in a similar fashion as outlined by the example below.

> **Example 7.10** We consider a variation point (a state whose future behaviour depends on the features) where depending on the availability of features F_a, F_b the model behaves differently. For each feature, a Boolean parameter f is 1 if the feature is active and 0 otherwise. At a variation point, the probability of every transition is scaled by factor p, which equals f if the feature enables the transition and $1-f$ otherwise. This results in parametric MC encoded in Figure 7.11a. We use a sketch as in Figure 7.11b.

Furthermore, in a sketch, additional dependencies between features may naturally be expressed by constraints. Such a dependency may, e.g., state that there is no product having both the large battery and the thin cover. We partition the realisations in realisations that accept and reject the specification.

Results. | Although this case study is the largest product line example used in [CDKB18], verifying a family with just 1024 family members, with an average size of the realisation of roughly 100 states is mostly trivial. Within seconds, we can categorise the different realisations based on their reliability, either by our approaches in Chapter 11 or by enumeration. Experiments in Chapter 11 show that for larger sketches in a similar spirit, parameter synthesis methods scale better than naive enumeration scales.

```
hole N either {5,8,10};
module transmission
c : [0..10] init 0;
s : [0..10] init 0;
// loop head
s=0 & c<N -> 1: s'=s+1 & c'=c+1;
s=0 & c>=N -> 1: s's=8;
..
s=7 -> s'=0;
// end of loop
..
endmodule
```

Figure 7.12: Holes in guards of a sketch

7.3.2 User-centric improvements

To use sketching as above, we currently rely on experienced users of modelling for probabilistic model checkers. To really use our approaches, some usability improvements are necessary, and we survey some ideas that we deem essential to support more flexible sketching. Below, we discuss some aspects of a conservative extension that may be combined with the methods in Chapters 10 and 11.

7.3.2.1 Extensions to sketches

We already mentioned that PRISM sketches come with support for multiple parallel modules, just as in the regular PRISM language. Holes may occur in multiple modules without particular restrictions. Below, we discuss the occurrence of holes in guards and the use of different hole types.

Holes in guards | Holes in guards occur naturally in product lines and sketching.

> **Example 7.11** Consider a communication protocol which aborts a packet transmission after N lost packets, where N is a parameter affecting the topology, or consider a redundant hardware circuit, which only fails if N out of M circuits have failed. Such a system may be described by the sketch in Figure 7.12. Closer to the program sketching view, guards typically reflect loop conditions, and allowing holes in guards allows to synthesise an upper bound on, e.g., the number of retries that may be performed.

Special care is required to avoid deadlocks and/or overlapping guards if these guards contain holes: If the guards occurs in pairs `g`, `!g`, we do not obtain overlapping guards or deadlocks—without these pairs, sketches are prone to contain these shortcomings. We discuss how to handle them in Section 7.3.2.2.

```
hole p either { 0.2, 0.7, 0.9 }
module encode
s : [0..10] init 0;
s = 0 -> p: s'=1 & 1-p: s'=2;
..
endmodule
```

(a) Proposed extension to sketch

```
hole p either { 1, 2, 3 }
module encode
s : [0..10] init 0;
aux : [0..3] init 0
s=0 & aux=0 -> aux' = p;
s=0 & aux=1 -> 0.2: s'=1 & aux'=0 + 0.8: s'=2 & aux'=0;
s=0 & aux=2 -> 0.7: s'=1 & aux'=0 + 0.3: s'=2 & aux'=0;
s=0 & aux=3 -> 0.9: s'=1 & aux'=0 + 0.1: s'=2 & aux'=0;
...
endmodule
```

(b) Equivalent standard sketch

Figure 7.13: Holes describing a selection of probabilities

```
hole g either { x < 3,  x < 4 }
module encode
x : [0..10] init 0;
s : [0..10] init 0;
s = 0 -> 1: s' = 1;
s = 1 & g : 1: s'=2;
s = 1 & !g : 1: s'=3;
...
endmodule
```

(a) Proposed extension to sketch

```
hole g either { x < 3,  x < 4 }
module encode
x : [0..10] init 0;
s : [0..10] init 0;
aux : bool init false;
s = 0 -> 1: s' = 1 & aux'=g=1? x<3 : x<4;
s = 1 & aux : 1: s'=2;
s = 1 & !aux : 1: s'=3;
...
endmodule
```

(b) Equivalent standard sketch

Figure 7.14: Supporting holes in guards

Different hole types | We may introduce various different hole types as syntactic sugar, which we exemplify for (a finite set of) probabilities as options in Figure 7.13. We may transform this into an equivalent sketch. We store in an auxiliary variable which hole value should be used and then behave accordingly in the next step. A natural downside is that the information that these probabilities preserve the topology is lost, as after the transformation the precise topology is no longer preserved.

Alternatively, we may use these ideas to allow holes for Boolean expressions. As example, consider the synthesis of a loop condition in a program without using holes in guards, cf. Figure 7.14. We already discussed that, in general occurrence of holes in guards may yield side-effects such as deadlocks.

Similar extensions as those outlined above may be made for changing the left-hand sides of assignments and for more complex expressions, see Figure 7.15. All proposed extensions as syntactic sugar increase the size of the underlying MCs. The underlying solver remains oblivious of the original meaning of the holes, which prevents the solver from making tailored decisions. Therefore, a tighter integration of the methods with the sketch extensions seems essential once more interesting sketches are available.

```
hole v either { x, y }
module encode
x : [0..10] init 0;
y : [0..10] init 0;
x<8 & y < 5 -> 0.33: v' =0 + 0.67: x'=y;
...
endmodule
```

(a) Proposed extension to sketch

```
hole v either { s+1, s-1 }
module encode
s : [0..10] init 0;
s < 8 -> 0.5: s'=v & 0.5: s'=2;
..
endmodule
```

(b) Proposed extension to sketch

Figure 7.15: Further extensions to sketches

7.3.2.2 Well-formed sketches

Not every syntactic sketch describes valid programs after instantiating them with a realisation, an issue that we also saw for pMDPs, cf. well-defined valuations in Definition 4.3 on page 65. Already (hole-free) PRISM programs may be illformed: The most important instances of ill-formed programs are programs with

- a deadlock in the model, i.e., programs in which in a (reachable) state all guards evaluate to false,
- variables that go out of their bounds,
- guards that overlap[1].

Such errors are typically considered modelling errors and most model checkers may not even properly check for such errors as these checks may be computationally intricate. When analysing, such occurrences are assumed to be absent, i.e., the analysis results of such programs is mostly unspecified.

While it is convenient to adopt such a policy for sketches, the existence of such flaws is much more likely as we are in fact describing many programs using a single sketch. Furthermore, while modelling concise sketches with small underlying MCs, it may be hard to avoid including realisations that do not describe valid programs. It is not helpful to resort to the extensive use of constraints. The rigour of such a policy is also unnecessary, especially when the sketch is used to search for a feasible realisation. Instead, it is good to handle sketches that include ill-formed realisations. However, properly handling sketches which include ill-formed realisations requires checking the well-formedness of every realisation and may significantly degrade the overall performance. Currently, we rigorously consider the presence of ill-formed programs a modelling error in the method from Chapter 11, while the method from Chapter 10 includes optimisations to efficiently deal with the existence of flawed realisations.

[1]There are two semantics here. We consider the natural semantics that overlapping guards induce nondeterminism—which is an issue if we are modelling MCs.

We have seen how finding small memory strategies for POMDPs and other extensions of MDPs is (a special case of) a parameter synthesis problem. Furthermore, we have seen that the synthesis and verification of many of these problems may be expressed using sketches, which in turn may be analysed with typical parameter synthesis approaches. We concluded this overview with some practical aspects regarding the expressivity of probabilistic sketches. The listed applications are only a selection. We refer to the introduction for further pointers to successful applications of parameter synthesis.

Part III

Inside Parameter Synthesis

8 Basic Methods for Parameter Synthesis

Essential techniques for synthesis

Summary | We discuss well-known approaches to parameter synthesis in five parts. First, we show how to compute a solution function and present some variants. Second, we discuss some practical aspects of applying the ETR encoding for automated feasibility and validity checking. Third, we discuss sampling-based approaches to feasibility, and fourth, we discuss how to utilise validity and feasibility checkers for parameter space partitioning. Finally, we review these methods and discuss improvements made in further chapters.

Origins | The section on computing the solution function resembles the corresponding part in [17], but includes a treatment of a fraction-free method, which is an extension to its presentation in [20]. The section on the ETR encoding is based on [17], but in its presentation mostly novel. The section on the sampling is novel in its presentation. The section on parameter space partitioning strongly resembles [17]. The bird's view is original.

Background | We assume familiarity with the basics of pMCs and pMDPs, and partially with the ETR encoding for feasibility, all of which is covered in Chapter 4.

8.1 Computing the Solution Function

We consider the computation of solution functions as defined in Definition 4.7. This function is a rational function if one considers (cyclic) pMCs and graph-preserving valuations, cf. Theorem 4.21, and a polynomial if one considers acyclic pMCs and well-defined valuations, cf. Corollary 5.19. In this section, we consider pMCs and assume graph-preserving valuations. We do not discuss (trivial) extensions to (exponential) algorithms for MDPs and well-defined valuations on cyclic models.

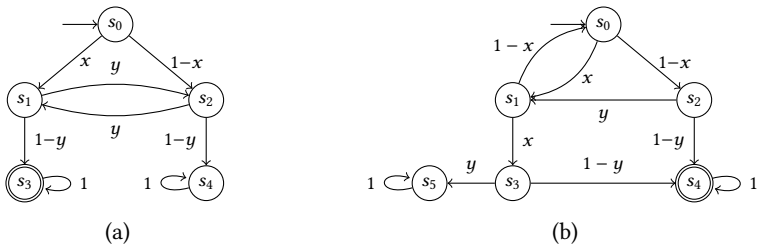


Figure 8.1: Two running examples

8.1.1 Gaussian elimination

Recall from Section 4.3.4 that the solution function is the solution of a linear equation system with rational functions as coefficients. Classically, solving such systems may be done with Gaussian elimination. We start with a running example and then provide a variant to Gaussian elimination whose run time is exponential in the number of parameters, but polynomial in the number of states.

Example 8.1 Consider the equations for the pMC from Figure 8.1a.

$$\begin{aligned} p_0 &\doteq x \cdot p_1 + (1-x) \cdot p_2, & p_3 &\doteq 1, \\ p_1 &\doteq y \cdot p_2 + (1-y) \cdot p_3, & p_4 &\doteq 0, \\ p_2 &\doteq y \cdot p_1 + (1-y) \cdot p_4. \end{aligned}$$

Bringing the system in matrix normal form yields:

$$\begin{aligned} p_0 - x \cdot p_1 - (1-x) \cdot p_2 \qquad\qquad\qquad\qquad &\doteq 0 \\ p_1 \qquad - y \cdot p_2 - (1-y) \cdot p_3 \qquad\qquad &\doteq 0 \\ -y \cdot p_1 \qquad + p_2 \qquad\qquad - (1-y) \cdot p_4 &\doteq 0 \\ p_3 \qquad\qquad &\doteq 1 \\ p_4 &\doteq 0. \end{aligned}$$

Adding y times the second equation to the third equation yields an upper triangular matrix. From here onwards, we omit the last two equations for conciseness.

$$\begin{aligned} p_0 - x \cdot p_1 - (1-x) \cdot p_2 \qquad\qquad\qquad\qquad &\doteq 0 \\ p_1 \qquad - y \cdot p_2 \; - (1-y) \cdot p_3 \qquad\qquad &\doteq 0 \\ (1-y^2) \cdot p_2 - y(1-y) \cdot p_3 - (1-y) \cdot p_4 &\doteq 0 \end{aligned}$$

Observe that the triangular matrix is fraction-free, i.e., all entries are polynomials. We perform back-substitution (first for p_3, p_4):

$$p_0 - x \cdot p_1 - (1-x) \cdot p_2 \doteq 0, \quad p_1 - y \cdot p_2 \doteq 1 - y, \quad (1-y^2) \cdot p_2 \doteq y(1-y),$$

and then for p_2:

$$p_0 - x \cdot p_1 \doteq \frac{(1-x)(y-y^2)}{1-y^2}, \quad p_1 \doteq 1 - y + \frac{y^2(1-y)}{1-y^2}, \quad p_2 \doteq \frac{y(1-y)}{1-y^2},$$

Finally, we obtain for p_0:

$$p_0 \doteq x\big(1 - y + \frac{y^2(1-y)}{1-y^2}\big) + \frac{(1-x)(y-y^2)}{1-y^2} \doteq \frac{x(1-y)(1+y) + xy^2 + (1-x)y}{1+y},$$

which simplifies to

$$\frac{x + y - xy}{1+y}.$$

We witness an extensive growth of the coefficients—i.e., the matrix entries—during back-substitution. The entries may grow even larger in the presence of fractions. Fractions may also occur during triangulation, which then propagates forward:

Example 8.2 Consider the equations for the pMC from Figure 8.1b:

$$\begin{array}{rrrrrrrl}
p_0 & -x \cdot p_1 & -(1-x) \cdot p_2 & & & & & \doteq 0 \\
-(1-x) \cdot p_0 & +p_1 & & -x \cdot p_3 & & & & \doteq 0 \\
& -y \cdot p_1 & +p_2 & & -(1-y) \cdot p_4 & & & \doteq 0 \\
& & & p_3 & -(1-y) \cdot p_4 & -y \cdot p_5 & & \doteq 0 \\
& & & & & p_4 & & \doteq 1 \\
& & & & & & p_5 & \doteq 0
\end{array}$$

We first add $1 - x$ times the first row to the second row, which becomes:

$$+(1-(1-x)x) \cdot p_1 - (1-x)^2 \cdot p_2 - x \cdot p_3 \doteq 0$$

Now, we add $\frac{y}{1-(1-x)x}$ times p_1 to the third row, which becomes:

$$\Big(\frac{y}{1-(1-x)x}(1-x)^2 + 1\Big) \cdot p_2 - \frac{xy}{1-(1-x)x} \cdot p_3 - (1-y) \cdot p_4 \doteq 0$$

This finishes the triangulation. The back-substitution spreads the large coefficients. Alternatively, in larger models, the triangulation would now spread the large coefficients forward. We omit the remaining steps for conciseness.

Recall that the elementary operation in the m-th triangulation step of Gaussian elimination is:

$$a_{i,j}^{(m)} := a_{i,j}^{(m-1)} - \frac{a_{i,m}^{(m-1)} a_{m,j}^{(m-1)}}{a_{m,m}^{(m-1)}}.$$

In the worst case, the (total) degree of the factors is doubled in every step of the triangulation. When doubling the degree in each step, the degree grows exponentially. The growth of fractions may often be limited by eliminating common factors, i.e., by keeping the numerator and the denominator coprime. This operation requires the computation of the greatest common divisor (GCD), which is an expensive operation. In [JCVW+14], data structures to avoid the computations of GCDs are introduced. Another option is to avoid the occurrence of fractions altogether.

·fraction-free

Fraction-free Gaussian elimination | *Fraction-free* Gaussian elimination is a variant of Gaussian elimination that avoids a fractional representation of the intermediate matrix values during matrix triangulation. For example, when starting with an integer matrix it ensures that the intermediate values are integers as well. Now, when using (naïve) fraction-free Gaussian elimination the new coefficients after the m-th step, $m = 1, \ldots, n-1$, are computed as

$$a_{i,j}^{(m)} := a_{i,j}^{(m-1)} a_{m,m}^{(m-1)} - a_{i,m}^{(m-1)} a_{m,j}^{(m-1)}$$

for $i, j = m+1, \ldots, n$, where $a_{i,j}^{(0)} = a_{i,j}$. The b_i are updated analogously. We observe that the degree still grows exponentially.

·one-step fraction-free

One step fraction-free Gaussian elimination | In *one-step fraction-free* Gaussian elimination [Bar72] (see Algorithm 1), the computation in step m changes to

$$a_{i,j}^{(m)} := \left(a_{i,j}^{(m-1)} a_{m,m}^{(m-1)} - a_{i,m}^{(m-1)} a_{m,j}^{(m-1)} \right) / a_{m-1,m-1}^{(m-1)}$$

with $a_{0,0}^{(0)} = 1$, analogously for the b_i. Using Sylvester's identity one may prove that $a_{i,j}^{(m)}$ is again a polynomial and that $a_{m-1,m-1}^{(m-1)}$ is in general the maximal possible divisor of $a_{i,j}^{(m-1)} a_{m,m}^{(m-1)} - a_{i,m}^{(m-1)} a_{m,j}^{(m-1)}$. Here, the application of division limits the growth of the polynomials and, as an exact divisor is known by construction, the costly computation of the GCD is avoided. We repeat our two examples. The first is calculation-wise simpler, the second example is challenging but is better suited to observe the differences from standard Gaussian elimination.

Algorithm 1 One-step fraction-free Gaussian elimination [Bar72]

```
function OneStepFFGauss(A = (a_ij)_{i,j=1,...,n}, b = (b_i)_{i=1,...,n})
    a_{0,0} ← 1
    for m = 1, ..., n−1 do                                  ▷ triangulation, assuming a_{m,m} ≠ 0
        for i ← m+1, ..., n do
            for j ← m+1, ..., n do
                a_{i,j} ← (a_{m,m} · a_{i,j} − a_{i,m} · a_{m,j}) / a_{m−1,m−1}      ▷ exploit exact divisibility
            b_i ← (a_{m,m} · b_i − a_{i,m} · b_m) / a_{m−1,m−1}                      ▷ exploit exact divisibility
            a_{i,m} ← 0
    for m ← n−1, ..., 1 do                                  ▷ back substitution
        b_m ← (a_{n,n} · b_m − Σ_{i=m+1}^{n} a_{m,i} · b_i) / a_{m,m}                ▷ exploit exact divisibility
    return (b_i / a_{n,n})_{i=1,...,n}                      ▷ rational solution functions
```

Example 8.3 We start with the equation system as outlined in Example 8.1. In the first two iterations, nothing changes, as the first column has zero entries everywhere but in the first entry. The second iteration replicates the results from standard Gaussian. So we look at the iteration $m = 3$ and obtain (the slightly odd):

$$\begin{aligned}
p_0 - x \cdot p_1 - (1-x) \cdot p_2 &\doteq 0 \\
p_1 \qquad - y \cdot p_2 \; - (1-y) \cdot p_3 &\doteq 0 \\
(1-y^2) \cdot p_2 - y(1-y) \cdot p_3 - (1-y) \cdot p_4 &\doteq 0 \\
(1-y^2) p_3 &\doteq (1-y^2) \\
(1-y^2) p_4 &\doteq 0.
\end{aligned}$$

Applying step $m = 4$ does not change this result. We now do the alternative back-substitution and start with $m = 4$:

$$b_4 := 1 - y^2, \quad b_3 := y - y^2, \quad b_2 := 1 - y, \quad b_1 := xy^2 - 2xy + x - y^2 + y,$$

Which leads to the solutions:

$$p_0 := \frac{xy^2 - 2xy + x - y^2 + y}{1 - y^2} = \frac{x + y - xy}{y + 1}, \quad p_1 := \frac{1 - y}{1 - y^2} = \frac{1}{y + 1},$$
$$p_2 := \frac{y - y^2}{1 - y^2} = \frac{y}{y + 1}, \quad p_3 := \frac{1 - y^2}{1 - y^2} = 1$$

Notice that the numerator and denominator in the solutions may not be coprime.

Example 8.4 We start with the equations as in Example 8.2. The first iteration is as in standard Gaussian and thus we obtain the following at the start of $m = 2$.

$$\begin{array}{lllllll}
p_0 & & -x \cdot p_1 & -(1-x) \cdot p_2 & & & \doteq 0 \\
 & +(1-(1-x)x) \cdot p_1 & -(1-x)^2 \cdot p_2 & -x \cdot p_3 & & & \doteq 0 \\
 & & -y \cdot p_1 & +p_2 & -(1-y) \cdot p_4 & & \doteq 0 \\
 & & & & p_3 & -(1-y) \cdot p_4 - y \cdot p_5 & \doteq 0 \\
 & & & & & p_4 & \doteq 1 \\
 & & & & & p_5 & \doteq 0
\end{array}$$

For $i = 3$, we update the third row to

$$(1-(1-x)x-(1-x)^2y) \cdot p_2 - xy \cdot p_3 - (1-y)(1-(1-x)x) \cdot p_4 \doteq 0,$$

for $i = 4$, we upgrade the fourth row to

$$(1-(1-x)x) \cdot \Big(p_3 - (1-y) \cdot p_4 - y \cdot p_5\Big) \doteq 0,$$

for $i = 5$ and $i = 6$, we upgrade the last rows to

$$1-(1-x)x \cdot p_4 \doteq 1-(1-x)x \quad \text{and} \quad 1-(1-x)x \cdot p_5 \doteq 0.$$

For $m = 3$, we obtain $a_{m,m} := 1-(1-x)x-(1-x)^2y$ and $a_{m-1,m-1} \doteq 1-(1-x)x$. We first update $i = 4$:

$$(1-(1-x)x-(1-x)^2y) \cdot \Big(p_3 - (1-y) \cdot p_4 - y \cdot p_5\Big) \doteq 0$$

and for $i = 5$, $i = 6$:

$$(1-(1-x)x-(1-x)^2y) \cdot p_4 \doteq 1-(1-x)x-(1-x)^2y \quad \text{and}$$
$$(1-(1-x)x-(1-x)^2y) \cdot p_5 \doteq 0$$

In the remaining iterations, the equations remain unchanged. We then start the back-substitution as before (detailed steps are too long):

$$b_5 := (1-(1-x)x-(1-x)^2y), \quad b_4 := (1-y)(1-(1-x)x-(1-x)^2y),$$
$$b_3 := xy(1-y) + (1-(1-x)x)(1-y), \quad b_2 := ((1-x)^2(1-y) + x - xy),$$
$$b_1 := x \cdot b_2 + (1-x) \cdot b_3$$

The solutions may be read from these values.

Remark 8.1 We used a partial factorisation to simplify the presentation, but this factorisation is not adequately representing what an implementation does.

Proposition 8.1 For a polynomial pMC, Algorithm 1 computes the solution function in $\mathcal{O}\big(\mathsf{poly}(n,d)^k\big)$, where n is the number of states, d the maximal degree in the polynomials for the transition probabilities, and k the number of parameters.

Proof. If the maximal degree of the initial coefficients of A and b is d, Algorithm 1 guarantees that after m steps the degree of the coefficients is at most $(m{+}1) \cdot d$, i. e., it grows linearly in d during the procedure. For polynomials, the division by $a^{(m-1)}_{m-1,m-1}$ may be done using standard polynomial division. The time-complexity of the exact multivariate polynomial division in this case is in each step $\mathcal{O}\big(\mathsf{poly}(m,d)^k\big)$, so for the full one-step fraction-free Gaussian elimination it is $\mathcal{O}\big(\mathsf{poly}(n,d)^k\big)$. ■

The algorithm thus has a polynomial run time as it keeps the coefficients polynomially large. We stress that polynomial does not necessarily mean small, as the examples show. Some experiments, e.g., in [20], show that fraction-free elimination seems beneficial in dense pMCs.

8.1.2 State elimination

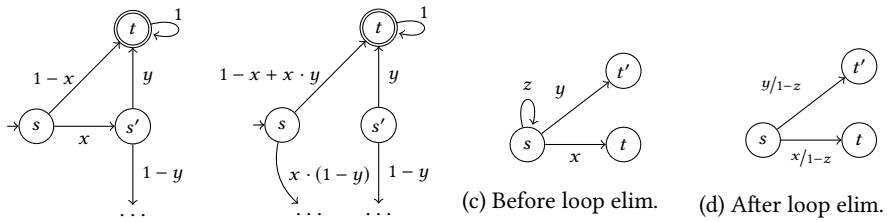
The original approach to compute the solution function of pMCs is via *state elimination* [Daw04; HHZ11b] and is analogous to the computation of regular expressions from nondeterministic finite automata (NFAs) [HMU03]. It is suitable for a range of indefinite-horizon properties, including reachability and expected reward. An example below illustrates that there is little advantage in doing state elimination over resorting to solving the linear equation system by (some variant of) Gaussian elimination. However, if we are only interested in the probability from the initial state, we do not need to solve the full equation system. The state-elimination algorithm, in which we can remove unreachable states, optimises for this observation, in contrast to (standard) linear equation solving. Moreover, state elimination easily allows to utilise heuristics that take into account the ordering of the states[1].

state elimination

Observation | We assume w. l. o. g. that all target states T are absorbing and that $\iota \notin T$. The core idea behind state elimination is based on two operations:

- *Adding short-cuts:* Consider the pMC-fragment in Figure 8.2a. The reachability probabilities from any state to t are as in Figure 8.2b, where we replaced the transition from s to s' by shortcuts from s to t and all other successors of s', bypassing s'. By successive application of shortcuts, any path from the initial state to the target state eventually has length 1.

[1] Personally, I like state elimination as a way to solve small MCs as it makes mistakes more obvious.



(a) pMC fragment (b) reachability prob. (c) Before loop elim. (d) After loop elim.

Figure 8.2: Essential ideas for state elimination

Algorithm 2 State elimination for pMCs

function ELIMINATE_SELFLOOP($\mathcal{P}, s \in S$)
  assert $\mathcal{P}(s,s) \neq 1$
  for each $s_{out} \in \mathsf{succ}(s), s \neq s_{out}$ **do**
    $\mathcal{P}(s, s_{out}) \leftarrow \frac{\mathcal{P}(s,s_2)}{1-\mathcal{P}(s,s)}$
  $\mathcal{P}(s,s) \leftarrow 0$

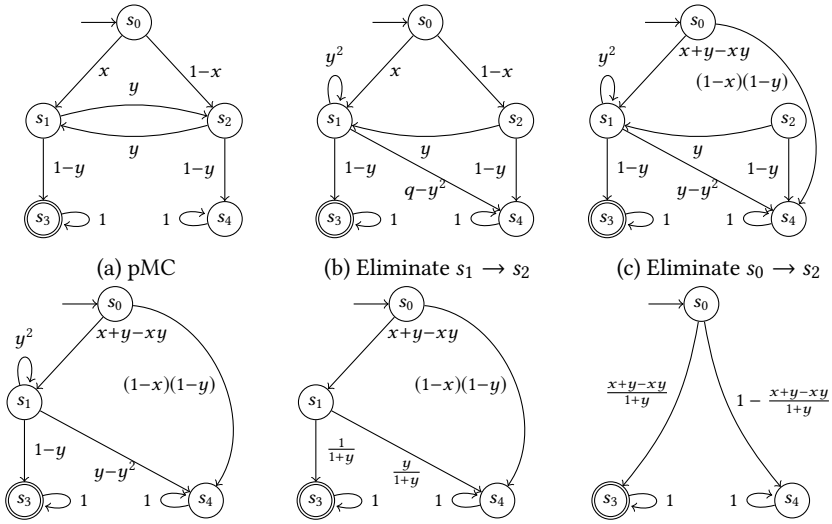
function ELIMINATE_TRANSITION($\mathcal{P}, s_{in} \in S, s \in S$)
  assert $s_{in} \in \mathsf{prec}(s), \mathcal{P}(s,s) = 0$
  for each $s_{out} \in \mathsf{succ}(s)$ **do**
    $\mathtt{old} \leftarrow \mathcal{P}(s_{in}, s) \cdot \mathcal{P}(s, s_{out})$
    $\mathcal{P}(s_{in}, s_{out}) \leftarrow \mathcal{P}(s_{in}, s_{out}) + \mathtt{old}$
  $\mathcal{P}(s_{in}, s) \leftarrow 0$

function ELIMINATE_STATE($\mathcal{P}, s \in S$)
  assert $\mathcal{P}(s,s) = 0$
  for each $s_{in} \in \mathsf{prec}(s)$ **do**
    ELIMINATE_TRANSITION($\mathcal{P}, s_{in}, s$)

function REACHABILITY(pMC $\mathcal{D}, T \subseteq S$)
  $S^? \leftarrow \{s \in S \mid s \neq \iota \wedge s \in \Diamond T \setminus T\}$
  while exists $s \in S^?$ **do**
    ELIMINATE_SELFLOOP($\mathcal{P}, s$)
    ELIMINATE_STATE($\mathcal{P}, s$)
    $S^? \leftarrow S^? \setminus \{s\}$
  ELIMINATE_SELFLOOP($\mathcal{P}, \iota$)
return $\sum_{t \in T} \mathcal{P}(\iota, t)$

- *Elimination of self-loops:* A prerequisite for introducing a short-cut is that the bypassed state is loop-free. Recall that the probability of staying forever in a non-absorbing state is zero and justifies elimination of self-loops by rescaling all other outgoing transitions, as depicted in the transition from Figure 8.2c to Figure 8.2d.

Reachability probabilities | We describe the algorithm to compute reachability probabilities based on state-elimination in Algorithm 2. In the following, $\mathcal{P}$ is the transition matrix. The function ELIMINATE_SELFLOOP($\mathcal{P}, s$) rescales all outgoing probabilities of a non-absorbing state s by eliminating its self-loop. The function ELIMINATE_TRANSITION($\mathcal{P}, s_1, s_2$) adds a shortcut from s_1 to the successors of s_2. Both operations preserve reachability to T. The function ELIMINATE_STATE($\mathcal{P}, s$) "bypasses" a state s by adding shortcuts from all its predecessors. More precisely, we eliminate the incoming transitions of s and after all incoming transitions are removed, the state s is unreachable. It is thereby effectively removed from the model.After removing all non-absorbing,



(a) pMC (b) Eliminate $s_1 \longrightarrow s_2$ (c) Eliminate $s_0 \longrightarrow s_2$

(d) Remove unreachable s_2 (e) Eliminate loop on s_1 (f) Eliminate s_1

Figure 8.3: State elimination by example

non-initial states $S^?$, the remaining model contains only self-loops at the absorbing states and transitions emerging from the initial state. Eliminating the self-loop on the initial state (by rescaling) yields a pMC. In this pMC, after a single step, an absorbing state is reached. The absorbing states are either a target state or not. The solution function is then the sum over all transition probabilities to target states in T.

Example 8.5 Consider the pMC from Figure 8.1a, repeated in Figure 8.3a. Assume state s_2 is to be eliminated. Applying the function ELIMINATE_STATE($\mathcal{P}, s_2$), we first eliminate the transition $s_1 \longrightarrow s_2$, which yields Figure 8.3b and subsequently eliminate the transition $s_0 \longrightarrow s_2$ (Figure 8.3c). State s_2 is now unreachable, so we may remove s_2, reducing computational effort when eliminating state s_1. For state s_1, we first eliminate the self-loop (Figure 8.3e) and then eliminate the transition $s_0 \longrightarrow s_1$. The final result, after additionally removing the now unreachable s_1, is depicted in Figure 8.3f. The result, i.e., the probability to reach s_3 from s_0 in the original model, can now be read from the single transition between these two states.

As for computing of regular expressions from NFAs, the order in which the states are eliminated is essential. Computing an optimal order with respect to minimality of the result, however, is already NP-hard for acyclic NFAs, see [Han13]. For state elimination

on pMCs, the analysis is more intricate, as the cost of every operation crucially depends on the size and the structure of the rational functions. For the transition elimination, a wide range of heuristics for the ordering are available. STORM supports a variety of static (pre-computed) and dynamic orderings for the elimination:

- Several static orders based on the order of state-generation by the model construction algorithms. This latter order is typically determined by a depth-first search through the high-level model description. However, this order is destroyed during the computation of a bisimulation quotient.
- Static orders based on the pMC topology, such as the SCC decomposition [JCVW+14].
- Dynamic orders which take into account the in-degree (the number of incoming transitions at a state), inspired by [Sak05; Han13],
- Dynamic orders which take into account the complexity of the rational function corresponding to the transition probability. The complexity is defined by the degree and number of terms of the occurring polynomials.

The orders are computed as penalties for states, and we eliminate states based on the order. For dynamic orderings the penalties are recomputed as the in-degree of states and complexity of transition probabilities change during the elimination.

Expected rewards | The state elimination approach can also be adapted to compute expected rewards [HHZ11b]. When eliminating a state s, in addition to adjusting the probabilities of the transitions from all predecessors s_1 of s to all successors s_2 of s, it is also necessary to "summarise" the reward that would have been gained from s_1 to s_2 via s. The presentation in [HHZ11b] describes these operations on transition rewards. For the analysis of expected rewards in MCs, we can always reformulate transition rewards in terms of state rewards. We formulate the algorithm immediately on the state rewards, this adjustment simplifies the necessary operations considerably.

The treatment of the expected reward computation is easiest from an adapted (and more performant) implementation of state elimination for reachability probabilities, as outlined in Algorithm 3. Here, we eliminate the probabilities to reach a target state in exactly one step and collect these probabilities in a vector $\vec{p}$ which we refer to as *one-step-probabilities.* Then, we proceed similar as before. However, the elimination of a transition from s_{in} to s now has two effects: it updates the probabilities within the non-target states as before, and (potentially) updates the probability $\vec{p}(s_{in})$ to reach the target within one step from s_{in} (with the probability that the target was reached via s in two steps). Upon termination of the outer loop, the vector $\vec{p}$ contains the probabilities from all states to reach the target, that is, $\vec{p}(s_i) = {}^{\mathbb{P}}_{\mathrm{gp}}\mathsf{sol}^{\mathcal{D}}_{s_i \to T}$.

Finally, when considering rewards, the *one-step-probabilities* contain initially the rewards for the states. Eliminating a transition then moves the (expected) reward to the predecessors by the *same* sequence of arithmetic operations.

Algorithm 3 State elimination with one-step probabilities

function ELIMINATE_TRANSITION($\mathcal{P}, \vec{p}, s_{in} \in S, s \in S$)
  assert $s_{in} \neq s, \mathcal{P}(s,s) \neq 1$
  $\vec{p}(s_{in}) \leftarrow \vec{p}(s_{in}) + \frac{\mathcal{P}(s_{in},s)\cdot\vec{p}(s)}{1-\mathcal{P}(s,s)}$ ▹ to targets
  for each $s_{out} \in \mathsf{succ}(s), s \neq s_{out}$ **do**
    $\mathcal{P}(s_{in}, s_{out}) \leftarrow \mathcal{P}(s_{in}, s_{out}) + \frac{\mathcal{P}(s_{in},s)\cdot\mathcal{P}(s,s_{out})}{1-\mathcal{P}(s,s)}$
  $\mathcal{P}(s_{in}, s) \leftarrow 0$

function ELIMINATE_STATE($\mathcal{P}, \vec{p}, s \in S$)
  assert $\mathcal{P}(s,s) = 0$
  for each $s_{in} \in \mathsf{prec}(s)$ **do**
    ELIMINATE_TRANSITION($\mathcal{P}, \vec{p}, s_{in}, s$)

function REACHABILITY(pMC $\mathcal{D}$, $T \subseteq S$)
  $S^? \leftarrow \{s \in S \mid s \in \Diamond T \setminus T\}$
  $\vec{p}(s) \leftarrow \sum_{t \in T} \mathcal{P}(s,t)$ for each $s \in S^?$ ▹ $\vec{p}\colon S^? \to [0,1]$
  $\mathcal{P}(s,t) \leftarrow 0$ for all $s \in S, t \in T$
  while exists $s \in S^?$ **do**
    ELIMINATE_STATE($\mathcal{P}, \vec{p}, s$)
    $S^? \leftarrow S^? \setminus \{s\}$
  return $\vec{p}(\iota)$

8.1.3 Set-based transition elimination

To succinctly represent large state spaces, MCs are often represented by multi-terminal binary decision diagrams (or variants thereof) [BCHK+97]. Such a *symbolic representation* handles sets of states instead of single states (and thus also sets of transitions) and thereby exploits symmetries and similarities in the underlying graph of a model. To support efficient elimination, we describe how to eliminate sets of transitions at once. The method is similar to the Floyd-Warshall algorithm for all-pair shortest paths [CLRS09]. The transition matrix contains one-step probabilities for every pair of source and target states. Starting with a self-loop-free pMC (obtained by eliminating all self-loops from the original pMC), we iterate two operations until convergence. By doing a matrix-matrix multiplication, we effectively eliminate all transitions emanating from all non-absorbing states *simultaneously*. As this step may reintroduce self-loops, we eliminate them in a second step. As before, eventually only direct transitions to absorbing states remain, which effectively yield the unbounded reachability probabilities. The corresponding pseudo-code is given in Algorithm 4.

The approach of this algorithm can conveniently be explained in the equation system representation. Let us therefore conduct one step of the algorithm as an example, where we use the observation that the matrix-matrix multiplication corresponds to replacing the variables p_s by their defining equations in all *other* equations.

Algorithm 4 Set-based transition elimination for pMCs

```
function REACHABILITY(pMC 𝒟, T ⊆ S)
    S? ← {s ∈ S | s ≠ ι ∧ s ∈ ◇T \ T}
    for each s ∈ S? do                                  ▷ can be done in parallel for all s
        ELIMINATE_SELFLOOP(𝒫, s)
    while exists s, s′ ∈ S?. 𝒫(s, s′) ≠ 0 do
        for each s ∈ S?, s′ ∈ S do                      ▷ can be done in parallel for all s, s′
            𝒫′(s, s′) ← ∑_{s″} 𝒫(s, s″) · 𝒫(s″, s′)
        for each s ∈ S? do                              ▷ can be done in parallel for all s
            ELIMINATE_SELFLOOP(𝒫′, s)
        𝒫 ← 𝒫′
    return ∑_{t∈T} 𝒫(ι, t)
```

Example 8.6 Reconsider our running example pMC, also depicted in Figure 8.4a:

$$
\begin{aligned}
p_0 &\doteq x \cdot p_1 + (1-x) \cdot p_2, & p_3 &\doteq 1,\\
p_1 &\doteq y \cdot p_2 + (1-y) \cdot p_3, & p_4 &\doteq 0,\\
p_2 &\doteq y \cdot p_1 + (1-y) \cdot p_4.
\end{aligned}
$$

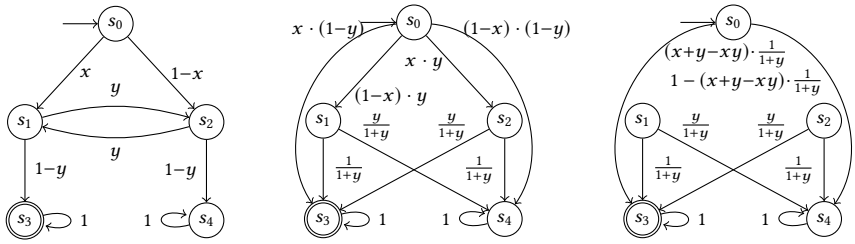
Replacing the equations for p_0, p_1, p_2 in all *other* equations yields:

$$
\begin{aligned}
p_0 &\doteq x \cdot (y \cdot p_2 + (1-y) \cdot p_3) + (1-x) \cdot y \cdot p_1 + (1-y) \cdot p_4)\\
p_1 &\doteq y \cdot (y \cdot p_1 + (1-y) \cdot p_4) + (1-y) \cdot p_3\\
p_2 &\doteq y \cdot (y \cdot p_2 + (1-y) \cdot p_3) + (1-y) \cdot p_4,
\end{aligned}
$$

which simplifies to

$$
\begin{aligned}
p_0 &\doteq (1-x) \cdot y \cdot p_1 + x \cdot y \cdot p_2 + x \cdot (1-y) \cdot p_3 + (1-x)(1-y) \cdot p_4\\
p_1 &\doteq \frac{1}{1+y} \cdot p_3 + \frac{y}{1+y} \cdot p_4\\
p_2 &\doteq \frac{y}{1+y} \cdot p_3 + \frac{1}{1+y} \cdot p_4.
\end{aligned}
$$

We depict the pMC which corresponds to this equation system in Figure 8.4b. Again, notice the similarity to state-elimination. For completeness, the result after another iteration is given in Figure 8.4c.



(a) pMC (b) First iteration (c) Second iteration

Figure 8.4: Set-based transition elimination by example

The correctness follows from the following argument: After every iteration, the equations describe a pMC over the same state space as before. As all absorbing states have defining equations $p_i \in \{0, 1\}$, the equation system is known to have a unique solution [BK08]. Moreover, as the equation system in iteration i implies the equation system in iteration $i+1$, they preserve the same (unique) solution. For explicit state space representations, this method is inferior to standard state elimination, as it introduces quite some overhead.

8.1.4 Empirical evaluation

We give a brief empirical evaluation to back two claims:

- *The solution functions may grow large for many case studies*, showing the limited use of solution functions and the inherit complexity of the problem, after all, every heuristic eventually computes this function (in a partially factorised fashion).
- *It is useful to have various heuristics for computing the solution function.* This evaluation does not (aim to) answer which heuristic for computing solution functions is most suitable. Rather, we conclude—in line with more experiments in [HHZ11b; 3; 20]—that given the current implementations, each approach has its benefits.

We report on the computation of the solution function using 18 configurations. We use state elimination with 7 different heuristics, set-based transition elimination, and Gaussian elimination. All methods are evaluated with and without strong bisimulation minimisation as preprocessing.

Case studies | Throughout this chapter, we use the selection of case studies from [17] and refer to that paper for more details about the case studies. The *NAND* model is a famous probabilistic model checking benchmark [NPKS05] that models a technique to build reliable devices from unreliable components. The probabilities model failure probabilities. We take the pMC from [3], that replaced these probabilities with paramet-

ers. The *Herman* model is a famous protocol for a self-stabilising distributed system, i.e., a protocol that from any system configuration reaches valid configurations. It uses randomisation to break the symmetry. One of the critical questions is how the randomisation affects the recovery time, i.e., the time until reaching a valid configuration. This question has been investigated with probabilistic model checking [KNP12] and parameter synthesis [AVBK+17]. The *HECS* dynamic fault tree [VS02] describes how faulty components in a computer may cause the failure of the computer itself. We use parametric failure rates, as first described in [6] and analyse the mean-time-to-failure of the computer. The *Netw* model [YMZ11] describes traffic scheduling in a wireless downlink with partially observable channel qualities. The system is modelled by a POMDP [NPZ17] and translated into a pMC, cf. Section 7.1. Concretely, the parameters represent how the strategy randomises. We evaluate the expected packet loss.

Results | First, we show the sizes of the solution function: The results are summarised in Table 8.1. The first columns summarise relevant information about the concrete instances that we took from the benchmarks. The *id* is used for reference. The *name* refers to the name of the benchmark set, while the *instance* describes the particular instance from this benchmark set. We give the total number of *parameters* $|X|$ both in the transition matrix as well as in the reward structure whenever applicable. For the next columns, we give two numbers per benchmark instance: The upper row describes the original model, the latter describes the (strong) bisimulation quotient. The columns give the number of *states* and *transitions*. The next row gives the *time* (in seconds) required for constructing the model (top) and constructing the bisimulation quotient (bottom). Then, there is a double line. The next four columns state properties of the resulting rational function. We give the degree of both the numerator (*degree num*) and denominator (*degree denom*), as well as the number of terms in both polynomials (*# terms num*, *# terms denom*). The next column gives the number of configurations (out of the 18) which *successfully* finished within the time limit. The last two columns indicate timings. We give the times (in seconds) to compute the solution function (*time elim*) and the *total time* including model building, (optional) bisimulation minimisation, and computing the solution function. For these timings we give two numbers per benchmark instance: The upper row describes the median value over all successful configurations and the lower row describes the best result obtained. Thus, *while functions often grow prohibitively large, medium-sized functions can still be computed.* The *Netw* model scales rather well, even in the presence of more parameters, as the model is acyclic and parameters occur in an orderly fashion. In particular, it shows that the number of parameters is, in a practical setting, not a good indicator of the run time. However, the model itself is rather small and the number of terms in the solution function is already larger than the number of states. Contrary to model checking for parameter-free models, model building is typically *not* the bottleneck.

Table 8.1: Computation of Solution Function

id	name	instance	$\|X\|$	states	# trans	time	degree num	degree denom	# terms num	# terms denom	success	time elim.	time total
1	*HECS*	m=1,k=1,i=1	2	129	489	0.02	23	24	234	247	16	2.00	2.09
				25	71	0.00						0.64	0.72
2		m=1,k=1,i=2		145	589	0.02	31	32	408	425	16	9.12	9.21
				49	173	0.00						3.00	3.08
3	*Herman*	N=3	1	9	36	0.02	0	2	1	2	18	0.00	0.09
				3	5	0.00						0.00	0.08
4		N=5		33	276	0.03	4	6	5	6	17	0.04	1.55
				5	15	0.00						0.00	0.10
5		N=7		129	2316	0.11	28	30	29	30	11	0.62	0.82
				16	137	0.02						0.37	0.56
6		N=9		513	20196	0.92	150	152	151	152	8	247.00	248.14
				347	15009	0.12						114.49	115.64
7	*NAND*	K=2,N=2	2	178	243	0.03	10	0	32	1	18	0.00	0.11
				125	167	0.00						0.00	0.09
8		K=2,N=20		154942	239832	2.81	100	0	2106	1	15	43.05	46.88
				102012	154722	0.91						15.46	19.35
9		K=2,N=30		681362	1065797	12.56	150	0	4653	1	13	469.29	486.74
				474847	732768	4.65						110.54	128.48
10		K=5,N=10		35112	52647	0.63	110	0	1220	1	15	6.30	7.24
				23603	34093	0.21						3.30	4.25
11		K=5,N=20		384772	594792	7.04	200	0	4640	1	13	245.47	256.05
				288102	436332	3.17						88.18	98.71
12		K=5,N=30		1697732	2653937	31.45	330	0	10260	1	1	3031.34	3083.88
				1345507	2074758	18.49						3031.34	3083.88
13	*Network*	c=2,K=2,T=2	8	52	133	0.00	1	0	23	1	16	0.00	0.07
				52	133	0.00						0.00	0.06
14		c=2,K=2,T=3	16	106	269	0.01	1	0	111	1	16	0.01	0.08
				106	269	0.00						0.01	0.07
15		c=2,K=2,T=4	24	164	411	0.01	1	0	519	1	16	0.04	0.11
				164	411	0.00						0.03	0.09
16		c=2,K=4,T=2	20	136	365	0.01	1	0	65	1	16	0.01	0.08
				136	365	0.00						0.01	0.08
17		c=2,K=4,T=3	36	262	691	0.01	1	0	289	1	16	0.07	0.15
				262	691	0.00						0.03	0.10
18		c=2,K=4,T=4	52	392	1023	0.01	1	0	1377	1	16	0.40	0.48
				392	1023	0.00						0.12	0.20

Furthermore, we see that the selected heuristic is indeed crucial. Consider instance 5: 11 heuristics successfully compute the solution function (and most of them within a second). However, 7 others yield a timeout. *Regex* based configurations were the fastest eight times, *DPen* based ones four times and three times configurations based on *FwRev* were fastest. Typically, bisimulation was beneficial. The other three instances were solved fastest by one of the other configurations. From these numbers, we conclude that *the selection of the heuristic is essential and depends on the model to be analysed.* From the graph, we further observe that although using a Gaussian elimination yields good performance, state-elimination based approaches can (significantly) outperform the Gaussian elimination on some benchmarks. The *DPen* solves all instances (the only configuration to do so), but *Regex* is overall (slightly) faster. The uninformed

FwRev with bisimulation works surprisingly well for these benchmarks (which likely is coincidence).

8.2 Feasibility and validity checking via ETR solvers

We have seen that the feasibility problem may be cast into an ETR-encoding in Section 4.3 and that any ETR-encoding may be reformulated into a feasibility problem in Theorems 5.49 and 5.56. Thus, a natural approach is to utilise methods for solving ETR to tackle the feasibility and validity problems.

Encoding for validity | To assess validity of $\neg\varphi$ on a region, we may simply take the encoding for feasibility for φ that states that:

- the valuation is inside the region,
- the valuation is well defined,
- the valuation is accepting.

If there is no such valuation, there is no accepting valuation inside that region, i.e., all valuations are rejecting φ and thus accepting $\neg\varphi$. Consequently, the validity problem for $\neg\varphi$ is true. Slight care is necessary when not all valuations in a region are graph-preserving, but this is mostly technical. The complete encodings are given in [17].

Practical aspects of the encoding

DIR
SF

We have seen two variants of the ETR encoding. One directly generalises the linear equation system (cf. Theorem 4.13, *DIR* encoding), the other uses the solution function as preprocessing (cf. Theorem 4.22, *SF* encoding). While the SF encoding only contains the parameters as variables, it is an exact encoding and thereby may be rather large. In particular, this encoding generally does not contain any obvious structure from, e.g., the topology of the pMC. It is a monolithic block, giving the ETR solver few options to draw any conclusions. On the other hand, the DIR encoding contains variables for every state and if the solver fails to comprehend the structure it generally will not efficiently solve a formula with that many variables. With state elimination on a subset of the states we obtain a compromise between the two encodings.

Any encoding may be extended with various spurious constraints. Here, we consider extending the DIR encoding. Alternatively, one may consider approximations of the solution function. The simplest of these spurious constraints is that—when considering reachability probabilities—all state variables in the encoding are between 0 and 1. While these constraints are already implied by the encoding, they sometimes help the solver. Inspired by, e.g., [CGIR+17], we may decide to leave out some of the original nonlinear constraints and thereby obtain an over-approximation. These ideas have not reached a

mature stage—in particular, parameter lifting as introduced in Chapter 12 is typically faster. Moreover, the performance of the ETR solvers is currently unpredictable and it remains a challenge to consistently improve the performance with spurious constraints.

Example 8.7 Some spurious constraints for the example in Figure 8.4a are:

$$p_0 \leqslant p_1 + p_2, \quad p_0 \geqslant x \cdot p_1, \quad p_0 \leqslant x + p_2.$$

Furthermore, we may use the continuity of the solution function on the graph-preserving valuations, even if we do not use the solution function for the encoding. In particular, rather than checking whether there is a valuation that is larger than the threshold, we may ask if there is a valuation equal to the threshold. In absence of such a valuation, we may deduce that all valuations are accepting or all are rejecting, and the evaluation of a single instantiation clarifies in which case we are. This change is interesting in combination with further (equisatisfiable) preprocessing of the formula.

Finally, we often see a need to check multiple regions on the same model. Depending on the used solver, this may be accommodated by tools that take as input a stack of formulae (e.g., incremental SMT-solvers): Therefore, consider that our encodings are a conjunction of subformulae. We may put the formulae that do not change (such as the inequalities that originate from the transition probabilities) first, then push and pop the constraints that describe the region. The solver then knows and profits from the fact that many constraints are unchanged.

Tools and methods

Solving ETR formulae is an intricate topic that cannot be covered in this thesis. Algebraic foundations are covered by a rather thick [BPR06], there is a vast body of literature and an even wider range of implementations, which are briefly reviewed below.

Selecting a solver | Solvers for ETR can be found in two flavours:

- *Computer algebra systems*, such as the open-source tools Maxima[1] and Sage[2], or the proprietary Maple[3], or Mathematica[4], to name a few.
- *SMT-solvers* for *quantifier free nonlinear real arithmetic*—the SMT name for ETR—such as Z3 [MB08], Yices [Dut14], and SMT-RAT [2].

So far, parameter synthesis methods have focussed on employing SMT solvers, with notable exception of [AVBK+17] and [FTG16] which use computer algebra systems to compute and analyse the solution function.

[1] http://maxima.sourceforge.net
[2] http://www.sagemath.org
[3] https://www.maplesoft.com/products/maple/academic/
[4] http://www.wolfram.com/mathematica/

Selecting a technique | We briefly discuss the most prominent techniques, embedded in the context of this thesis. So far, each of these techniques has been successful on some sets of problems, and both their integration, e.g. [LSCÁ+13] and the automatic selection of the right technique [MP13] is an ongoing process.

CAD

Without doubt, the most prominent approach is the *cylindrical algebraic decomposition* (*CAD*) [Col75; CH91; Bro04]. The technique is also suitable for the (full, quantified) theory of the reals. Intuitively, the CAD decomposes the parameter space into finitely many cells, where each cell is a semi-algebraic set where all polynomials in the formula are sign-invariant. The CAD first iteratively eliminates variables, then finds roots of a doubly exponential set of univariate polynomials, and then backtracks through the iterations. In each backtracking/lifting step, these roots are inserted into the polynomials from the previous iteration. In this sense, CAD does a parameter space partitioning as we discuss in Section 8.4. However, the representation of the decomposition is intricate, as a representation of the roots is essential: these roots may be arbitrary algebraic numbers. For feasibility analysis, one may argue that CAD does too much work: It always computes a cell around a solution.

Rather than working from the input formula and generating a decomposition from that, model constructing SMT [JM12]—intuitively—selects a (partial) valuation for (some) variables in the formulae and considers the CAD cell around this valuation. Thereby, it is better in learning the structure and better in finding satisfiable solutions. This approach is at the heart of every modern SMT solver for ETR and may be combined with other techniques and embedded into an SMT framework, see [MJ13].

Gröbner bases

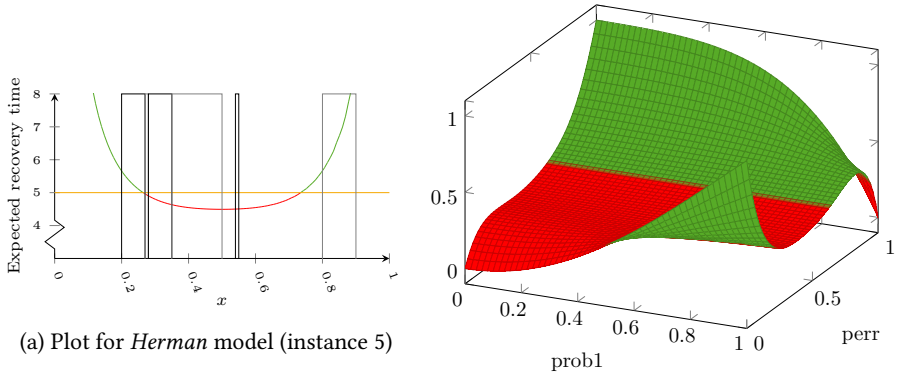
A prominent idea from algebra is the computation of a normal form, a basis. In particular, *Gröbner bases* [Buc65; CLO97] may be considered as a generalisation of the Euclidean algorithm for division of integers (or polynomials), and as a generalisation of Gaussian elimination for solving linear equation systems. In particular, it allows not only to eliminate the state-variables in the encoding, but also the parameters. However, it is necessarily slower than Gaussian elimination when applied to the state variables (as there is more overhead). Rather than for elimination, it can be used as a preprocessing technique [1] to generate additional constraints.

virtual substitution

Virtual substitution [Wei97; CÁ11; Stu18] generalises Gaussian elimination to formula with polynomials up to degree four. However, when the degree rises above degree four during substitution, the method is no longer applicable. Therefore, it is not applicable to many standard benchmarks in parameter synthesis.

ICP

Finally, *interval constraint propagation* (*ICP*) [Moo79; BG06] is an incomplete method that tries to prove the absence of a satisfying valuation. Therefore, ICP starts with a rectangular region and refines the interval for one variable in which a satisfying solution may lie, if any. To find this refinement, the most basic idea is that an interval is computed by substituting the intervals for all other variables in the encoding (propagation). Other contraction operators exist. For the encoding that generalises the linear inequalities, the major challenge is to find the right combination of constraint and



(a) Plot for *Herman* model (instance 5)

(b) Plot for *NAND* model (instance 7)

Figure 8.5: Illustrating accepting and rejecting regions.

variable that may be refined. Generally, substituting backwards through the topology of the MC is a good idea, but an almost similar result is obtained by parameter lifting in Chapter 12, with significantly less overhead. On the high-degree solution function, ICP struggles to find significant contractions.

Empirical evaluation

We evaluate region verification using two ETR-encodings (SF: based on first computing the Solution Function, or DIR: DIRectly encoding the equations into ETR), and parameter lifting (PLA), a technique from Chapter 12. We show that using SMT solvers to analyse these encodings is a method that, although just in a few cases, is state-of-the-art, and that from a practical point, both an SF and an DIR encoding are justified (for pMCs).

In particular, we present some results for the *Herman* benchmark: it features a single parameter, and therefore is well-suited for the illustration of some concepts. We visualised the results for instance 5 in Figure 8.5a. The x-axis represents the parameter value for parameter x, and reflects the bias of the coins. The y-axis represents the induced expected recovery time. The threshold in the following is set to $\lambda = 5$ and indicated by the orange horizontal line. We indicate the solution function in red/green, depending on how its value compares to the threshold. The black columns depict six different regions[1] that are evaluated with region checking. For each region we want to verify whether the expected recovery time is at least 5. The results are summarised in Table 8.2a. The first columns indicate the considered *region* and the *technique*. The last

[1]While regions are given by the intervals for the parameter, we depict the columns for better visibility.

Table 8.2: Some benchmarks for region verification

(a) instance 5, $\lambda := 5$

region	techn.	result	time
[0.20, 0.27]	DIR	inconsistent	12,11
	PLA	unknown	0.01
	SF	unknown	TO
[0.27, 0.28]	DIR	reject	20.68
	PLA	reject	0.01
	SF	unknown	TO
[0.28, 0.35]	DIR	reject	53.47
	PLA	unknown	0.01
	SF	unknown	TO
[0.35, 0.50]	DIR	reject	23.41
	PLA	reject	0.00
	SF	unknown	TO
[0.54, 0.55]	DIR	reject	22.35
	PLA	reject	0.01
	SF	unknown	TO
[0.80, 0.90]	DIR	unknown	TO
	PLA	accept	0.01
	SF	unknown	TO

(b) instance 7, $\lambda := 0.3$

region	techn.	result	time
$[0.01, 0.99] \times [0.70, 0.90]$	DIR	accept	16.20
	PLA	unknown	0.01
	SF	accept	0.16
$[0.01, 0.99] \times [0.90, 0.99]$	DIR	inconsistent	19.41
	PLA	unknown	0.01
	SF	inconsistent	0.04
$[0.01, 0.50] \times [0.65, 0.70]$	DIR	accept	45.61
	PLA	unknown	0.01
	SF	accept	0.13
$[0.01, 0.50] \times [0.75, 0.90]$	DIR	accept	4.58
	PLA	accept	0.01
	SF	accept	0.12
$[0.80, 0.99] \times [0.01, 0.10]$	DIR	accept	9.90
	PLA	accept	0.00
	SF	accept	0.07
$[0.01, 0.99] \times [0.40, 0.50]$	DIR	reject	19.82
	PLA	reject	0.00
	SF	reject	0.08

columns give the *result* of the region verification and the *time* (in seconds) needed for the computation. The timeout (TO) was set to 120 seconds. For this benchmark, PLA computes a result within milliseconds and the computation time is independent of the considered region. The SMT-based techniques take longer and the SF technique in particular does not terminate within two minutes. However, the DIR technique could yield a result for region $[0.28, 0.35]$ whereas PLA could not give a conclusive answer due to its inherent over-approximation.

We now consider region verification on the *NAND* model with two parameters. We visualised the solution function for instance 7 in Figure 8.5b. The considered threshold is $\lambda := 0.3$. Green coloured parts indicate parameter instantiations leading to probabilities above λ and red parts lie below λ. The results of the verification for different regions are given in Table 8.2b. PLA is again the fastest technique, but for larger regions close to the threshold PLA can often not provide a conclusive answer. Contrary to before, SF is superior to DIR.

The performance of the SMT-based techniques (again) greatly depends on the considered region. It is only natural that *the size of the region, and the difference to the threshold have a significant influence on the performance of region verification.* These observations are general and do *hold on all other benchmarks.* Furthermore, parameter lifting seems broadly applicable, and in the setting evaluated here, clearly faster than SMT-based approaches. PLA over-approximates and therefore might only give a decisive result in a refinement loop (not applied here), such as presented in Section 8.4. The SMT-based approaches are a valuable fallback. When relying on the SMT techniques,

it is heavily model-dependent which performs better. Table 8.4 at the end of the next section gives additional results, indicating the performance of different verification techniques.

> **Remark 8.2** We deliberately do not provide more numbers in this evaluation. Above, and Sections 7.1 and 8.4 and Chapters 9 and 12 show that generally, using an SMT solver is *currently* not competitive, although there are some interesting corner cases in which either DIR or SF outperforms other approaches, e.g., when the solution function is (almost) constant. Proper evaluation of the current state of ETR-based parameter synthesis is a labor intensive task, and the underperforming implementation is not relevant enough to justify this effort. We are optimistic that using the tremendous research steps, e.g. [ÁABB+16], towards understanding and solving ETR, and after integrating parameter synthesis better, the ETR encoding will become practically relevant, and recommend to do a proper evaluation then.

8.3 Sampling the Parameter Space

We report on sampling-based approaches to parameter synthesis. Sampling is an essential idea that reoccurs in various approaches, yet we feel that there is a lot of room for improvement. Below, we briefly study existing ideas and approaches.

The idea behind sampling is to select instantiations $M \in \langle \mathcal{M} \rangle$ and then evaluate whether M accepts φ. The hope is to quickly sample an instantiation that accepts φ and solves the feasibility problem. In this best case, sampling scales much better than the ETR-based approach, but sampling an uncountable domain is necessarily incomplete.

A typical sampling-based method may consist of two blocks, as seen in Figure 8.6. In general, given a parameter space, and a description of the model, the sample synthesiser selects a valuation of the parameters (a *sample*), and passes this sample to a verifier. The verifier analyses the instantiation and either reports that the instantiation accepts the given specification, or reports any valuable information back to the synthesiser. Below, we go into details for both blocks.

sample

Sample synthesiser

Let us first consider the sample synthesiser. The bare minimum input for the synthesiser is the parameter space. In particular, there is no need to know the model, but it may be beneficial. There has been little work on selecting samples in the context of pMCs and pMDPs. Most prominently, evolutionary algorithms are used that explore the parameter space [CHHK+13]. From these approaches, particle swarm optimisation (*PSO*) has been considered as a successful technique, and is implemented in PROPhESY.

PSO

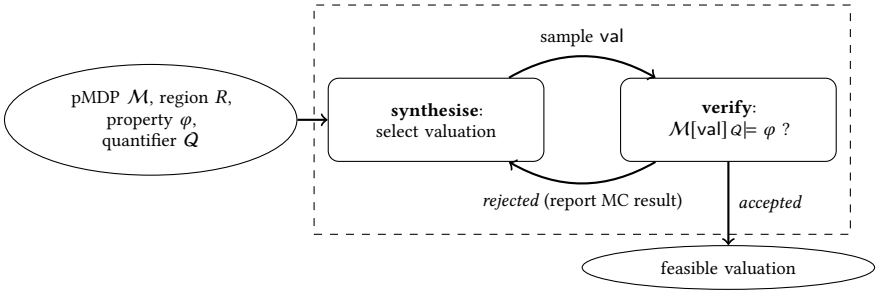


Figure 8.6: Scheme for sampling based feasibility.

Particle swarm optimisation | PSO [EK95; ES98; Ken10] is a heuristic sampling approach which searches the parameter space. In the context of feasibility analysis of pMDPs with a fixed reachability property, PSO takes the reachability probability as a cost, and aims to maximise (or minimise) these cost. Thereby, it ultimately finds a sample that is above (below) the threshold specified in the property. PSO maintains a set of samples in each iteration, and iteratively moves these samples in the parameter space, based on the evaluation of the samples from the previous iterations. After moving the samples, the samples are all evaluated by the verifier. To move the samples, the synthesiser associates each particle with a direction and a speed in the parameter space, and updates these according to previous iterations by a user-defined formula. The speed and direction for each sample are measures to prevent premature convergence.

We choose not to go into details about PSO. Generally, we have been able to use it to some success as a blackbox technique, cf. Section 7.1, which includes also various benchmarks. As pMCs may model a wide variety of polynomials, we have no tangible general results about PSO for feasibility analysis. We have made two main observations: (1) In general, and contrary to most PSO literature, sampling is rather expensive compared to selecting new samples. Therefore, PSO might not be the most adequate sampling-based approach to feasibility analysis—and it might be better to carefully select samples. This particularly holds for using large sets of samples in early iterations, or for models with many parameters. Contrary, for parameters with just a few parameters, PSO generally performs very well. (2) For many models, feasible solutions lie at the borders of the well-defined parameter space, i.e., at the bounds of the intervals for some parameters. It is therefore necessary to not steer the samples away from the ill-defined valuations beyond the border, while at the same time avoiding to accumulate all samples in the corners. Simple pMDPs with a rectangular well-defined region simplify the technical realisation of such ideas.

> **Remark 8.3** We do not provide a premature evaluation of PSO or other sampling approaches, and refer to Section 7.1 and Chapter 9 for some experimental results. We believe that sampling is an essential ingredient, but think that the methods mentioned above fail to exploit the structure of the problem.

Sampling beyond evolutionary algorithms | A notable exception to the use of evolutionary algorithms is a greedy approach [PÁJT+15]. However, the approach is only successful when the solution function is monotonic in its parameters. Many examples are monotonic, but asserting this automatically is intricate (and ETR-complete) [18].

Sampling beyond feasibility | Sampling is an evident choice to find good candidates in optimal synthesis or model repair problems. For optimal synthesis, it becomes more important to avoid any local optima than it is in plain feasibility synthesis. For model repair, the search consists of two components: Not only should the instantiation be cheap in terms of the model repair costs, it should accept the property. However, once an accepting instantiation has been found, it becomes important to tweak it towards a cheaper solution. We have seen instances in which these two aspects yield an intricate trade-off, which is hard to tackle with (vanilla, black-box) sampling approaches, as it is nontrivial to steer the sampling. In a similar spirit, specifications that contain multiple properties are a challenge for vanilla sampling.

In Section 8.4, we consider sampling to get an impression of the parameter space. Roughly, we select samples such that we have multiple accepting and rejecting samples spread across the accepting and rejecting region. We mostly use various variants of extrapolation and adaptions of Voronoi diagram computations [Aur91].

Sample verifier

A sample verifier checks samples, i.e., it checks parameter-free models. Thus, the verifier may rely on standard probabilistic model checking, it is therefore well understood and well supported by a variety of methods and tools.

Model checking for sampling | The trivial approach is to rely on a model checker as a black box. Minor adaptions of the model checking procedure may yield significant improvements in the context of sampling-based feasibility analysis:

First and foremost, *model construction* is often a bottleneck in model checking. Observing that the structure of the model is not changed when sampling multiple graph-preserving instantiations, it becomes essential to avoid reconstructing the model in every iteration from scratch. For sampling the graph-preserving instantiations, the model is constructed once and then changed based on a set of annotations.

Furthermore, model checking approaches may be improved by starting with some *hint*, a mapping from states to probabilities, which—optimally—is close to the reachability probability to the target for these states. When checking two instantiations whose corresponding valuations are close in the parameter space, it is—on average—beneficial to start the second model checking call with the solution from the first as a hint.

Strictly, we are only interested in determining whether an instantiation is accepting φ or rejecting φ. Many model checkers may determine this without computing the (precise) *reachability probabilities*. However, such early termination prevents the construction of a good hint, and limits the amount of information that the verifier may communicate back to the synthesiser: Instead of providing the reachability probability for the instantiation, the verifier now just returns that the instantiation rejects φ. A good compromise is the usage of (variants of) interval iteration [HM18; BKLP+17; QK18], yielding for every state an interval in which the reachability probability lies.

One intricacy is numerical stability and/or convergence behaviour of the model checker. When modelling a single MC, modellers may (should!) take into account the effects that small probabilities or long chains have on the model checking performance, i.e., of the condition of the underlying matrix. When modelling a set of Markov chains, this becomes much more challenging. Whereas model checkers may be manually tweaked using the large set of options such that their performance is optimised for a single (type) of MC, a pMC describes a set of MCs which may behave drastically different, and tweaking is both far from obvious and almost impossible to do manually.

Beyond model checking | An alternative to directly using model checking is to sample from the solution function. This approach requires a representation of the solution function, either in an explicit form computed as in Section 8.1, or as a computer program constructed with similar ideas [GHS18]. Constructing either of these representations yields an often prohibitive overhead—but eventually sampling on the solution function is faster. We stress that an evaluation of the solution function is numerically unstable, and should be done with exact arithmetic.

8.4 Parameter Space Partitioning

We consider pMDPs and a fixed property, and a fixed quantifier over the strategies, and may then concisely write that a pMDP accepts (rejects) a property.

Typically, not all instantiations of a pMDP homogeneously accept or reject a specification, and it is interesting to get (nontrivial, easy to represent) sets of instantiations that do homogeneously all accept/reject the specification. Sampling, especially in low dimensions, may give a good impression about the accepting and rejecting instantiations, but lacks any guarantees beyond the sampled instantiations. We therefore generalise the idea from Figure 8.6, by choosing and checking regions instead of samples.

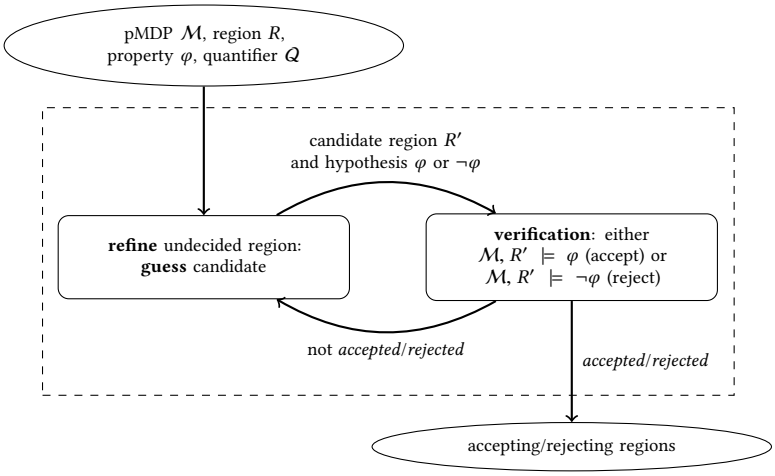


Figure 8.7: Parameter space partitioning outlined.

More precisely, in this section, the central issue is to obtain representations of the accepting region R_a and the rejecting region R_r as a union of simple regions such as affine or rectangular regions. Our approach for this *parameter space partitioning* iteratively obtains partial partitions of the parameter space. The main idea is to compute a sequence $\left(R_a^i\right)_i$ of simple accepting regions that successively extend each other. Similarly, an increasing sequence $\left(R_r^i\right)_i$ of simple rejecting regions is computed. At the i-th iteration, $R^i = R_a^i \cup R_r^i$ is the covered fragment of the parameter space. The iterative approach halts when R^i is at least $c\%$ of the entire parameter space. Termination is guaranteed under mild assumptions: in the limit a solution to the exact synthesis problem is obtained if $\lim_{i\to\infty} R_a^i = R_a$ and $\lim_{i\to\infty} R_r^i = R_r$. Figure 8.7 outlines such a partitioning procedure. As part of our synthesis method, we algorithmically *guess* a *candidate region* R and guess whether it is accepting or rejecting. We then exploit one of our verification methods to verify whether R is indeed accepting (or rejecting). If R is not accepting (rejecting), we exploit this information together with any additional information obtained during verification to refine the candidate region. This process is repeated until an accepting or rejecting region results. Such a region is then added to either R_r^i or R_a^i, i is incremented, and we continue with the undecided subregions.

candidate region

8.4.1 Naive refinement loop

Let us describe the synthesis loop in more detail. In particular, we discuss how to generate candidate regions which are dispatched to the verifier along with a *candid-*

candidate hypothesis

ate hypothesis whether the candidate region is accepting or rejecting. We focus on *rectangular regions* for several reasons:

- their generation is easier to generalise to multiple dimensions,
- earlier experiments [3] revealed that they lead to a more efficient SMT-based verification of regions (cf. Section 8.2), and
- model-checking based region verification (cf. Chapter 12) requires them.

A downside of rectangular regions is that they are neither well-suited to approximate a region partitioning given by a diagonal, nor to cover well-defined regions that are not rectangular themselves.

> **Remark 8.4** In the following, we assume that the parameter space is given by a rectangular well-defined region R. If the parameter space is not rectangular, we may over-approximate R by a rectangular region $\hat{R} \supseteq R$. If the potential over-approximation of the parameter space $\hat{R}$ is not well-defined, then we iteratively approximate $\hat{R}$ by a sequence of well-defined and ill-defined regions. A region R is ill-defined if *no* instantiation in R is well-defined. The regions in the sequence of well-defined regions are then subject to the synthesis problem. Constructing the sequence of regions is done analogously to the partitioning into accepting and rejecting regions.

Before we detail the procedure, we outline a naive refinement procedure.

> **Example 8.8 (Naive refinement loop)** Consider the parametric die from Example 1.4. Suppose we want to synthesise the partitioning as depicted in Figure 1.5 on page 10. We start by verifying the full parameter space R against φ. The verifier returns `false`, as R is not accepting. Since R (based on our knowledge at this point) might be rejecting, we invoke the verifier with R and $\neg\varphi$, yielding `false` too. Thus, the full parameter space R is inconsistent. We now split R into four equally-sized regions, all of which are inconsistent. Only after splitting again, we find the first accepting and rejecting regions. After various iterations, the procedure leads to the partitioning in Figure 8.8b (ignore the dots).

Algorithm 5 describes this naive region partitioning procedure. It takes a pMDP, a region R, a property φ, and a quantifier over the strategies as input. It first initialises a (priority) queue Q with R. In each iteration, a subregion R' of R is taken from the queue, the counter i is incremented, and the sequence of accepted and rejected regions is updated. There are three possibilities. Either R' is accepting (or rejecting), and R_a^i (R_r^i) extends R_a^{i-1} (R_r^{i-1}) with R', or R' is inconsistent[1]. In the latter case, we split R' into finitely many subregions and insert all of them into the queue Q. This algorithm may only terminate if R_a and R_r are a finite union of hyper-rectangles. However, the

[1] recall that any region which is not accepting or rejecting is inconsistent.

Algorithm 5 Naive refinement loop

```
function NAIVE-REFINEMENT(pMDP M, rectangular region R, Q ∈ {∃, ∀}, property φ)
    i ← 0, Q ← {R}, R_a^i ← ∅, R_r^i ← ∅
    while Q ≠ ∅ do
        R′ ← Q.pop(), i ← i + 1
        if R′ ⊆ acc_{Q,φ}^M then
            R_a^i ← R_a^{i−1} ∪ R′, R_r^i ← R_r^{i−1}
        else if R′ ⊆ rej_{Q,φ}^M then
            R_a^i ← R_a^{i−1}, R_r^i ← R_r^{i−1} ∪ R′
        else
            R_a^i ← R_a^{i−1}, R_r^i ← R_r^{i−1}
            Q ← Q ∪ SPLIT(R′)
```

algorithm can be terminated after any iteration yielding a sound approximation. The algorithm ensures $\lim_{i\to\infty} R^i = R$, if we order Q according to the size of the regions. We omit the technical proof here; the elementary property is that the regions are Lebesgue-measurable (and have a positive measure by construction).

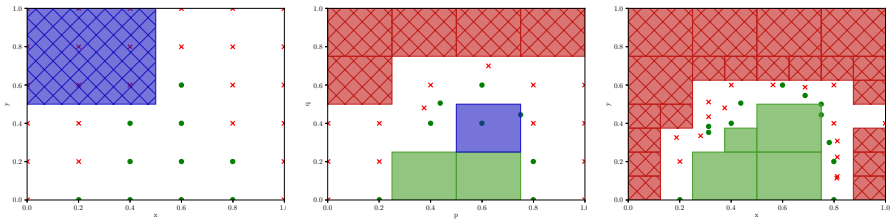
The naive algorithm has a couple of structural weaknesses:

- invoking region verification twice to determine that a region is inconsistent,
- not providing any diagnostic information from a failed verification invocation,
- checking if a rejecting region is accepting before checking if it is rejecting,
- splitting inconsistent regions into 2^k equally large regions, instead of splitting in a smaller number of regions (e.g., only split in one dimension),
- splitting naively, leading to many inconsistent subregions. Splitting in only one dimension even increases this number of inconsistent regions.

In the remainder of this section, we discuss ways to alleviate these weaknesses. The proposed improvements are based on empirical observations that we made while considering typical benchmarks, and are in line with the implementation in the tool PROPhESY, cf. Chapter 13. In particular, we assume "well-behaved" models and specifications. The notion of being well-behaved refers to

- a limited number of accepting (rejecting) valuations that are close to a rejecting (accepting) valuation. We call valuations that form a border between R_a and R_r *border* valuations. ·border
- a limited number of connected accepting and rejecting regions with smooth (albeit highly non-linear) borders between these regions.

The parameter space depicted in Figure 8.8c is well-behaved. It features only two connected regions, with a smooth border between them. The regions have a considerable interior, or equivalently, many valuations are not too close to the border.



(a) (Uniform) sampling (b) Generating candidates (c) Preliminary result

Figure 8.8: Parameter space partitioning in progress: Images generated by PROPHESY.

8.4.2 Finding region candidates

We use sampling to steer the selection of candidate regions.

sample

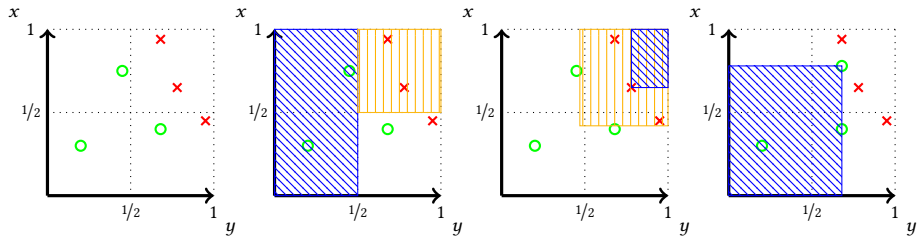
Sampling | A simple but effective improvement is to verify some *samples* $M \in \langle \mathcal{M} \mid R \rangle$. If the instantiation is accepting, then any region containing the corresponding valuation cannot be rejecting, and vice versa. Thus, already two samples within a region R may suffice to conclude that R is inconsistent. In order to quickly find inconsistent regions by sampling, it is beneficial to seek for border valuations. To this end, a good strategy is to start with a coarse sampling to get a first indication of border valuations. We then select additional instantiations by intra-/extrapolation.

> **Example 8.9** We illustrate how sampling may improve the loop discussed in Example 8.8. Figure 8.8a shows a uniform sampling. Red crosses indicate that the instantiated pMC satisfies $\neg\varphi$, while green dots indicate that the instantiation satisfies φ. The blue rectangle is a candidate region (with the hypothesis $\neg\varphi$, indicated by the hatching), which is consistent with all samples.

Sampling driven splitting | If the sampling reveals that a region is inconsistent, we do not need to apply verification on this region.

> **Example 8.10** Consider the parameter space with six samples depicted in Figure 8.9a. After verifying only six instantiated models, we conclude that the parameter space is inconsistent. We may make this conclusion even earlier, but the additional samples give additional information *how* to split.

The use of samples improves the naive refinement scheme as given in Algorithm 5. This improvement is given in Algorithm 6. For each region R, we have a finite set V of



(a) six samples (b) equal splitting (c) growing rectangles (d) largest candidate

Figure 8.9: Creating region candidates based on samples.

Algorithm 6 Sampling-based refinement loop

function SAMPLING-REFINEMENT(pMDP $\mathcal{M}$, rectangular region R, $\mathcal{Q} \in \{\exists, \forall\}$, property φ)
  $i \leftarrow 0$, $Q \leftarrow \{\langle R, \text{SAMPLE}(R)\rangle\}$, $R_a^i \leftarrow \emptyset$, $R_r^i \leftarrow \emptyset$
  while $Q \neq \emptyset$ **do**
    $i \leftarrow i + 1$, $\langle R', V'\rangle \leftarrow Q.\text{pop}()$
    if $V' \subseteq \mathsf{acc}^{\mathcal{M}}_{\mathcal{Q},\varphi}$ **and** $R' \subseteq \mathsf{acc}^{\mathcal{M}}_{\mathcal{Q},\varphi}$ **then**
      $R_a^i \leftarrow R_a^{i-1} \cup R'$, $R_r^i \leftarrow R_r^{i-1}$
    else if $V' \subseteq \mathsf{rej}^{\mathcal{M}}_{\mathcal{Q},\varphi}$ **and** $R' \subseteq \mathsf{rej}^{\mathcal{M}}_{\mathcal{Q},\varphi}$ **then**
      $R_a^i \leftarrow R_a^{i-1}, R_r^i \leftarrow R_r^{i-1} \cup R'$
    else
      $R_a^i \leftarrow R_a^{i-1}$, $R_r^i \leftarrow R_r^{i-1}$
      $Q \leftarrow Q \cup$ SPLIT(R', V')

samples. For each sample $\mathsf{val} \in V$, we may compute whether it accepts φ. The queue Q now contains pairs $\langle R, V\rangle$ with V being a set of annotated samples, i.e., valuations together with, e.g., the model checking results for them.

In each iteration, a pair $\langle R', V'\rangle$ is taken from the queue, where R' is (as before) a subregion of R and V' are samples in R'. Then, we distinguish (again) three possibilities. Only when all samples in V' accept φ, it is verified whether R' is accepting. If R' is accepting, we proceed as before: R_a^i is extended by R' while R_r^i remains unchanged. In the symmetric case that all samples in V' reject φ, we proceed in a similar way by verifying whether R' rejects φ. Otherwise, R' is split into a finite set of subregions with corresponding subsets of V', and added to the queue Q. Some region verification engines provide counterexamples: these are accepting (rejecting) samples that show that the region cannot be rejecting (accepting). We may add these counterexamples as a new sample. We thus ensure that for all $(R', V') \in Q$, $\mathsf{val} \in V'$ implies $\mathsf{val} \in R'$.

The algorithm may be easily extended such that sampling is also done once a region without samples is obtained: rather than inserting $(R', \emptyset)$ into Q, we insert the entry $(R', \textsc{sample}(R'))$.

Example 8.11 After several more iterations, the refinement loop started in Example 8.9 has proceeded to the state in Figure 8.8b. First, we see that the candidate region from Figure 8.8a was not rejecting. The verification engine gave a counterexample in form of an accepting sample (around $x \mapsto 0.45, y \mapsto 0.52$). Further iterations with smaller regions had some success, but some regions were inconsistent. From these failed verification invocations, additional samples were generated as counterexamples. The current blue candidate is to be checked next. In Figure 8.8c, we see a further continuation, with even smaller regions being verified. Note the white box on the right border: It has been checked, but the verification timed out without a conclusive answer. Therefore, we do not have a counterexample in this subregion.

Thus, the use of samples reduces region verification invocations. It remains to discuss some methods to split a region, and how for some of the constructed regions, verification may be skipped. We outline more details below.

8.4.2.1 How to split?

Splitting of regions, based on the existing samples, may be done using different schemes. We outline two basic approaches. These approaches may be easily mixed and extended, and their performance heavily depends on the concrete example at hand.

Equal splitting | This approach splits regions in equally-sized regions; the main rationale is that this generates small regions with nice bounds (the bounds are typically fractions whose denominators are powers of two). Splitting in equally sized regions may be done recursively: One projects all samples down to a single dimension and splits if both accepting and rejecting samples are in the region. The procedure halts if all samples in a region are either accepting or rejecting. The order in which parameters are considered plays a crucial role. Typically, we first split along the dimensions where the corresponding intervals are larger.

Example 8.12 A split in equally-sized regions is depicted in Figure 8.9b, where first the left region candidate is created. The remaining region can be split either horizontally or vertically to immediately generate another region candidate. A horizontal split in the remaining region yields a region without any samples.

The downside of equal splitting is that the position of the splits are not tailored to the

samples. Therefore, the number of splits might be significantly larger than necessary, increasing the number of region verification invocations.

Growing rectangles | This approach tries to gradually obtain a large region candidate[1]. The underlying rationale is to quickly cover vast amounts of the parameter space, as illustrated in Figure 8.9d (we adapted the samples for a consistent but concise description) where an initial sampling leads to a large rectangle as the region candidate.

> **Example 8.13** Consider the shaded regions in Figure 8.9c. Starting from vertex $v = (1, 1)$, the outer rectangle is maximised to not contain any accepting samples. Taking this outer rectangle as candidate region is very optimistic, it assumes that the accepting samples are on the border. A more pessimistic variant of growing rectangles is given by the inner shaded region. It takes a rejecting sample as vertex v' such that v and v' span the largest region.

Growing rectangles iterates over a subset of the hyper-rectangle's vertices: For each vertex (referred to as *anchor*), among all possible sub-hyper-rectangles containing the anchor and only accepting or only rejecting samples, the largest is constructed.

> **Example 8.14** The growing rectangles approach in its pessimistic fashion takes anchor $(0, 0)$ as anchor and yields the candidate region in Figure 8.9d.

The verification fails more frequently on large regions (either due to time-outs or due to the over-approximation). Consequently, choosing large candidate regions comes at the risk of failed verification invocations and fragmentation of the parameter space.

Furthermore, growing rectangles require a fall-back splitting technique: To see why, consider Figure 8.8c on page 234. The accepting (green) region does not contain any anchors of the full parameter space, therefore the hypothesis for any created subregion is rejection. In this particular setting, eventually one of the anchors of the created subregions might be accepting. However, if all edges of the initial region are rejecting (or accepting), no region with an accepting (rejecting) anchor will become a candidate.

8.4.2.2 Neighbourhood analysis

Besides considering samples within a region, we like to illustrate that the analysis of a region R may and should take information from outside of R into account. First, take Figure 8.9b, and assume that the left region is indeed accepting. The second generated region contains only rejecting samples, but it is only rejecting if all points, including all those on the border to the left region, are rejecting. In other words, the border between

[1] The approach shares its rationale with the approach formerly implemented in PROPHESY [3], but is realised slightly differently to overcome challenges for n-dimensional hyper-rectangles.

the accepting and rejecting regions needs to exactly follow the border between the generated region candidates, which is unlikely, and it is thus reasonable to split the second generated region. Secondly, a sensible hypothesis for candidate regions without samples inside is helpful to prevent excessive sampling. Typically, we then infer the hypothesis via the closest known region or sample.

8.4.3 Requirements on verification back-ends

In this section, we have described techniques for iteratively partitioning the parameter space into accepting and rejecting regions. The approaches rely on a verifier for regions (and sets of samples). Its usage imposes requirements on the verifier, most prominently:

- The verification should work *incrementally*: Verification results from previous iterations should be re-used in successive iterations. Verifying different regions share the same model (pMC or pMDP). A simple example of working incrementally is reusing a minimised model over various invocations. When checking subregions, the problem is even incremental in a more narrow sense: any bounds obtained for the super-region remain valid for the subregion.
- If the verification procedure fails, i.e., if the verifier returns `false`, obtaining additional *diagnostic information* in the form of a counterexample is beneficial. A simple counterexample is a sample refuting that the region is accepting (rejecting).

The full list is very similar to the typical requirements that theory solvers in lazy SMT frameworks should fulfil [BHMW09]. Therefore, SMT-based verification as in Section 8.2 naturally matches the wish-list. Typical heuristics, e.g. the approach from Chapter 12 may reuse the model, as discussed in sampling. Due to the approximative nature, many heuristics provide limited diagnostics.

8.4.4 Empirical evaluation

We give some empirical results on the benchmarks used in this chapter. The main claim is that *sampling-steered parameter space partitioning may be used successfully and profits from having a variety of possible techniques in the synthesiser and in the verifier.*

Figure 8.10a depicts the computation on the *Herman* (instance 4) and threshold $\lambda = 5$. The plot depicts the covered area for the three techniques discussed, with both quads (straight lines) and rectangles (dashed lines) as candidate region generation. The x-axis represents the computation time (in seconds) on a logarithmic scale and the y-axis represents the percentage of covered area. A point (x, y) in the plot thus represents y percent of the parameter space being covered within x seconds.

For *Herman* and this (carefully selected) threshold, SMT-based techniques perform better than PLA. PLA was able to cover 64% of the parameter space within milliseconds. However, in the remaining hour only 2% more space was covered. The SMT-based

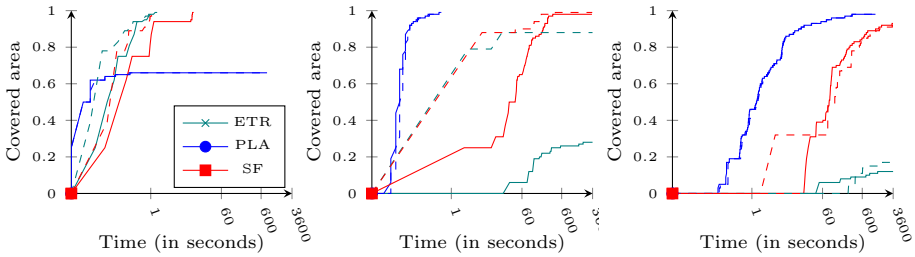


Figure 8.10: Performance of parameter space partitioning.

techniques were able to cover at least 99% of the parameter space within 15 seconds. Moreover, the rectangles cover the parameter space faster than quads. We also perform the parameter space partitioning on the *NAND* model (instance 7) with two different thresholds: We compare the parameter space partitioning techniques for threshold $\lambda = 0.1$ in Figure 8.10b, and for threshold $\lambda = 0.3$ in Figure 8.10c. For both instances, PLA performs better than the SMT-based techniques. For threshold $\lambda = 0.1$, PLA covered at least 99% of the parameter space within 1 second. The main reason is that the border is in a corner of the parameter space. Additionally, the SMT-based techniques with rectangles are significantly faster than the quads for this threshold. For threshold $\lambda = 0.3$, more region verification steps were necessary. PLA still outperforms ETR and SF. However, the use of rectangles over quads does not lead to a better performance for this threshold. At any point in time, there can be very significant differences between the heuristics for candidate generation, especially in settings where a single region verification invocation is expensive.

Finally, we summarise an overview of the performance in Table 8.4. For brevity, we pruned various rows. The *id* is a reference to the benchmark instance. The verification *technique* is given in the next column. In the next three columns we give for each technique the *time* (in seconds) needed to cover at least 50%, 90% and 98% of the complete parameter space. The next two columns give the complete *covered area*—i.e. the sum of the sizes of all accepting or rejecting regions— when terminating the parameter space partitioning after 1 hour, together with the *safe area*, i.e. the sum of the sizes of all accepting regions. The last two columns indicate the percentage of the total time spent in generating the regions (*time reg gen*) and verifying the regions (*time analysis*). PLA is almost always superior, but not on all benchmarks, and not on all (sub)regions. Depending on the model, SF or ETR are the best SMT-based technique. There might be room for improvement by heuristically varying the

Table 8.4: Computation of Parameter Space Partitioning

id	techn.	time 50%	time 90%	time 98%	area cov.	area safe	percent reg gen	percent analysis
1	DIR	—	—	—	0.47	0.00	0.00 %	99.64 %
	PLA	0.00	0.00	0.00	1.00	0.00	0.00 %	0.00 %
	SF	—	—	—	0.00	0.00	—	—
2	DIR	—	—	—	0.00	0.00	0.00 %	99.77 %
	PLA	0.01	0.01	0.01	1.00	1.00	0.00 %	0.91 %
	SF	—	—	—	0.00	0.00	—	—
3	DIR	0.02	30.19	70.41	0.99	0.05	0.00 %	98.89 %
	PLA	0.08	—	—	0.55	0.06	0.15 %	73.89 %
	SF	0.02	0.09	0.23	0.99	0.05	0.00 %	18.60 %
4	DIR	0.12	0.45	1.29	0.99	0.16	0.00 %	57.09 %
	PLA	0.03	—	—	0.66	0.17	0.15 %	74.84 %
	SF	0.24	1.20	11.30	0.99	0.16	0.00 %	90.63 %
6	DIR	—	—	—	0.00	0.00	0.00 %	99.66 %
	PLA	1.75	—	—	0.56	0.43	0.15 %	75.39 %
	SF	—	—	—	0.00	0.00	0.00 %	96.20 %
7	DIR	—	—	—	0.28	0.28	0.00 %	99.80 %
	PLA	0.05	0.09	0.49	0.99	0.98	8.22 %	15.53 %
	SF	28.70	202.98	357.90	0.98	0.98	0.00 %	96.26 %
9	DIR	—	—	—	0.00	0.00	—	—
	PLA	20.27	91.18	854.48	0.99	0.14	30.27 %	61.95 %
	SF	—	—	—	0.00	0.00	0.00 %	92.56 %
11	DIR	—	—	—	0.00	0.00	0.00 %	18.75 %
	PLA	8.79	40.99	326.12	0.99	0.16	33.23 %	54.62 %
	SF	—	—	—	0.00	0.00	0.00 %	94.39 %
12	DIR	—	—	—	0.00	0.00	—	—
	PLA	53.69	254.13	1861.31	0.99	0.16	33.21 %	60.37 %
	SF	—	—	—	0.00	0.00	—	—
13	DIR	—	—	—	0.00	0.00	0.00 %	99.28 %
	PLA	—	—	—	0.12	0.12	0.00 %	99.54 %
	SF	—	—	—	0.32	0.32	0.00 %	98.22 %

verification technique. Furthermore, partitioning in many dimensions, such as in the *Netw* are out-of-reach, at least with these general purpose techniques.

8.5 Bird's Eye View

In this section, we review the methods presented in this chapter and use their strengths and weaknesses to motivate the approaches that we detail in the next chapters.

Recap | Earlier in this chapter, we have seen methods to:

- find a closed-form representation for accepting (graph-preserving) valuations,
- find feasible parameter valuations, either via a complete method solving the underlying ETR encoding, or heuristically, based on successive sampling, and
- a scheme for computing an approximate parameter space partitioning—into accepting and rejecting valuations—developed using various empirical observations.

Finding a closed-form is unlikely to become applicable on a wider range of large models: As the size grows prohibitive, it becomes increasingly difficult to compute this function. Potentially worse, it becomes harder to analyse this function, let alone the raw ETR encoding. Finally, sampling in high-dimensional parameter spaces is like searching for a needle in the haystack and guidance is essential.

Despite those potential issues with scalability, we think that significant improvement is possible by extracting and incorporating information from the model and feeding these into the synthesiser—e.g., by induction or abstraction—and by enabling methods in the verifier to use and to provide this information. In the chapters below, we improve the state-of-the-art in parameter synthesis by combining the strengths of the early approaches in different fashions. Here, we summarise the underlying ideas.

Guided sampling | Tackling the feasibility problem with the ETR encoding currently puts *all information* about the pMDP to the solver, whereas the sampling approach only puts single instantiations to the model checker. While this abundance of information enables the solver to solve the system in one shot, the formula is hard to solve, especially without the context clarifying that this formula encodes a parametric Markov model. Instead, we propose to iteratively search for the solution by putting an abstraction of the formula to the solver. Thereby, we encode some information from the model but do not attempt to address the problem in one shot.

Concretely, in Chapter 9 we propose to use the synthesiser/verifier framework and use an approximation of the ETR encoding in the verifier. We provide the algorithm for feasibility under demonic nondeterminism. In particular, in the verifier, we first run standard sampling and then use an abstraction of the ETR problem into a *convex optimisation problem*. We use the solution of this convex optimisation problem to give *positive hints* for the next sample.

Guided pruning | So far, in the parameter space partitioning loop, we used rejecting samples to show that a candidate region is not accepting. Sometimes, the samples carry sufficient information to even identify a surrounding region as rejecting. We put this idea in a sampling loop: we provide *negative hints*, i.e., given a rejecting sample, we provide a rejecting region that safely may be ignored when searching for a feasible instantiation. Concretely, we consider a version in which these hints come with strong guarantees: none of the subregions contains an accepting valuation and the search space may be pruned accordingly by the synthesiser.

We realise this idea in Chapter 10 by adapting counterexample-guided inductive synthesis to fMCs. The extended verifier generates negative hints based on counterexamples—parts of the model sufficient to reject the property. We use a symbolic method, i.e., we focus on applying this method on sketches, rather than on fMCs.

Abstraction-based region checking | One reason that region verification is hard is due to the parameter dependencies and the intricate trade-offs. These dependencies may be abstracted away, and while this initially may be too coarse[1], the adaption of an abstraction-refinement scheme may help to yield sufficiently precise models that are still easier to check. We consider a refinement scheme based on divide-and-conquer: if abstracting all instantiations generated by some region is too coarse, we split the region into subregions and abstract these subregions.

We consider two variants: The first, in Chapter 11, targets fMCs. Formally, we construct an abstraction of an MDP that concisely represents a set of instantiations. For the refinement, we consider an adaption of the parameter space partitioning that we have seen in Section 8.4, now tailored to the discrete parameter space of fMCs and the interaction with our abstraction. The second variant, in Chapter 12, targets pMCs and pMDPs. The idea for pMCs is a straightforward extension to the family case[2] and relatively easy to understand, which is why we make the effort of explaining that first. We then show how to generalise this idea to pMDPs.

Basic methods for parameter synthesis have been around for some time and have been experimented extensively with, in particular, the computation of solution functions. Nevertheless, there are lots of open and unanswered questions about the practical application of these methods, and the validity of some rule-of-thumbs is heavily depending on the state-of-the-art in SMT solving and probabilistic model checking. Nevertheless, some derived insights yield ample motivation and inspiration for the development of new methods.

[1] Otherwise, we would not need parametric models.
[2] Chronologically, however, this method was the main inspiration for the work on families.

9 Convex Solvers for Feasibility

Iteratively solving convex over-approximations

Summary | Feasibility in pMCs and pMDPs is an ETR-hard problem, giving little hope for a scalable and general solution. However, as guessing one valuation right suffices to solve the problem, there is plenty of room for heuristics. This chapter develops such a heuristic by employing convex optimisation. First, we formulate the ETR formula as a *quadratically-constrained quadratic program* (QCQP). To allow for efficient solving techniques, we massage the QCQP formulation into a *difference-of-convex* (DC) problem. The *convex-concave procedure* (CCP) yields local optima of a DC problem by a convexification towards a convex quadratic program, amenable for state-of-the-art solvers such as Gurobi [Gur13]. Yet, black-box CCP solvers suffer from severe numerical issues and only solve small problems. Therefore, in the final part, we integrate the procedure with probabilistic model checking, creating a method that yields (1) a speedup of multiple orders of magnitude compared to just using CCP as a black box and (2) ensures the correctness of the solution.

Origins | This chapter resembles [11] with some extensions and clarifications.

Background | The chapter builds on top of the ETR-encoding for feasibility in pMDPs under demonic nondeterminism from Section 4.3.2. For convenience, the encoding is repeated at the start of the chapter. The notation is in line with the ETR encoding, which means that some notation is slightly unusual from an optimisation perspective.

9.1 QCQP Reformulation of Feasibility Synthesis

Within this chapter, we consider the following problem: Given an affine pMDP $\mathcal{M} = (S, \iota, Act, X, \mathcal{P})$ with target states T, a threshold λ, an $\varepsilon > 0$, and a property $\varphi := \mathbb{P}_{\leq\lambda}(\Diamond T)$, find an ε-preserving valuation $\mathsf{val} \in \mathsf{acc}^{\mathcal{M}}_{\forall,\varphi}$, i.e., find val such that $\mathcal{M}[\mathsf{val}] \forall\models \varphi$. An ε-preserving valuation assigns to every transition at least ε, see Definition 5.15. For conciseness, we assume that $\mathcal{P}(s, \alpha, s') \in \mathbb{Q}$ implies $\mathcal{P}(s, \alpha, s') \geq \varepsilon$. Furthermore, as convention $n := |S|$ and $k := |X|$. The method in this chapter is also applicable to expected rewards or lower bounds, with adaptions that we have seen for the ETR encodings in Section 4.3. We restrict ourselves to only one case for conciseness.

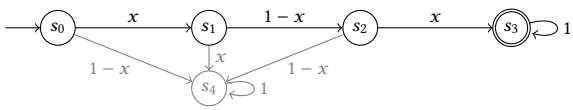


Figure 9.1: Example pMC

Recap of the encoding | We start with the ETR encoding for demonic feasibility on pMDPs, as discussed in Constraints 4.12 on page 98. A satisfying assignment to these constraints yields a valuation in $\mathsf{acc}^{\mathcal{M}}_{\forall,\varphi}$. As we are only interested in ε-preserving valuations, the constraints in Constraints 4.12 may be simplified by ideas from Section 4.3. Together with the constraints for ε-preserving valuations we obtain:

Constraints 9.1 Let $\mathcal{M}$ be a pMDP, $\varphi := \mathbb{P}_{\leq\lambda}(\Diamond T)$ a property, and some graph-preservation constant $\varepsilon \in \mathbb{R}_{>0}$. We consider the following constraint system with real-valued variables $\{p_s \mid s \in S_{>0} := S \setminus S^{\mathrm{gp},T}_{\exists=0}\}$ and variables for each parameter:

$$
\begin{aligned}
& p_\iota \leqslant \lambda, && \\
& \mathcal{P}(s,\alpha,s') \geqslant \varepsilon && \text{for all } s,s' \in S, \alpha \in \mathbb{A}\mathrm{ct}(s), \\
& \sum_{s' \in S} \mathcal{P}(s,\alpha,s') \doteq 1 && \text{for all } s \in S, \alpha \in \mathbb{A}\mathrm{ct}(s), \\
& p_s \doteq 1 && \text{for all } s \in T, \\
& p_s \geqslant \sum_{s' \in S_{>0}} \mathcal{P}(s,\alpha,s') \cdot p_{s'} && \text{for all } s \in S_{>0} \setminus T, \alpha \in \mathbb{A}\mathrm{ct}(s).
\end{aligned}
$$

Variables p_s thus encode that the probability from state s to the target is at least p_s. The constraints encode that the probability from state p_ι is at most λ, that all transitions have probability at least ε, that the outgoing transitions are a well-defined distribution over successor states, and encode the Bellman characterisation lifted to pMDPs.

The constraints also include a minor optimisation that allows for a concise notation below: No variables for the reachability from zero-states are added to the constraints. Instead, these variables are immediately substituted with zero. This substitution is reflected by summing over the non zero-state successors in the last constraint.

Example 9.1 We use the pMC in Figure 9.1 to illustrate the transformation of the encoding presented in this chapter. We keep λ as a constant, use p_i to denote p_{s_i}, and obtain the constraints:

$$p_0 \leqslant \lambda, \quad \varepsilon \leqslant x \leqslant 1-\varepsilon, \quad p_3 \doteq 1, \quad p_0 \geqslant x \cdot p_1, \quad p_1 \geqslant (1-x) \cdot p_2, \quad p_2 \geqslant x \cdot p_3.$$

QCQPs | In affine pMDPs all entries of $\mathcal{P}$ are linear and thus all constraints in Constraints 9.1 are quadratic. We rewrite the ETR formula as a standard form of a *quadratically-constrained quadratic program* (*QCQP*) [BV04]. In general, a QCQP is an optimisation problem with a quadratic objective function and m quadratic constraints, written in standard form: QCQP

Constraints 9.2 (QCQP standard form)

$$\begin{aligned}&\text{minimise} \quad \vec{v}^T A_0 \vec{v} + b_0^T \vec{v} + c_0 \quad \text{such that}\\ &\vec{v}^T A_i \vec{v} + b_i^T \vec{v} + c_i \leqslant 0 \quad \text{for all } i \in \{1, \dots, m\},\end{aligned}$$

where $\vec{v}$ is a vector of l *variables* and the coefficients are $A_i \in \mathbb{R}^{l \times l}$, $b_i \in \mathbb{R}^l$, $c_i \in \mathbb{R}$ for $0 \leq i \leq m$. We assume $A_0, \dots, A_m$ are symmetric without loss of generality.

Equality-constraints of the form $\vec{v}^T A_i \vec{v} + b_i^T \vec{v} + c_i = 0$ may be encoded by

$$\vec{v}^T A_i \vec{v} + b_i^T \vec{v} + c_i \leq 0 \text{ and } -\vec{v}^T A_i \vec{v} - b_i^T \vec{v} - c_i \leq 0 \ .$$

If $A_i = 0$ for all $0 \leq i \leq m$, the polynomials $b_i^T \vec{v} + c_i$ are *linear*, and the QCQP is in fact an LP. If every A_i is positive semi-definite (PSD), the function $\vec{v}^T A_i \vec{v} + q_i^T \vec{v} + r_i$ is *convex*, the QCQP is a *convex optimisation problem*, and may be solved in polynomial time [AG03]. If any A_i is not PSD, the resulting QCQP is nonconvex. Checking whether a quadratic constraint system is satisfiable is ETR-complete [Sch13], cf. Theorem B.1.

Reformulating feasibility as QCQP | To ease the presentation, we deviate from the standard ETR encoding and move closer to an optimisation formulation.

Constraints 9.3 The following constraints are a reformulation of Constraints 9.1 to the standard QCQP form (for the quadratic constraints) in Constraints 9.2[1]:

$$\begin{aligned}&\text{minimise} \quad p_\iota \quad \text{such that}\\ &p_\iota \leqslant \lambda, \\ &p_s \doteq 1 && \text{for all } s \in T,\\ &\mathcal{P}(s, \alpha, s') \geqslant \varepsilon && \text{for all } s, s' \in S, \alpha \in \mathbb{A}\text{ct}(s),\\ &\sum_{s' \in S} \mathcal{P}(s, \alpha, s') \doteq 1 && \text{for all } s \in S, \alpha \in \mathbb{A}\text{ct}(s),\\ &p_s \geqslant \vec{v}^T Q_{s,\alpha} \vec{v} + q_{s,\alpha}^T \vec{v}, && \text{for all } s \in S_{>0} \setminus T, \alpha \in \mathbb{A}\text{ct}(s),\end{aligned}$$

[1] We allow an affine non-zero right-hand side (rhs) for better readability. Naturally, the rhs may be set to zero by subtracting the rhs from the left-hand side (lhs). Furthermore, we have flipped lhs and rhs, as this notation is closer to the standard presentation of the equation system (and easier to read).

where $\vec{v}$ is a vector consisting of the probability variables p_s for all $s \in S$ and the pMDP parameters, i.e., $\vec{v}$ has $n + k$ rows. Furthermore, $Q_{s,\alpha} \in \mathbb{R}^{(n+k)\times(n+k)}$ is a symmetric matrix and $q_{s,\alpha} \in \mathbb{R}^{(n+k)}$.

Example 9.2 We reformulate the ETR encoding in Example 9.1. We omit matrix columns and matrix rows that only contain zeros to have a concise representation. Accordingly, the vectors are pruned.

$$\begin{aligned}
&\text{minimise} \quad p_\iota \quad \text{such that} \\
&\lambda \geqslant p_0, 1 - \varepsilon \geqslant x \geqslant \varepsilon, \quad p_3 \doteq 1, \\
&p_0 \geqslant \begin{bmatrix} x \\ p_1 \end{bmatrix}^T Q_0 \begin{bmatrix} x \\ p_1 \end{bmatrix}, \quad p_1 \geqslant \begin{bmatrix} x \\ p_2 \end{bmatrix}^T Q_1 \begin{bmatrix} x \\ p_2 \end{bmatrix} + p_2, \quad p_2 \geqslant \begin{bmatrix} x \\ p_3 \end{bmatrix}^T Q_2 \begin{bmatrix} x \\ p_3 \end{bmatrix}, \\
&\text{with} \quad Q_0 := \begin{bmatrix} 0 & 1/2 \\ 1/2 & 0 \end{bmatrix}, \quad Q_1 := \begin{bmatrix} 0 & -1/2 \\ -1/2 & 0 \end{bmatrix}, \quad Q_2 := \begin{bmatrix} 0 & 1/2 \\ 1/2 & 0 \end{bmatrix}.
\end{aligned}$$

Equivalently, but denoted more precisely, the quadratic constraints are:

$$\begin{aligned}
&\begin{bmatrix} x \\ p_1 \end{bmatrix}^T Q_0 \begin{bmatrix} x \\ p_1 \end{bmatrix} + \begin{bmatrix} 1 \end{bmatrix}^T \begin{bmatrix} p_0 \end{bmatrix} \leqslant 0, \\
&\begin{bmatrix} x \\ p_2 \end{bmatrix}^T Q_1 \begin{bmatrix} x \\ p_2 \end{bmatrix} + \begin{bmatrix} -1 \\ 1 \end{bmatrix}^T \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} \leqslant 0, \\
&\begin{bmatrix} x \\ p_3 \end{bmatrix}^T Q_2 \begin{bmatrix} x \\ p_3 \end{bmatrix} + \begin{bmatrix} 1 \end{bmatrix}^T \begin{bmatrix} p_2 \end{bmatrix} \leqslant 0.
\end{aligned}$$

We want to stress that $Q_0 \neq Q_2$, as the rows and columns that are zero (and not displayed here) are different.

In the example above, we magically constructed the right matrices. Below we clarify the (trivial) construction. We use matrices $Q_{s,\alpha}$ to capture the *quadratic* part and the vector $q_{s,\alpha}$ to capture the *affine* part of the constraints.

More precisely, consider an affine transition probability $a \cdot x + b := \mathcal{P}(s, \alpha, s')$ with $a, b \in \mathbb{R}$. The polynomial occurs in the constraint from Constraints 9.1 in $\mathcal{P}(s, \alpha, s') \cdot p_{s'}$. The quadratic part thus occurs as $a \cdot x \cdot p_{s'}$ and the affine part as $b \cdot p_{s'}$.

We focus on the product $\vec{v}^T Q_{s,\alpha} \vec{v}$, which denotes the sum over all products of entries in $\vec{v}$. Thus, in $Q_{s,\alpha}$, each row or column corresponds either to a probability variable p_s for a state $s \in S$ or to a parameter $v \in V$. Thus, the matrix cells indexed $(x, p_{s'})$ and $(p_{s'}, x)$ correspond to the product of parameter x with the state-variable $p_{s'}$. As the matrix $Q_{s,\alpha}$ needs to be symmetric, there are two entries in $Q_{s,\alpha}$ that represent the product of p_s and x. These two entries are summed in $\vec{v}^T Q_{s,\alpha} \vec{v}$. Thus, the coefficient a

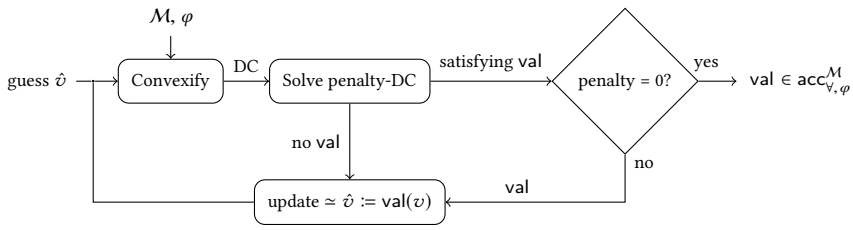


Figure 9.2: Penalty CCP loop

is represented by $a/2$ in both entries, making $Q_{s,\alpha}$ a symmetric matrix. For example, if $\mathcal{P}(s, \alpha, s') = 1/5 \cdot x + 1/3$, we insert $1/10$ in $Q_{s,\alpha}$ at the two entries for $x \cdot p_{s'}$. Similarly, we construct $q_{s,\alpha}$ by setting the entry for $p_{s'}$ to b. We do not modify the affine functions for the QCQP form.
The following observation is not surprising, as the feasibility problem is ETR-hard.

Proposition 9.1 The QCQP in Constraints 9.3 is nonconvex in general.

Proof. The matrices Q_0, Q_1, Q_2 in Example 9.2 have an eigenvalue of $-1/2$ and are not PSD. Thus, the constraints and the resulting QCQP are nonconvex. ■

9.2 Efficiently Solving the QCQP

The QCQP in Constraints 9.3 is nonconvex and hard to solve in general. We provide a solution by employing a heuristic called the *convex-concave procedure* (*CCP*) [LB16], CCP
which relies on the ability to efficiently solve convex optimisation problems. Later in this section, we improve this method by employing probabilistic model checking.

9.2.1 A CCP approach to the feasibility problem

Overview | Roughly, the CCP approach for the feasibility problem is captured by Figure 9.2. It is a search for a solution of the nonconvex QCQP, that literately solves approximations of the QCQP in form of convex DC problems around some initial $\hat{v} \in \mathbb{R}^l$. For any satisfying valuation of such a DC problem, we may check some penalty-value of this solution. If this penalty is zero, we have found a solution to the original nonconvex QCQP. Otherwise, we update (or guess) a new $\vec{v}$ which we use to convexify the QCQP. This loop may get stuck, which requires restarts. We start with the construction of the (penalty-)DP, then discuss the update function, and finally discuss how to tune this method by an improved method for generating the DP. We also include some remarks on how we selected the numerous constants in the method.

DC reformulation | The CCP computes a *local optimum* of a nonconvex *difference-of-convex* (*DC*) problem.

DC

Constraints 9.4 (DC standard form with polynomial constraints)

$$\begin{aligned} &\text{minimise} \quad f_0 - g_0 \quad \text{such that} \\ &f_i - g_i \leqslant 0 \quad \text{for all } 1 \leq i \leq m, \end{aligned}$$

with convex $f_i, g_i \in \mathbb{Q}[\vec{v}]$ for $0 \leq i \leq m$.

The polynomials $-g_i$ are *concave*. Every quadratic polynomial can be rewritten as a DC polynomial, i.e., as difference of two convex polynomials. Consider the quadratic polynomial $\vec{v}^T Q_{s,\alpha} \vec{v} + q_{s,\alpha}^T \vec{v}$ from Constraints 9.3. We decompose the matrix $Q_{s,\alpha}$ into the difference of two matrices

$$Q_{s,\alpha} = Q_{s,\alpha}^+ - Q_{s,\alpha}^- \quad \text{with} \quad Q_{s,\alpha}^+ := Q_{s,\alpha} + tI \text{ and } Q_{s,\alpha}^- := tI, \tag{9.1}$$

where I is the identity matrix and $t \in \mathbb{R}_{>0}$ is sufficiently large to render $Q_{s,\alpha}^+$ PSD, i.e., larger than the largest eigenvalue of $Q_{s,\alpha}$. Then, we rewrite

$$\vec{v}^T Q_{s,\alpha} \vec{v} + q_{s,\alpha}^T \vec{v} \leqslant 0 \rightsquigarrow \underbrace{\vec{v}^T Q_{s,\alpha}^+ \vec{v} + q_{s,\alpha}^T \vec{v}}_{f_i} - \underbrace{\vec{v}^T Q_{s,\alpha}^- \vec{v}}_{g_i} \leqslant 0,$$

which is in the form of Constraints 9.4. Together, we obtain the following constraints:

Constraints 9.5 (DC for feasibility) The following constraints are a reformulation of Constraints 9.1 to the standard DC form in Constraints 9.4:

$$\begin{aligned} &\text{minimise} \quad p_\iota \quad \text{such that} \\ &p_\iota \leqslant \lambda, \\ &p_s \doteq 1 && \text{for all } s \in T, \\ &\mathcal{P}(s, \alpha, s') \geqslant \varepsilon && \text{for all} s, s' \in S, \alpha \in \mathbb{A}\text{ct}(s), \\ &\sum_{s' \in S} \mathcal{P}(s, \alpha, s') \doteq 1 && \text{for all } s \in S, \alpha \in \mathbb{A}\text{ct}(s), \\ &p_s \geqslant \vec{v}^T Q_{s,\alpha}^+ \vec{v} + q_{s,\alpha}^T \vec{v} - \vec{v}^T Q_{s,\alpha}^- \vec{v} && \text{for all } s \in S_{>0} \setminus T, \alpha \in \mathbb{A}\text{ct}(s). \end{aligned}$$

The reformulations so far are equivalences, formalised by the following corollary.

Corollary 9.2 The solutions for Constraints 9.1 and Constraints 9.5 coincide.

Example 9.3 Recall the pMC in Figure 9.1 and the QCQP from Example 9.2. All matrices Q_s in the QCQP are not PSD. We construct a DC problem with $t = 1$ for all Q_s:

$$
\begin{aligned}
&\text{minimise} \quad p_0 \quad \text{such that} \\
&\lambda \geqslant p_0, \quad 1 - \varepsilon \geqslant x \geqslant \varepsilon, \quad p_3 \doteq 1, \\
&p_0 \geqslant \begin{bmatrix} x \\ p_1 \end{bmatrix}^T \begin{bmatrix} 1 & 1/2 \\ 1/2 & 1 \end{bmatrix} \begin{bmatrix} x \\ p_1 \end{bmatrix} - \begin{bmatrix} x \\ p_1 \end{bmatrix}^T \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ p_1 \end{bmatrix}, \\
&p_1 \geqslant \begin{bmatrix} x \\ p_2 \end{bmatrix}^T \begin{bmatrix} 1 & -1/2 \\ -1/2 & 1 \end{bmatrix} \begin{bmatrix} x \\ p_2 \end{bmatrix} - \begin{bmatrix} x \\ p_2 \end{bmatrix}^T \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ p_2 \end{bmatrix} + p_2, \\
&p_2 \geqslant \begin{bmatrix} x \\ p_3 \end{bmatrix}^T \begin{bmatrix} 1 & 1/2 \\ 1/2 & 1 \end{bmatrix} \begin{bmatrix} x \\ p_3 \end{bmatrix} - \begin{bmatrix} x \\ p_3 \end{bmatrix}^T \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ p_3 \end{bmatrix}.
\end{aligned}
$$

Penalty CCP for DP | For the obtained DC problem, we consider the iterative *penalty CCP method* [LB16]. The procedure is initialised with any initial assignment $\hat{v}$ for the variables $v \in \vec{v}$. In the *convexification* stage, we compute affine approximations in form of a linearisation of g_i around $\hat{v}$:

$$\tilde{g}_i := g_i[\hat{v}] + \nabla g_i[\hat{v}]^T \cdot (\vec{v} - \hat{v}), \tag{9.2}$$

where ∇g_i is the gradient of the polynomial g_i at $\hat{v}$. The polynomial $f_i - \tilde{g}_i$ is clearly convex. The gradient of $\vec{v}^T Q^-_{s,\alpha} \vec{v}$ is $2 \cdot Q^-_{s,\alpha} \hat{v}$.

Constraints 9.6 (Over-approximating DC for feasibility) For a pMDP and property as before and some valuation $\hat{v}$ for *all* variables in the DC, consider the following optimisation problem:

$$
\begin{aligned}
&\text{minimise} \quad p_\iota \quad \text{such that} \\
&p_\iota \leqslant \lambda, \\
&p_s \doteq 1 && \text{for all } s \in T, \\
&\mathcal{P}(s, \alpha, s') \geqslant \varepsilon && \text{for all } s, s' \in S, \alpha \in \mathbb{A}\mathrm{ct}(s), \\
&\sum_{s' \in S} \mathcal{P}(s, \alpha, s') \doteq 1 && \text{for all } s \in S, \alpha \in \mathbb{A}\mathrm{ct}(s), \\
&p_s \geqslant \vec{v}^T Q^+_{s,\alpha} \vec{v} + q^T_{s,\alpha} \vec{v} - \hat{v}^T Q^-_{s,\alpha} (2\vec{v} - \hat{v}) && \text{for all } s \in S_{>0} \setminus T, \alpha \in \mathbb{A}\mathrm{ct}(s).
\end{aligned}
$$

The following proposition clarifies the meaning of the over-approximation:

Proposition 9.3 A satisfying assignment for Constraints 9.6 is also satisfying Constraints 9.5.

Proof (sketch). Using the notation of Constraints 9.4 and equation (9.2): It holds that:

$$f_i - g_i \leq f_i - \tilde{g}_i.$$

This inequality is the direct consequence of the following standard result on convex functions[1]: Let $f \in \mathbb{Q}[\boldsymbol{x}]$ be convex. For $\mathsf{val}, \mathsf{val}' \in \mathsf{Val}$,

$$f[\mathsf{val}'] \geq f[\mathsf{val}] + \nabla f[\mathsf{val}](\mathsf{val}' - \mathsf{val}).$$

We thus obtain $g_i \geq \tilde{g}_i$ for all $1 \leq i \leq m$, and consequently $-g_i \leq -\tilde{g}_i$. Then, $f_i - g_i \leq f_i - \tilde{g}_i$. From that result, it is clear that any val that satisfies

$$p_s \geqslant \vec{v}^T Q^+_{s,\alpha} \vec{v} + q^T_{s,\alpha} \vec{v} - \tilde{v}^T Q^-_{s,\alpha}(2\vec{v} - \tilde{v})$$

also satisfies

$$p_s \geqslant \vec{v}^T Q^+_{s,\alpha} \vec{v} + q^T_{s,\alpha} \vec{v} - \vec{v}^T Q^-_{s,\alpha} \vec{v}.$$

The claim follows, as all other constraints remain untouched. ■

The stricter set of constraints is often too strict and may not have any feasible solution. To loosen the constraints, a *penalty variable* $\delta_{s,\alpha}$ for all $s \in S \setminus T$ and $\alpha \in Act$ is added to all convexified constraints. These variables allow us to measure the 'amount' of infeasibility. The larger the assigned value for a penalty variable, the larger the violation. We then seek to minimise the violation of the original DC constraints by minimising the sum of the penalty variables. The resulting convex optimisation problem is:

Constraints 9.7 (penalty-DC for feasibility) For an approximating DC over variables as in Constraints 9.6, a constant penalty constant $\tau \in \mathbb{R}_{>0}$, and additional *penalty variables* $\{\delta_{s,\alpha} \mid s \in S, \alpha \in \mathbb{A}\mathrm{ct}(s)\}$ we formulate the following constraint system:

$$\begin{aligned}
&\text{minimise} \quad p_\iota + \tau \cdot \sum_s \sum_\alpha \delta_{s,\alpha} \quad \text{such that} \\
&p_\iota \leqslant \lambda, \\
&p_s \doteq 1 && \text{for all } s \in T, \\
&\delta_{s,\alpha} \geqslant 0 && \text{for all } s \in S, \alpha \in \mathbb{A}\mathrm{ct}(s), \\
&\mathcal{P}(s, \alpha, s') \geqslant \varepsilon && \text{for all } s, s' \in S, \alpha \in \mathbb{A}\mathrm{ct}(s), \\
&\sum_{s' \in S} \mathcal{P}(s, \alpha, s') \doteq 1 && \text{for all } s \in S, \alpha \in \mathbb{A}\mathrm{ct}(s), \\
&p_s + \delta_{s,\alpha} \geqslant \vec{v}^T Q^+_{s,\alpha} \vec{v} + q^T_{s,\alpha} \vec{v} - \hat{v}^T Q^-_{s,\alpha}(2\vec{v} - \hat{v}) && \text{for all } s \in S_{>0} \setminus T, \alpha \in \mathbb{A}\mathrm{ct}(s).
\end{aligned}$$

[1]Readers unfamiliar with this result may want to draw the derivative of a univariate convex function.

Example 9.4 We continue with Example 9.3 and bring the constraints into the form of Constraints 9.7. We assume some valuation $\hat{v}$ for each $v \in \{x, p_0, p_1, p_2, p_3\}$.

$$
\begin{aligned}
&\text{minimise} \quad p_0 + \tau \sum_{i=0}^{2} \delta_i \quad \text{such that} \\
&\lambda \geqslant p_0, \quad 1 - \varepsilon \geqslant x \geqslant \varepsilon, \quad p_3 \doteq 1 \\
&\delta_0 + p_0 \geqslant \begin{bmatrix} x \\ p_1 \end{bmatrix}^T \begin{bmatrix} 1 & 1/2 \\ 1/2 & 1 \end{bmatrix} \begin{bmatrix} x \\ p_1 \end{bmatrix} - \begin{bmatrix} \hat{x} \\ \hat{p_1} \end{bmatrix}^T \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \cdot x - \hat{x} \\ 2 \cdot p_1 - \hat{p_1} \end{bmatrix}, \\
&\delta_1 + p_1 \geqslant \begin{bmatrix} x \\ p_1 \end{bmatrix}^T \begin{bmatrix} 1 & -1/2 \\ -1/2 & 1 \end{bmatrix} \begin{bmatrix} x \\ p_2 \end{bmatrix} - \begin{bmatrix} \hat{x} \\ \hat{p_2} \end{bmatrix}^T \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \cdot x - \hat{x} \\ 2 \cdot p_2 - \hat{p_2} \end{bmatrix} + p_2, \\
&\delta_2 + p_2 \geqslant \begin{bmatrix} x \\ p_3 \end{bmatrix}^T \begin{bmatrix} 1 & 1/2 \\ 1/2 & 1 \end{bmatrix} \begin{bmatrix} x \\ p_3 \end{bmatrix} - \begin{bmatrix} \hat{x} \\ \hat{p_3} \end{bmatrix}^T \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \cdot x - \hat{x} \\ 2 \cdot p_3 - \hat{p_3} \end{bmatrix}, \\
&\delta_0 \geq 0, \quad \delta_1 \geq 0, \quad \delta_2 \geq 0.
\end{aligned}
$$

If all penalty variables are assigned to zero, we may terminate the loop in Figure 9.2.

Theorem 9.4 For a pMDP $\mathcal{M}$, property φ, and corresponding convex DC problem Constraints 9.7, it holds that a satisfying assignment val of the DC problem with

$$\Big(\tau \sum_{\forall s \in S \setminus T} \sum_{\forall \alpha \in Act} \delta_{s,\alpha}\Big)[\mathsf{val}] = 0$$

accepts φ on $\mathcal{M}$ (under demonic nondeterminism).

Proof (sketch). Let val be a satisfying valuation as in the theorem. With the side condition on the penalty variables, the constraints in Constraints 9.7 reduce to the constraints in Constraints 9.6. From Proposition 9.3 we conclude that the valuation is also satisfying the constraints from Constraints 9.5. Via Corollary 9.2, the valuation satisfies the constraints from Constraints 9.1. Using the results from Section 4.3, the valuation accepts φ on $\mathcal{M}$. The technical report of [11] contains an alternative proof.■

Update | If any of the penalty variables are assigned to a positive value, we update the penalty parameter τ by $\mu + \tau$ for a $\mu > 0$, until an upper limit for τ_{max} is reached to avoid numerical problems. Then, we compute a linearization of the g_i functions around the current (not feasible) solution and solve the resulting problem. This procedure is repeated until we find a feasible solution. The procedure may get stuck: Small violations of the original over-approximating DC, i.e., small $\delta_{s,\alpha}$ values may lead

to big differences in the actual computed p_ι. Then, the CCP may be stuck in poor local optima, where the sum of constraint violations is small, but the violation for the probability threshold is too large. We circumvent the problem as follows: if the procedure converges to an infeasible solution, it may be restarted with adapted initial values $\hat{v}$.

Eigenvalue-less DP | We may use the transformation in Equation (9.1), but it involves expensive matrix operations, including computing the numerous eigenvalues to make the matrix PSD. Observe that the matrices $Q_{s,\alpha}$ and vectors $q_{s,\alpha}$ are sparse. Then, the eigenvalue method introduces more occurrences of the variables in every constraint and thereby increases the approximation error during convexification.

We use an alternative convexification which is based on the observation that the quadratic monomials are bilinear—state variables and parameters are never squared. Consider the bilinear polynomial $h \coloneqq 2c \cdot yz$, where y and z are variables and $c \in \mathbb{R}_{>0}$. We rewrite h equivalently using

$$h \rightsquigarrow h + c(y^2 + z^2) - c(y^2 + z^2).$$

Then, we rewrite $h + c(y^2 + z^2)$ as $2c \cdot yz + c(y^2 + z^2) = c(y + z)^2$. We obtain

$$h = c(y + z)^2 - c(y^2 + z^2).$$

The function $c(y+z)^2$ is a quadratic convex polynomial and we convexify the polynomial $-c(y^2 + z^2)$ as $-c(\hat{y}^2 + \hat{z}^2) + 2c(\hat{y}^2 + \hat{z}^2 - y\hat{y} - z\hat{z})$, where $\hat{y}$ and $\hat{z}$ are the assignments as before. Together, this yields the following reformulation into a convexified constraint:

$$2c \cdot yz \rightsquigarrow c(y + z)^2 - c(\hat{y}^2 + \hat{z}^2) + 2c(\hat{y}^2 + \hat{z}^2 - y \cdot \hat{y} - z \cdot \hat{z}).$$

We convexify the bilinear polynomial $h = 2c \cdot yz$ with $c \in \mathbb{R}_{<0}$ analogously by adding and subtracting the concave function $-c(y^2+z^2)$ to rewrite $2cyz$ as $-c(y-z)^2+c(y^2+z^2)$. Consequently, we reduce the occurrences of variables for sparse matrices compared to the eigenvalue method.

Tuning magic constants. | Optimally, we would initialise the CCP procedure, i.e., values $\hat{v}$ (for the parameters and the state variables) with a feasible valuation, but that would require to already solve the feasibility problem itself. Instead, we instantiate the $\hat{x}$ components of $\hat{v}$ as the centre of the parameter space, and thereby minimise the worst-case distance to a feasible solution. For $\hat{p_s}$, we use the threshold λ from the specification to initialise the probability variables, and analogously for expected cost. We initialise the penalty constant $\tau \coloneqq 0.05$ for reachability, and $\tau \coloneqq 5$ for expected cost, a conservative number in the same order of magnitude as the values $\hat{p_s}$. As expected cost evaluations have wider ranges than probability evaluations, a larger τ is sensible. We pick $\mu \coloneqq \max_{s \in S \setminus T} \hat{p_s}$. We update τ by adding μ after each

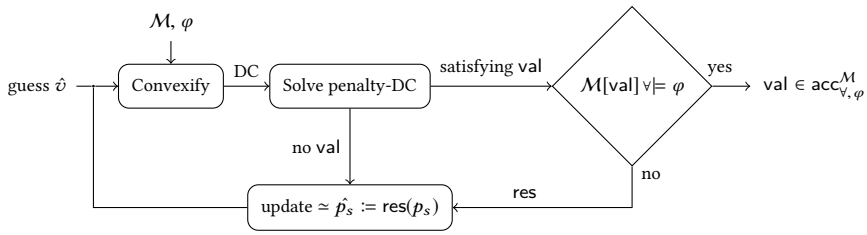


Figure 9.3: CCP and model checking in the loop

iteration. Empirically, increasing τ with bigger steps is beneficial for the run time, but induces more numerical instability. In contrast, in the literature, the update parameter μ is frequently used as a constant, i.e., it is not updated between the iterations. In, e.g, [LB16], τ is multiplied by μ after each iteration.

9.2.2 Integration with model checking

Model checking in the loop | The first and foremost assumption, backed by experiments, is that model checking single instantiations of a pMDP is much faster than an iteration of the CCP. Consequently, we slightly change the loop from Figure 9.2 to the loop in Figure 9.3, making two changes. The first change is that we no longer check the penalty of a solution. Instead, we use the values $\mathsf{val}(x)$ for each $x \in X$, which give rise to a parameter valuation. Model checking at these instantiations has two benefits: First, it allows for *early termination*. We check whether $\mathcal{M}[\mathsf{val}]$ accepts φ: If yes, we have found a solution even though the penalty variables have not converged to zero. Second, the model checking procedures are numerically more stable and allow for exact arithmetic. Thereby, the solutions are more reliable than the solutions obtained from the convex solver.

The second change is in the update procedure. Using the model checking results often overcomes convergence to infeasible solutions. These problems may be described as follows: Instead of instantiating the initial probability value for the next iteration as $\hat{p_s} := \mathsf{val}(p_s)$, we may use the model-checking result res of the MDP $\mathcal{M}[\mathsf{val}]$ and set $\hat{p_s} := \mathsf{res}(s)$. Model checking ensures that the probability variables are consistent with the parameter variables, i.e., that the constraints describing the transition relation in the original encoding are all satisfied.

Algorithmic improvements | We list three technical improvements that significantly improved the performance of the approach. (1) We precompute zero-states and one-states and simplify the constraints accordingly. (2) For simple(x) pMDPs, the well-formed constraints are written compactly. (3) Parts of the encoding are untouched

over multiple CCP iterations. Instead of rebuilding the encoding, we only update constraints which contain iteration-dependent values. The update is based on a preprocessed representation of the model. The improvement is two-fold: We spend less time constructing the encoding and the solver reuses previous results, making solving up to three times faster.

9.3 Empirical Evaluation

9.3.1 Set-up

Implementation | We implement the CCP with the above mentionend efficiency improvements in the parameter synthesis framework PROPHESY. We use the probabilistic model checker STORM [9] to extract an explicit representation of a pMDP. We keep the pMDP in memory and update the parameter instantiations using an efficient data structure to enable efficient repetitive model checking. To solve convex QCQPs, we use GUROBI 7.5 [Gur13], configured for numerical stability.

Environment | We evaluate on a HP BL685C G7 with 48 2 GHz cores, a 32 GB memory limit, and 1800 seconds time limit; the implementation uses a single thread.

Settings | We consider feasibility, i.e., the task is to find feasible parameter valuation for pMCs and pMDPs with non-trivial upper/lower thresholds on probabilities/rewards in the property. We ask for a ε-preserving valuation of the parameters, with $\varepsilon := 10^{-5}$. We run all the approaches with the exact same configuration of STORM. For pMCs, we enable weak bisimulation, which is beneficial for all presented examples. We do not use bisimulation for pMDPs.

Base line | We compare runtimes with a particle-swarm optimisation (PSO, Section 8.3) and the SMT-based approaches (direct or via solution function, Section 8.2). As PSO is a randomised procedure, we run it with random seeds 0–19. The PSO implementation requires the well-defined parameter regions to constitute a hyper-rectangle, as proper sampling from polygons is a non-trivial task.

We additionally compare against a prototype of the geometric programming (GP) approach [8] based on CvxPy [DB16] and the solver SCS [OCPB16], and the QCQP package [PB17], which implements several heuristics, including a naive CCP approach, for nonconvex QCQPs. Due to numerical instabilities, we failed to automatically and successfully apply these two approaches to a wide range of benchmarks.

Benchmarks | We include the standard pMC benchmarks from the PARAMwebsite, which contain two parameters. We consider a large set of POMDP strategy synthesis

Table 9.1: pMC benchmark results

Problem			Info			PSO			SMT	CCP		
Set	Inst	Spec	States	Trans.	Par.	tmin	tmax	**tavg**	**t**	**t**	solv	iter
BRP	16,2	$\mathbb{P}_{\leq 0.1}(\Diamond)$	98	194	**2**	0	0	**0**	40	0	30%	3
BRP	512,5	$\mathbb{P}_{\leq 0.1}(\Diamond)$	6146	12290	**2**	24	36	**28**	TO	33	24%	3
Crowds	10,5	$\mathbb{P}_{\leq 0.1}(\Diamond)$	42	82	**2**	4	5	5	8	**4**	2%	4
Nand	5,10	$\mathbb{P}_{\leq 0.05}(\Diamond)$	10492	20982	**2**	21	51	28	TO	**22**	21%	2
Zeroconf	10000	$\mathbb{E}_{\leq 10010}(\Diamond)$	10003	20004	**2**	2	4	**3**	TO	57	81%	3
GridA	4	$\mathbb{P}_{\geq 0.84}(\Diamond)$	1026	2098	**72**	11	11	**11**	TO	22	81%	11
GridB	8,5	$\mathbb{P}_{\geq 0.84}(\Diamond)$	8653	17369	**700**	409	440	427	TO	**213**	84%	8
GridB	10,6	$\mathbb{P}_{\geq 0.84}(\Diamond)$	16941	33958	**1290**	533	567	553	TO	**426**	84%	7
GridC	6	$\mathbb{E}_{\leq 4.8}(\Diamond)$	1665	305	**168**	261	274	267	TO	**169**	90%	23
Maze	5	$\mathbb{E}_{\leq 14}(\Diamond)$	1303	2658	**590**	213	230	219	TO	**67**	89%	8
Maze	5	$\mathbb{E}_{\leq 6}(\Diamond)$	1303	2658	**590**	–	–	TO	TO	**422**	85%	97
Maze	7	$\mathbb{E}_{\leq 6}(\Diamond)$	2580	5233	**1176**	–	–	TO	TO	**740**	90%	60
Netw	5,2	$\mathbb{E}_{\leq 11.5}(\Diamond)$	21746	63158	**2420**	312	523	359	TO	**207**	39%	3
Netw	5,2	$\mathbb{E}_{\leq 10.5}(\Diamond)$	21746	63158	**2420**	–	–	TO	TO	**210**	38%	4
Netw	4,3	$\mathbb{E}_{\leq 11.5}(\Diamond)$	38055	97335	**4545**	–	–	TO	TO	MO	-	-
Repud	8,5	$\mathbb{P}_{\geq 0.1}(\Diamond)$	1487	3002	**360**	16	22	18	TO	**4**	36%	2
Repud	8,5	$\mathbb{P}_{\leq 0.05}(\Diamond)$	1487	3002	**360**	273	324	293	TO	**14**	72%	4
Repud	16,2	$\mathbb{P}_{\leq 0.01}(\Diamond)$	790	1606	**96**	–	–	TO	TO	**15**	78%	9
Repud	16,2	$\mathbb{P}_{\geq 0.062}(\Diamond)$	790	1606	**96**	–	–	TO	TO	TO	-	-

problems from Section 7.1: *GridX* are gridworld problems with trap states (A), finite horizons (B), or movement costs (C). *Maze* is a navigation problem. *Netw* and *Repud* originate from distributed protocols, and were included in [NPZ17]. We obtain the pMDP benchmarks either from the PARAM website, or as parametric variants to existing PRISM case studies, and describe randomised distributed protocols.

9.3.2 Evaluation

9.3.2.1 Experiments

Results | Table 9.1 contains an overview of the results for pMCs. The first two columns refer to the benchmark instance, the next column to the specification evaluated. We give the states (States), transitions (Trans.) and parameters (Par.) *in the bisimulation quotient*, which is then used for further evaluation. We then give the *minimum* (tmin), the *maximum* (tmax) and *average* (**tavg**) runtime (in seconds) for PSO with different seeds, the best runtime obtained using SMT (**t**), and the runtime for CCP (**t**). For CCP, we additionally give the fraction (in percent) of time spent in Gurobi (solv), and the number of CCP iterations (iter). Table 9.2 reports on the results for pMDPs and additionally contains the number of actions (Act) for pMDPs. The boldfaced measures **tavg** and **t** for both SMT and CCP are the important measures to compare. Boldface values are the ones with the best performance for a specific benchmark.

Table 9.2: pMDP benchmark results

Problem			Info				PSO			SMT	CCP		
Set	Inst	Spec	States	Act	Trans.	Par.	tmin	tmax	**tavg**	**t**	**t**	solv	iter
BRP	4,128	$\mathbb{P}_{\leq 0.1}(\Diamond)$	17131	17396	23094	**2**	45	47	46	TO	**39**	33%	4
Coin	32	$\mathbb{E}_{\leq 500}(\Diamond)$	4112	6160	7692	**2**	117	119	**118**	TO	TO	-	-
CoinX	32	$\mathbb{E}_{\leq 210}(\Diamond)$	16448	24640	30768	**2**	1196	1222	1208	TO	**32**	78%	3
Zeroconf	1	$\mathbb{P}_{\geq 0.99}(\Diamond)$	31402	55678	70643	**3**	18	19	**19**	TO	79	82%	2
CSMA	2,4	$\mathbb{E}_{\leq 69.3}(\Diamond)$	7958	7988	10594	**26**	n.s.	n.s.	n.s.	TO	**79**	86%	10
Virus	-	$\mathbb{E}_{\leq 10}(\Diamond)$	809	3371	6741	**18**	113	113	113	TO	**13**	76%	4
Wlan	0	$\mathbb{E}_{\leq 580}(\Diamond)$	2954	3972	5202	**15**	n.s.	n.s.	n.s.	TO	**7**	72%	2

Interpretation | There is a constant overhead for model building, which is in particular large if the bisimulation quotient computation is expensive, see the small fraction of time spent solving DPs for *Crowds*. For the more challenging instances, this overhead is negligible. Roughly 80–90% of the time is spent within Gurobi for these instances, the remainder is used to feed the DPs into Gurobi. A specification threshold closer to the (global) optimum typically induces a higher number of iterations (see *Maze* or *Netw* with different thresholds). For the pMDP *Coin*, optimal parameter values are on the boundary of the parameter space and are quickly reached by PSO. The small parameter values together with the rather large expected costs are numerically challenging for CCP. For *CoinX*, the parameter values are in the interior of the parameter space and harder to hit via sampling. For CCP, the difference between small and large coefficients is smaller than in *Coin*, which yields better convergence behaviour. The benchmarks *CSMA* and *WLAN* are currently not supported by PSO due to the non-rectangular well-defined parameter space. CCP does not solve all instances: In *Netw* (4,3), CCP exceeds the memory limit. In *Repud*, finding values close the global optimum requires too much time. While the thresholds used here are close to the global optima, actually finding the global optimum itself is always challenging.

Effect of integrating model checking for CCP | A naive CCP method without model checking mostly yields an inferior performance for our synthesis problem.

The benchmark-set *Maze* profits most: Discarding the model checking results in our CCP implementation always yields time-outs, even for the rather simple *Maze* with threshold 14, which is solved with usage of model checking results within 70 seconds. Here, using model checking results thus yields a speed-up by a factor at least 25. More typical examples are *Netw*, where discarding the model checking results yields a factor five performance penalty. The *Repud* examples do not significantly profit from using intermediate model checking results.

Evaluation of the QCQP package, GP and SMT | We evaluate the GP on pMCs with two parameters: For the smaller *BRP* instance, the procedure takes 90 seconds,

for *Crowds* 14 seconds. Other instances yield timeouts. We also evaluate the QCQP package on some pMCs. For the smaller *BRP* instance, the package takes 113 seconds, and 13 seconds for the *Crowds*. For a *Repud* instance with 44 states, and 26 parameters, the package takes 54 seconds and returns a rejecting valuation. Our CCP-loop with integrated model checking takes less than a second.

The results in Table 9.1 and 9.2 make obvious that *SMT is not competitive*, irrespectively whether the formula is preprocessed via state elimination. Moreover, state elimination (for pMCs) within the given time limit *is only possible for the (considered) models with 2 parameters*, using either PRISM, PARAM, or STORM.

9.3.2.2 Discussion

Based on the empirical evaluation, we make the following claims.

A tuned variant of CCP improves the state-of-the-art. | Just applying out-of-the-box heuristics for QCQPs—like realised in the QCQP package or using our CCP implementation without integrated model checking—does not yield a scalable method. To solve the nonconvex QCQP, we require a CCP with a clever encoding and several algorithmic improvements. State space reductions shrink the encoding and model checking after each CCP iteration to terminate earlier typically saves 20% of iterations. Especially when convergence is slow, model checking saves significantly more iterations. Moreover, feeding model checking results into the CCP improves the runtime by up to an additional order of magnitude, at negligible costs. These combined improvements to the CCP method outnumbers any solver-based approach by orders of magnitude and is often superior to sampling based approaches, especially in the presence of many parameters. Benchmarks with many parameters pose two challenges for sampling based approaches: Sampling is necessarily sparse due to the high dimension and optimal parameter valuations for one parameter often depend significantly on other parameter values.

CCP performance can be boosted with particular benchmarks in mind. | For most benchmarks, choosing larger values for τ and μ improves the performance. Furthermore, for particular benchmarks, we can find a better initial valuation $\hat{v}$. These adaptions, however, are not suitable for a general framework. We have chosen magic constants that reflect a more balanced performance over several types of benchmarks. On the downside, the dependency on the constants means that minor changes in the encoding may have a significant, but hard to predict, effect. For SMT-solvers, additional and superfluous constraints often help steering the solver, but in the context of CCP, it deteriorates the performance.

Some benchmarks constitute numerically challenging problems. | For specification thresholds close to global optima and for some expected cost specifications in general, feasible parameter values may be tiny. Such extremal parameter values induce CCPs with large differences between the smallest and largest coefficient in the encoding and are numerically challenging. The pMDP benchmarks seem more susceptible to such numerical issues.

Affine pMDPs cover most pMDP benchmarks, and the ETR encoding for them is bilinear. This observation allows using dedicated (numerical) solvers to find feasible valuations. Crucially, it allows us to adapt the CCP technique. Intuitively, we accelerate sampling by solving a convex approximation of the original constraint system around each sample. Using this adapted CCP approach not only outperforms methods that rely purely on the ETR encoding, but also the sampling-based PSO method.

10 Inductive Synthesis for Families

Pruning the parameter space by counterexamples

Summary | We present an inductive synthesis algorithm, designed to solve feasibility problems in fMCs and sketches of probabilistic programs. The method adapts the well-known CEGIS framework to the probabilistic setting. In the first part, we present the idea of CEGIS, applied to the feasibility problem for fMCs. In a nutshell, CEGIS enumerates all instantiations of the fMC and attempts to generalise the result to other instantiations. To generalise, we compute counterexamples on the instantiation and analyse the fMC structure. In the second part, we focus on an implementation on the level of sketches. We adjust the scheme from families to sketches and then discuss the technical aspects of this approach. In the last part, we provide an empirical evaluation.

Origins | This chapter closely resembles—including some extensions—the content of a technical report of [14][1], with some images from [13].

Background | This chapter considers feasibility for family MCs *and* sketches. Therefore, the complete Chapter 6 is relevant. It touches on some ideas of model checking as presented in Chapter 3.

10.1 CEGIS for Markov Chain Families

This section introduces *counterexample-guided inductive synthesis* (*CEGIS*) for families of MCs. (Syntax-Guided) CEGIS [STBS+06; ASFS18; JS17; ADKK+18] is a successful technique to synthesise programs from a description, and therefore natural to adapt on the program level. However, conceptually, CEGIS for probabilistic models is easier to understand on an explicit model of fMCs than on sketches. Thus, the aim of this first section is to provide intuition on our adaption of CEGIS. CEGIS

The idea is simple: Let $\mathfrak{D}$ be a fMC and φ some property. We tackle the feasibility problem by an enumerative search for an accepting assignment in $\mathcal{F}_{\mathfrak{D}}$. After considering some assignment ν, we *prune* the search space with unconsidered assignments: If ν rejects φ, we ask for a reason why the instantiation $\mathfrak{D}[\nu]$ rejects the property. We get

[1] The final version of that paper has been submitted after writing this chapter.

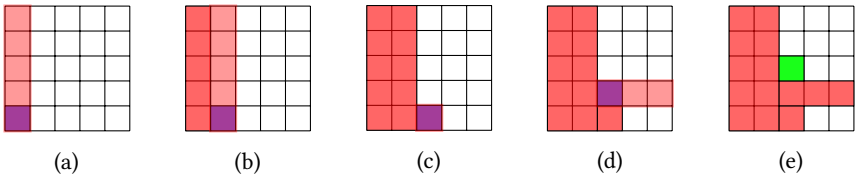


Figure 10.1: Schematic example run of CEGIS. The grid depicts a family with two parameters, each with 5 possible values. Thus, each cell corresponds to an assignment. Purple cells correspond to the currently considered assignment (rejecting), with light red indicating assignments pruned in this iteration. The green cell corresponds to an accepting assignment.

this reason in form of a counterexample. This counterexample may be thought of as a core of $\mathfrak{D}[\nu]$ that suffices to show that $\mathfrak{D}[\nu]$ does not accept the property. CEGIS then reasons that other instantiations having the same core also reject the φ.

Example 10.1 Figure 10.1 illustrates the reasoning used by CEGIS by drawing which instantiations are known to be rejecting. We consider some family with two parameters y_1 (horizontal axis) and y_2 (vertical axis), where each parameter has five possible values, and some property. We start considering the assignment represented in the left lower corner. The assignment is rejecting, and by the counterexample we obtain that all assignments that assign y_1 to the first value are rejecting. Thus, we prune the full column, i.e., we prune all five assignments that assign y_1 to its first value. Next, we consider a yet unpruned assignment, and again obtain that the value for parameter y_2 is irrelevant if y_1 is assigned to the second value. In the third iteration, we do not get any non-trivial counterexample. In the fourth iteration, we obtain that if y_2 is assigned to the fourth value, the value for y_1 is irrelevant, thereby pruning the fourth row. Finally, in the fifth iteration, we find a feasible instantiation and are done. Within 5 iterations, we have covered 15 instantiations.

synthesiser

We apply a typical separation of concerns as illustrated by Figure 10.2: a *synthesiser* (on the left) selects assignments ν that should be verified, and the verifier checks whether $\mathfrak{D}[\nu]$ accepts the specification. If it does accept the specification, we are done. Otherwise, the instantiation rejects the specification, and the verifier returns a *conflict* representing the holes that are relevant to form the core (the counterexample) of the MC. This conflict allows the synthesiser to prune some instantiations, meaning that the synthesiser knows that these instantiations are also rejecting.

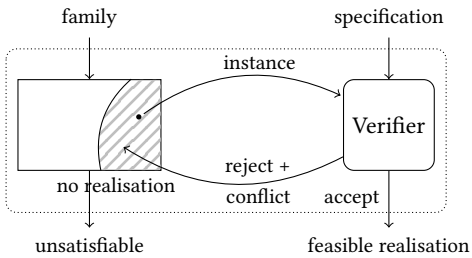


Figure 10.2: Schematic view on CEGIS for fMC feasibility.

A more concrete running example[1] is given below.

Example 10.2 Consider the fMC $\mathfrak{D} = (S, \iota, Y, \mathsf{Dom}_Y, \mathbf{P})$—in the special form of Remark 6.3—where $S \coloneqq \{s_0, \ldots, s_4\}$, $\iota \coloneqq s_0$, and $Y \coloneqq \{x, y\}$ with $\mathsf{Dom}_Y(x) \coloneqq \{s_2, s_3\}$ and $\mathsf{Dom}_Y(y) \coloneqq \{s_2, s_4\}$, and $\mathbf{P}$ given by:

$$\mathbf{P}(s_0) \coloneqq 1/2 \colon s_1 + 1/2 \colon x, \quad \mathbf{P}(s_1) \coloneqq 1/10 \colon s_0 + 4/5 \colon y + 1/10 \colon s_4, \quad \mathbf{P}(s_2) \coloneqq 1 \colon y,$$
$$\mathbf{P}(s_3) \coloneqq 1 \colon s_3, \qquad \mathbf{P}(s_4) \coloneqq 1 \colon s_4.$$

We depict $\langle \mathfrak{D} \rangle$ in Figure 10.3. We consider the specification

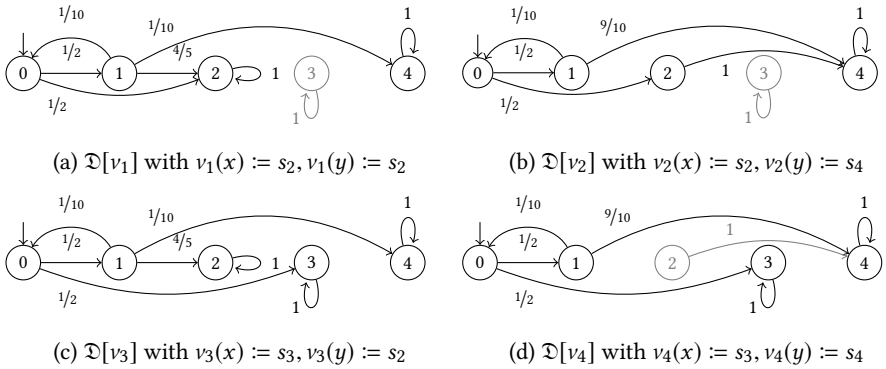
$$\Phi \coloneqq \{\varphi \coloneqq \mathbb{P}_{\leq 2/5}(\Diamond\{s_2\})\}.$$

The assignments ν_1, ν_2, ν_3 (see the captions in Figure 10.3 for their definition) reject the specification, ν_4 accepts it.
We may determine this complete partitioning by four model-checking invocations. Our goal is to use less. For that, we start by verifying $\mathfrak{D}[\nu_1]$. Assume the verifier (for now, magically) reports that the value for parameter y is irrelevant for rejecting the specification. We may then conclude that ν_2 also rejects the specification. Thereby, we save one model-checking call.

In this chapter, we focus on (optimal) feasibility, i.e., answering whether there exists an instantiation $D \in \langle \mathfrak{D} \rangle$ such that $D \models \Phi$. These ideas may be trivially extended to partitioning.

Related work | Related work includes the search-based approaches as outlined in Section 8.3 and enumerative approaches to families. However, the essential idea in this chapter is pruning the search space with guarantees. This idea has been hardly

[1] I apologise to the reader for introducing another family. The goals of universal running examples and concise examples seem to be contradictory.



(a) $\mathfrak{D}[v_1]$ with $v_1(x) := s_2, v_1(y) := s_2$

(b) $\mathfrak{D}[v_2]$ with $v_2(x) := s_2, v_2(y) := s_4$

(c) $\mathfrak{D}[v_3]$ with $v_3(x) := s_3, v_3(y) := s_2$

(d) $\mathfrak{D}[v_4]$ with $v_4(x) := s_3, v_4(y) := s_4$

Figure 10.3: The four different instantiations of family $\mathfrak{D}$. States s_i are denoted with i.

explored in the probabilistic domain. Recently, [DR18] proposed to exploit relationships between model and specification, reducing the number of model-checking instances.

10.1.1 Conflicts and the synthesiser

Before we focus on the conflict generation in the verifier, we introduce a naive synthesiser that may be used in combination with the verifier. The main goal is to formalise the type of conflicts that we expect the verifier to return. Intuitively, a conflict may be thought of as encoding a particular set of assignments. Therefore, we define the following: A *partial assignment* for an fMC $\mathfrak{D}$ is a function

partial assignment

$$\bar{v} \colon Y \to \mathsf{Dom}_Y(Y) \cup \{\bot\} \text{ such that } \bar{v}(y) \in \mathsf{Dom}_Y(y) \cup \{\bot\} \text{ for all } y \in Y.$$

For any partial assignments $\bar{v}_1$, $\bar{v}_2$, let

$$\bar{v}_1 \subseteq \bar{v}_2 \quad \text{iff} \quad \bar{v}_1(y) \in \{\bar{v}_2(y), \bot\} \text{ for all } y \in Y.$$

conflict

Definition 10.1 (Conflict) Let $\mathfrak{D}$ be an fMC and φ a property. Let $v \in \mathcal{F}_{\mathfrak{D}}$ be a rejecting assignment, that is $\mathfrak{D}[v] \not\models \varphi$. A partial assignment $\bar{v}_\varphi \subseteq v$ is a *conflict in v for φ* if

$$\mathfrak{D}[v'] \not\models \varphi \text{ for each } v' \supseteq \bar{v}_\varphi.$$

conflict set

A set of conflicts is called a *conflict set*.

·trivial

Every rejecting assignment $v \in \mathcal{F}_{\mathfrak{D}}$ is a *trivial conflict*.

To explore all instantiations, the synthesiser starts with $Q := \mathcal{F}_{\mathfrak{D}}$ and picks some assignment $v \in Q$[1]. Either $\mathfrak{D}[v] \models \Phi$ and we immediately return v, or the verifier

[1]We defer discussing details—like how to represent Q—to program-level CEGIS.

returns a conflict set $\{\nu_\varphi\}$: then Q is pruned by removing all $\nu \supseteq \bar{\nu}_\varphi$ for each $\bar{\nu}_\varphi$ in the conflict set. If Q is empty, we are done: each assignment rejects a property $\varphi \in \Phi$.

Example 10.3 Recall Example 10.2. After considering ν_1, the verifier magically returns the partial assignment $\bar{\nu} := \{x \mapsto s_2, y \mapsto \bot\}$ as a conflict for φ. Assignment $\nu_2 \supseteq \bar{\nu}$ only varies from ν_1 in the value of y, but $\bar{\nu}(y) = \bot$, i.e., y is not included in the conflict. We may conclude $\mathfrak{D}[\nu_2] \not\models \Phi$. The partial assignments $\{x \mapsto s_3, y \mapsto \bot\}$ and $\{x \mapsto \bot, y \mapsto s_3\}$ are not conflicts.

Thus, the synthesiser merely iterates over all assignments, while discarding any assignment that is covered by some conflict. The essential step thus is to find conflicts.

10.1.2 Counterexample-based verifier

The key task of a verifier is to check whether an instantiation accepts the given specification. Formally, we require the verifier to be sound and complete:

Definition 10.2 A verifier is sound and complete, if for fMC $\mathfrak{D}$, assignment ν, and specification Φ, the verifier terminates, the returned conflict set is empty iff ν is accepting Φ, and if it is not empty, it contains a conflict $\bar{\nu}_\varphi \subseteq \nu$ for some $\varphi \in \Phi$.

An example of a sound and complete verifier is a model-checking procedure that checks $\mathfrak{D}[\nu]$. If $\mathfrak{D}[\nu] \models \Phi$, then it returns an empty conflict set, and otherwise it returns a conflict set containing the trivial conflict ν. While this verifier is sound and complete, the synthesiser still has to iterate over all assignments.

State-level counterexamples | The verification procedure that we use to generate conflicts is built on top of the concept of state-level counterexamples for MCs [WJÁB+12].

Definition 10.3 (sub-fMC) Let $\mathfrak{D} = (S, \iota, Y, \mathsf{Dom}_Y, \mathbf{P}, \mathsf{rew})$ be an fMC with *critical states* $C \subseteq S$, $\iota \in C$. The *sub-fMC* of $\mathfrak{D}$ and C is the fMC critical states ·sub-fMC

$$\mathfrak{D}{\downarrow}C := (S' := C \cup \mathsf{succ}(C), \iota, \bigcup_{s \in C} \mathsf{occurs}(s), \mathbf{P}', \mathsf{rew}')$$

with $\mathbf{P}'$ defined by

$$\mathbf{P}'(s) := \begin{cases} \mathbf{P}(s) & \text{if } s \in C, \\ \{s \mapsto 1\} & \text{if } s \in \mathsf{succ}(C) \setminus C, \end{cases}$$

and rew'(s) = rew(s) for all $s \in C$ and rew(s) = 0 for all $s \in \mathsf{succ}(C) \setminus C$.

If the fMC is indeed an MC, we refer to this construction as a *subMC*. ·subMC

(a) Fragment of $\mathfrak{D}[\nu_1]$ (b) subMC of $\mathfrak{D}[\nu_1]$ with $C = \{0\}$

Figure 10.4: Fragment and corresponding subMC that suffices to refute $\mathbb{P}_{\leq 2/5}(\Diamond\{s_2\})$

Example 10.4 Figure 10.4b shows a subMC $\mathfrak{D}[\nu_1]{\downarrow}C$ with critical states $C = \{0\}$.

Recall that upper-bounded reachability properties are *safety* properties. Lower-bounded reachability properties are referred to as *liveness* properties. The essential attribute of subMCs is:

Proposition 10.1 ([WJÁB+12]) Let D be an MC and $\varphi := \mathbb{P}_{\leq\lambda}(\Diamond T)$ a safety property. For any set of critical states $C \subseteq S$ it holds:

$$D{\downarrow}C \not\models \varphi \quad \text{implies} \quad D \not\models \varphi.$$

counterexample

Definition 10.4 (Counterexample) Let D be an MC and $\varphi := \mathbb{P}_{\leq\lambda}(\Diamond T)$ a property. Any set of critical states $C \subseteq S$ with $D{\downarrow}C \not\models \varphi$ is a *counterexample* (CE) for D and φ.

Example 10.5 Recall Example 10.2. We have to check whether $\mathbb{P}_{\leq 2/5}(\Diamond\{s_2\})$ is satisfied by $\mathfrak{D}[\nu_1]$. In fact, the paths in the fragment of $\mathfrak{D}[\nu_1]$ in Figure 10.4a (ignoring the outgoing transitions of states 1 and 2) suffice to show that the probability to reach state 2 exceeds $2/5$. These paths are present in the subMC $\mathfrak{D}[\nu_1]{\downarrow}C$ with critical states $C = \{0\}$ as depicted in Figure 10.4b.
The fragment from Figure 10.4a is part of $\mathfrak{D}[\nu_2]$ too. Formally, the subMC of $\mathfrak{D}[\nu_2]{\downarrow}C$ is isomorphic to $\mathfrak{D}[\nu_1]{\downarrow}C$ and therefore rejects Φ. Thus, $\mathfrak{D}[\nu_2] \not\models \Phi$.

Next, we embed finding subMCs and generating conflicts from them into a verifier.

Verifier | Algorithm 7 outlines a basic verifier. First, we construct $D := \mathfrak{D}[\nu]$ once. We then use an off-the-shelf probabilistic model-checking procedure CHECK(D, φ) to determine all rejected $\varphi \in \Phi$. The algorithm then iterates over the violated φ and computes critical sets C of D that induce subMCs such that $D{\downarrow}C \not\models \varphi$ (line 9). These critical sets for safety properties may be obtained via standard methods [ÁBDJ+14; QJDW+15]. Adaptions for liveness properties (concretely for lower bounded reachability properties) are discussed at the end of the section. Finally, the procedure GENERATECONFLICT($\mathfrak{D}$, C) identifies the subset of parameters Y that occur in the subMCs $\mathfrak{D}{\downarrow}C$ and returns the corresponding conflict $\mathit{Conflict}(C, \nu) \subseteq \nu$ and stores it in the conflict set Conflict (line 9). The proposition below clarifies the relation between critical sets and conflicts.

Algorithm 7 Verifier (for safety properties)

```
1:  function VERIFY(fMC 𝔇, assignment ν, specification Φ)
2:      Violated ← ∅; Conflict ← ∅;
3:      D ←GENERATEMC(𝔇, ν)                                  ▹ construct 𝔇[ν] once
4:      for all φ ∈ Φ do
5:          if not CHECK(D, φ) then
6:              Violated ← Violated ∪ {φ}
7:      for all φ ∈ Violated do
8:          C_φ ← COMPUTECOUNTEREXAMPLE(D, φ)
9:          Conflict ← Conflict ∪ GENERATECONFLICT(𝔇, C_φ)
10:     return Conflict
```

Proposition 10.2 If C is a counterexample for $\mathfrak{D}[\nu]$ and φ, with $\mathfrak{D}[\nu] \not\models \varphi$, then C is also a counterexample for each $\mathfrak{D}[\nu']$ with $\nu' \supseteq \mathit{Conflict}(C, \nu)$.

Example 10.6 Recall from Example 10.5 that ν_2 rejects Φ, and that we may deduce this by inspecting the subMC from Figure 10.4b. In particular, we may deduce that ν_2 rejects Φ *without constructing* $\mathfrak{D}[\nu_2]$. Just considering ν_2, $\mathfrak{D}$ and C suffices: First, take all parameters occurring in $\mathfrak{D}{\downarrow}C$. This operation yields $\{x\}$ and may be implemented by inspection of $\mathfrak{D}$. The partial realisation $\bar{\nu} := \{x \mapsto s_2, y \mapsto \bot\}$ is a conflict. The assigned values for the other parameters do not affect the shape of the subMC induced by C.

Conflicts for liveness properties | We deem support for liveness properties essential. Consider a family describing possible controllers moving a robot and a specification consisting of only a safety property that something bad may only happen with a small probability. An accepting controller might simply never move and not make any progress. While not doing anything often induces safety, the synthesised controller is not useful.

To support liveness properties such as $\varphi := \mathbb{P}_{>\lambda}(\Diamond T)$, we first consider a (standard) dual safety property $\varphi' := \mathbb{P}_{<1-\lambda}(\Diamond B)$. The first idea is to fix an instantiation and set $B := S^T_{=0}$, similar to ideas in, e.g., Lemma 5.46. For any MC, the set of states B may be efficiently computed using graph algorithms. We have to be careful, however.

Example 10.7 We adapt Example 10.2. Consider $\mathfrak{D}[\nu_1]$ and let $\varphi := \mathbb{P}_{>3/5}(\Diamond\{s_4\})$. Assignment ν_1 rejects φ. Then, $\varphi' = \mathbb{P}_{<2/5}(\Diamond\{2\})$, which is refuted with critical set $C = \{0\}$ as before. Although $\mathfrak{D}[\nu_2] \downarrow C$ is again isomorphic to $\mathfrak{D}[\nu_1] \downarrow C$, assignment ν_2 accepts φ. The problem here is clear: The zero states depend on the

assignment. Concretely, state s_2 is a zero-state for $\mathfrak{D}[\nu_1]$ as $\nu_1(y) = s_2$, but not a zero-state for $\mathfrak{D}[\nu_2]$, as $\nu_2(y) = s_4$.

To prevent the flawed reasoning above, we ensure that the states in B are zero states in *all* instantiations. We enforce this by including the zero states for the current instantiation in the counterexample of φ:

Definition 10.5 Let D be an MC and $\varphi := \mathbb{P}_{\geq\lambda}(\Diamond T)$ a property. Let C be the counterexample for D and $\mathbb{P}_{\leq 1-\lambda}(\Diamond S^T_{=0})$. Then $C \cup S^T_{=0}$ is a counterexample for D and φ.

Together, this ensures that—in any instantiation with an isomorphic subMC—we reach states B with a critical probability mass[1] and never leave B. Indeed, this ensures that Proposition 10.2 holds also for liveness properties.

Example 10.8 We continue with the example above, in particular, $\varphi := \mathbb{P}_{>3/5}(\Diamond\{s_4\})$. In $\mathfrak{D}[\nu_1]$, we compute critical states $\{s_0, s_2\}$, preventing the erroneous reasoning from Example 10.2. For $\mathfrak{D}[\nu_4]$, we compute $C' := \{0\} \cup \{3\}$ as critical states, and as $\mathfrak{D}[\nu_4]{\downarrow}C'$ is isomorphic to $\mathfrak{D}[\nu_3]{\downarrow}C'$, we obtain that ν_3 also rejects this (liveness property) φ.

Conflicts for expected rewards | Performance criteria such as expected time to completion or expected energy consumption are widespread and typically expressed as upper bounds on expected rewards. The notion for counterexamples also exists for upper bounds on rewards [QJDW+15]. Formally, for MCs with finite expected rewards, the subMCs are changed such that all states in $\mathsf{succ}(C) \setminus C$ are made target states and get zero reward assigned. If the reward collected in the counterexample already rejects the upper bound, then clearly any extension would. The results as presented for safety properties carry over immediately. We omit a technical treatment of this construction. For lower bounds, there is no analogue to the construction above.

10.2 CEGIS for Probabilistic Programming Sketches

Below we adapt CEGIS for probabilistic models from state level to program level. In particular, we employ so-called *program-level counterexamples*, rather than counterexamples on the state level. This adaption is motivated by a number of observations. First, the sketch contains additional information that may be essential for finding feasible solutions quickly. Second, CEGIS has originally been formulated on program level, making the adaption to probabilistic program sketches natural. Third, methods for state-level CEs concentrate on generating small CEs, where small is measured in

[1] A good implementation takes a subset of $B' \subseteq B$ by considering $Pr_D(\Diamond B')$.

```
hole HX either { XA is 1 cost 3, 2}
hole HY either { YA is 1, 3 }
hole HZ either { 1, 2 }
constraint  !(XA && YA);
module rex
s : [0..3] init 0;
s = 0 -> 0.5: s'=HX + 0.5: s'=HY;
s = 1 -> 1: s'=s+HZ;
s >= 2 -> 1: s'=s;
endmodule
```

(a) Program sketch $\mathcal{S}$

```
module rex
s : [0..3] init 0;
s = 0 -> 0.5: s'=1 + 0.5: s'=3;
s = 1 -> 1: s'=s+2;
s >= 2 -> 1: s'=s;
endmodule
```

(b) Instantiation $\mathcal{S}(\{\mathrm{HX} \mapsto 1, \mathrm{HY} \mapsto 3, \mathrm{HZ} \mapsto 2\})$

Figure 10.5: Running example (repeated).

Algorithm 8 Synthesiser (feasibility synthesis)

1: **function** SYNTHESIS(program sketch $\mathcal{S}$, specification Φ, budget $\mathcal{B}$)
2: $\quad\psi \leftarrow$ INITIALISE$(\mathcal{S}, \mathcal{B})$
3: $\quad$**while** $\mathsf{sat}(\psi) \neq \emptyset$ **do**
4: $\quad\quad\mathbf{v} \leftarrow$ GETREALISATION$(a_{\mathbf{v}} \in \mathsf{sat}(\psi))$
5: $\quad\quad\mathsf{Conf} \leftarrow$ VERIFY$(\mathcal{S}[\mathbf{v}], \Phi)$
6: $\quad\quad$**if** $\mathsf{Conf} = \emptyset$ **then return** $\mathbf{v}$
7: $\quad\quad\psi \leftarrow \psi \wedge \left(\bigwedge_{\bar{\mathbf{v}} \in \mathsf{Conf}} \text{LEARNFROMCONFLICT}(\mathcal{S}, \bar{\mathbf{v}}) \right)$
8: $\quad$**return** `Unsat`

the size of the CE. For generating conflicts, we would like a more flexible notion of small CEs, as we are interested in finding small conflicts. The notion of size in program-level CEs is more flexible and thus easier to adapt. Starting from program-level CEs is therefore more natural. This section presents a synthesiser working on the level of sketches and an adapted verifier that uses these program-level counterexamples instead of state-level counterexamples. The section finishes with some technical details about the verifier and a brief outline of the treatment of ill-formed sketches by the synthesiser. Our method support sketches with holes in guards. We reuse the running example from Figure 6.7, repeated in Figure 10.5.

10.2.1 Program-level synthesiser

We present a synthesiser for feasibility and an adaption for optimal feasibility.

10.2.1.1 Feasibility

Overview | The synthesiser is outlined in Algorithm 8. We give a brief overview before detailing the steps. During the synthesis process, the synthesiser stores and

queries the set of realisations not yet pruned. These remaining realisations are represented by (the satisfying assignments of) the formula ψ over hole-assignments. Iteratively adjoining conjunctions to ψ thus prunes the set of realisations. $\textsc{Initialise}(\mathcal{S}, \mathcal{B})$ constructs ψ such that the formula represents *all* realisations that satisfy the constraints in the sketch $\mathcal{S}$ within the budget $\mathcal{B}$. We exploit an SMT-solver for linear bounded integer arithmetic to obtain a realisation $\mathbf{v}$ consistent with ψ, or to return `Unsat` if no such realisation exists. As long as new realisations are found, the verifier analyses them and returns a conflict set Conf. If $\mathsf{Conf} = \emptyset$, then $\mathbf{v}$ is accepting, and the search is terminated. Otherwise, the synthesiser updates ψ based on the conflicts. The realisation $\mathbf{v}$ is always pruned, i.e., after updating, $\mathbf{v}$ is no compliant with ψ.

Initialisation | $\textsc{Initialise}(\mathcal{S}, \mathcal{B})$: Let hole $h \in \mathcal{H}$ have (ordered) options $\mathbf{O}_h = \{o_h^1, \ldots, o_h^n\}$. To encode realisation $\mathbf{v}$, we introduce integer-valued meta-variables $K_H := \{\kappa_h \mid h \in \mathcal{H}\}$ with the semantics that $\kappa_h = i$ whenever hole h has value o_h^i, i.e., $\mathbf{v}(h) = o_h^i$. We set

$$\psi := \psi_{\text{opti}} \wedge \psi_{\Gamma} \wedge \psi_{\text{cost}},$$

where ψ_{opti} ensures that each hole is assigned to some option, ψ_{Γ} ensures that the sketch's constraints Γ are satisfied, and ψ_{cost} ensures that the budget is respected. These sub-formulae[1] are:

$$\psi_{\text{opti}} := \bigwedge_{h \in \mathcal{H}} 1 \leqslant \kappa_h \leqslant |\mathbf{O}_h|, \qquad \psi_{\Gamma} := \bigwedge_{\gamma \in \Gamma} \gamma[N_h^i / \kappa_h \doteq i],$$

$$\psi_{\text{cost}} := \sum_{h \in \mathcal{H}} \omega_h \leqslant \mathcal{B} \wedge \left(\bigwedge_{h \in \mathcal{H}} \bigwedge_{i=1}^{|\mathbf{O}_h|} \kappa_h \doteq i \rightarrow \omega_h \doteq \mathsf{cost}(o_h^i) \right),$$

where

- $\gamma[N_h^i / \kappa_h \doteq i]$ denotes that in every constraint $\gamma \in \Gamma$ we replace each option name N_h^i for an option o_h^i with $\kappa_h \doteq i$,
- ω_h are fresh variables storing the cost for the selected option at hole h used to limit the total cost of the options by the budget $\mathcal{B}$.

Let us clarify the encoding with an example.

[1] We additionally have to ensure that the integer variables are assigned integers.

Example 10.9 For the sketch in Figure 10.5a, we obtain (slightly simplified):

$$\begin{aligned}\psi \coloneqq\ & 1 \leq \kappa_{\mathrm{HX}} \leqslant 2 \wedge 1 \leqslant \kappa_{\mathrm{HY}} \leqslant 2 \wedge 1 \leqslant \kappa_{\mathrm{HZ}} \leqslant 2 \\ & \wedge\ \neg(\kappa_{\mathrm{HX}} \doteq 1 \wedge \kappa_Y \doteq 1) \wedge \omega_{\mathrm{HX}} + \omega_{\mathrm{HY}} + \omega_{\mathrm{HZ}} \leqslant \mathcal{B} \\ & \wedge\ \big(\kappa_{\mathrm{HX}} \doteq 1 \rightarrow \omega_{\mathrm{HX}} \doteq 3\big) \wedge \big(\kappa_{\mathrm{HX}} \doteq 2 \rightarrow \omega_{\mathrm{HX}} \doteq 0\big) \wedge \omega_{\mathrm{HY}} \doteq 0 \doteq \omega_{\mathrm{HZ}}.\end{aligned}$$

The formula ψ encodes that there are two options for each hole and HX and HY are not allowed to be both 1 by the sketch constraint. Finally, the cost for the first option for HX is 3, all other cost are zero.

Finding a realisation | GetRealisation(ψ): To obtain a realisation $\mathbf{v}$, we check satisfiability of ψ. The solver either returns `Unsat` indicating that the synthesiser is finished, or `Sat`, together with a satisfying assignment $a_{\mathbf{v}} \colon K_H \rightarrow \mathbb{N}$. The assignment $a_{\mathbf{v}}$ uniquely identifies a realisation $\mathbf{v}$ by $\mathbf{v}(h) \coloneqq o_h^{a_{\mathbf{v}}(\kappa_h)}$. The sum over all ω_H gives $\mathbf{c}(\mathbf{v})$.

Example 10.10 Consider ψ from Example 10.9. The satisfying assignment (for $\mathcal{B} \geq 3$) $a_{\mathbf{v}} \coloneqq \{\kappa_{\mathrm{HX}} \mapsto 1, \kappa_{\mathrm{HY}}, \kappa_{\mathrm{HZ}} \mapsto 2, \omega_{\mathrm{HX}} \mapsto 3, \omega_{\mathrm{HY}}, \omega_{\mathrm{HZ}} \mapsto 0\}$ represents $\mathbf{v}$—with $c(\mathbf{v}) = 3$—from Figure 10.5b.

Verifying the realisation | Verify($\mathcal{S}[\mathbf{v}], \Phi$): invokes any sound and complete verifier, e.g., the verifier from Section 10.1.2, or an adaption as presented in Section 10.2.2. Either $\mathbf{v}$ is accepting, or it returns a set of conflicts[1].

Example 10.11 Continuing with $\mathbf{v}$ from Example 10.10, Consider $\Phi = \{\mathbb{P}_{\leq 2/5}[\Diamond\ \mathtt{s{=}3}]\}$. The verifier (for now, magically) constructs a conflict set $\{\bar{\mathbf{v}}\}$ with $\bar{\mathbf{v}} = \{\mathrm{HY} \mapsto 3\}$.

Learning from the conflict | LearnFromConflict($\mathcal{S}, \bar{\mathbf{v}}$): For a conflict $\bar{\mathbf{v}} \in \mathsf{Conf}$, we adjoin ψ with the formula

$$\neg\Big(\bigwedge_{h \in \mathcal{H}, \bar{\mathbf{v}}(h) \neq \bot} \kappa_h \doteq a_{\mathbf{v}}(\kappa_h)\Big).$$

[1] As in Section 10.1.1: A *partial realisation* for $\mathcal{S}$ is a function $\bar{\mathbf{v}} \colon \mathcal{H} \rightarrow \mathbf{O}_{\mathcal{H}} \cup \{\bot\}$ such that $\forall h \in \mathcal{H}.\ \bar{\mathbf{v}}(h) \in \mathbf{O}_h \cup \{\bot\}$. For partial realisations $\bar{\mathbf{v}}_1, \bar{\mathbf{v}}_2$, let $\bar{\mathbf{v}}_1 \subseteq \bar{\mathbf{v}}_2$ iff $\bar{\mathbf{v}}_1(h) \in \{\bar{\mathbf{v}}_2(h), \bot\}$ for all $h \in \mathcal{H}$. Let $\mathbf{v}$ be a realisation such that $\mathcal{S}[\mathbf{v}] \not\models \varphi$ for $\varphi \in \Phi$. Partial realisation $\bar{\mathbf{v}}_\varphi \subseteq \mathbf{v}$ is a *conflict* for φ iff $\forall \mathbf{v}' \supseteq \bar{\mathbf{v}}_\varphi\ \mathcal{S}[\mathbf{v}'] \not\models \varphi$.

Algorithm 9 Synthesiser (max synthesis)

1: **function** SYNTHESIS($\mathcal{S}$, Φ, $\mathcal{B}$, target predicate $\mathbf{T}$, tolerance ε)
2: $\lambda^* \leftarrow \infty$, $\mathbf{v}^* \leftarrow$ Unsat
3: $\psi \leftarrow$ INITIALISE($\mathcal{S}, \mathcal{B}$)
4: **while** $\mathsf{sat}(\psi) \neq \emptyset$ **do**
5: $\mathbf{v} \leftarrow$ GETREALISATION($a_{\mathbf{v}} \in \mathsf{sat}(\psi)$)
6: $\mathsf{Conf}, \lambda_{\text{new}} \leftarrow$ OPTIMISEVERIFY($\mathcal{S}[\mathbf{v}], \Phi, \mathbf{T}, \lambda^*, \varepsilon$)
7: **if** $\mathsf{Conf} = \emptyset$ **then** $\lambda^* \leftarrow \lambda_{\text{new}}, \mathbf{v}^* \leftarrow \mathbf{v}$
8: $\psi \leftarrow \psi \wedge \left(\bigwedge_{\bar{\mathbf{v}} \in \mathsf{Conf}} \text{LEARNFROMCONFLICT}(\mathcal{S}, \bar{\mathbf{v}}) \right)$
9: **return** $\mathbf{v}^*$

The formula excludes realisations $\mathbf{v}' \supseteq \bar{\mathbf{v}}$. Intuitively, the formula states that the potentially accepting realisations (encoded by the updated ψ) must assign a different option to at least one of the holes h that are assigned by $\bar{\mathbf{v}}$.

Example 10.12 We continue with $\bar{\mathbf{v}} = \{\text{HY} \mapsto 3\}$ from Example 10.11. The synthesiser updates $\psi \leftarrow \psi \wedge \kappa_{\text{HY}} \neq 2$ (recall that $\kappa_{\text{HY}} = 2$ encodes $\text{HY} \mapsto 3$). A satisfying assignment $\{\kappa_{\text{HX}}, \kappa_{\text{HY}}, \kappa_{\text{HZ}} \mapsto 1\}$ for ψ encodes $\mathbf{v}'$ from Example 6.9. As $\mathfrak{D}[\mathbf{v}'] \models \Phi$, the verifier reports no conflict.

10.2.1.2 Optimal feasibility

We adapt the synthesiser to support optimal synthesis, cf. Algorithm 9. The problem asks to maximise the probability of reaching states T described by a predicate $\mathbf{T}$, w.r.t. the tolerance $\eta \in (0, 1)$. Algorithm 9 stores in λ^* the maximal probability $Pr_{\mathcal{S}[\mathbf{v}]}(\Diamond T)$ over all $\mathbf{v}$, and this maximising $\mathbf{v}$ as $\mathbf{v}^*$. In each iteration, an optimising verifier is invoked (line 6) on realisation $\mathbf{v}$. An optimising verifier wraps standard model-checking procedures to ensure the following behaviour: If $\mathcal{S}[\mathbf{v}] \models \Phi$ and $Pr_{\mathcal{S}[\mathbf{v}]}(\Diamond T) > \lambda^*$, it returns an empty conflict set *and* $\lambda_{\text{new}} := Pr_{\mathcal{S}[\mathbf{v}]}(\Diamond T)$. Otherwise, it reports a conflict set for $\Phi \cup \{\mathbb{P}_{\geq (1-\eta) \cdot \lambda^*}(\Diamond T)\}$.

Example 10.13 Consider some sketch with a target predicate. We select a first realisation, which does not satisfy the specification Φ. We apply pruning as in feasibility analysis. In the second iteration, we pick a realisation which accepts Φ. Furthermore, we obtain by model checking that the probability to reach the target is $1/3$. In the third iteration, we pick a realisation which accepts Φ, but the probability to the target is only $1/4$. We use $\mathbb{P}_{\geq 1/3}(\Diamond T)$ as a property for counterexample generation (we already know that it is rejected by the current

Algorithm 10 Verifier

```
1:  function VERIFY(Sketch 𝒮, realisation v, specification Φ)
2:      Violated ← ∅; Conflict ← ∅;
3:      D ←GENERATEMC(𝒮, v)                                  ▷ construct ⟦ 𝒮[v] ⟧ once
4:      for all φ ∈ Φ do
5:          if not CHECK(D, φ) then
6:              Violated ← Violated ∪ {φ}
7:      for all φ ∈ Violated do
8:          C_φ ← COMPUTECOUNTEREXAMPLE(𝒮[v], D, φ)
9:          Conflict ← Conflict ∪ GENERATECONFLICT(𝒮, C_φ)
10:     return Conflict
```

realisation). We may then prune additional realisations that do not improve our optimisation criterium. In the fourth iteration, we then pick a realisation which satisfies the specification, *and* improves the reachability probability to $^3/_4$. We continue with these iterations, until we have used all realisations to either improve our current optimum, or to prune (potentially other) realisations.

10.2.2 Program-level verifier

We now adapt the state-level verifier from Section 10.1.2 in Algorithm 7 to use program-level counterexamples [WJVÁ+15] for generating conflicts. We show the adapted algorithm in Algorithm 10. The important change is that we now work on the program-level and thus use a different counterexample generation. All other changes are just to match the syntax of this section.

Constructing the Markov chain | GENERATEMC($\mathcal{S}$, **v**): This procedure first constructs the instantiation $\mathcal{S}[\mathbf{v}]$, i.e., a program without holes, from $\mathcal{S}$ and **v**, as in Figure 10.5b on page 267: We replace each hole declaration by a declaration of integers.

`hole h either ...` with `const int h;`

Constraints in the sketch are removed, as they are handled by the synthesiser. Thus, the passed program is a standard PRISM program with open constants. The realisation is passed separately, the sketch is parsed *once* and then appropriately instantiated. The instantiation is then translated into the underlying MC $[\![\mathcal{S}[\mathbf{v}]]\!]$ via standard procedures, with transitions annotated with their generating commands.

```
module rex
s : [0..3] init 0;
s = 0 -> 0.5: s'=1 + 0.5: s'=3;
s = 1 -> 1: s'=s+2;
s >= 2 -> 1: s'=s;
endmodule
```

(a) Program (repeated)

```
const int X = 1, Y = 3;
...
module rex
s : [0..3] init 0;
s=0 -> 0.5: s'=X + 0.5: s'=Y;
endmodule
```

(b) CE for upper bound

```
module rex
s : [0..3] init 0;
s=0 -> 0.5:s'=X + 0.5:s'=Y;
s=3 -> s'=3
endmodule
```

(c) CE for lower bound

Figure 10.6: Program (a) with CEs for (b) $\mathbb{P}_{\leq 2/5}(\Diamond \mathsf{s{=}3})$ and (c) $\mathbb{P}_{>3/5}(\Diamond \mathsf{s{=}2})$

Computing program-level CEs | $\textsc{ComputeCounterExample}(\mathcal{S}[\mathbf{v}], D, \varphi)$ computes program-level CE as analogues of critical sets. These CEs are defined on commands rather than on states. Let $\mathcal{L} = (\mathsf{Var}, \mathsf{Cmd})$ be a program and $\mathsf{Cmd}' \subseteq \mathsf{Cmd}$. Let $\mathcal{L}_{|\mathsf{Cmd}'} := (\mathsf{Var}, \mathsf{Cmd}')$ denote the restriction of $\mathcal{L}$ to Cmd', with variables (and initial states) as in $\mathcal{L}$. Building $\mathcal{L}_{|\mathsf{Cmd}'}$ preserves non-overlapping guards, but may introduce deadlocks in $[\![\, \mathcal{L}_{|\mathsf{Cmd}'} \,]\!]$ (just like a critical set introduces deadlocks). To remedy this, we use the standard operation fixdl introduced in Section 3.5.

program-level counterexample

Definition 10.6 For program $\mathcal{L} = (\mathsf{Var}, \mathsf{Cmd})$ and specification Φ with $\mathcal{L} \not\models \Phi$, a *program-level counterexample* $\mathsf{Cmd}' \subseteq \mathsf{Cmd}$ is a set of commands, such that for all (non-overlapping) programs $\mathcal{L}' := (\mathsf{Var}, \mathsf{Cmd}'')$ with $\mathsf{Cmd}'' \supseteq \mathsf{Cmd}'$ it holds that $\mathsf{fixdl}(\mathcal{L}') \not\models \Phi$.

Example 10.14 We consider the setting from Example 10.11 in more detail. Recall that we consider $\mathcal{S}[\mathbf{v}]$ as in Figure 10.5b (repeated in Figure 10.6a) with $\Phi = \{\mathbb{P}_{\leq 2/5}[\Diamond\ \mathsf{s{=}3}]\}$. Figure 10.6b shows a CE. The probability to reach s=3 in the underlying MC is $1/2 > 2/5$.

We discuss how to compute program-level CEs in Section 10.2.3.

Remark 10.1 We choose *not* to use the counterexample notion of [WJVÁ+15] for two reasons. First, that notion is not straightforward to adapt to properties beyond upper-bounded reachability, and second, our notion makes the soundness of the verifier trivial. Section 10.2.3 contains a comparison of the two definitions.

From counterexample to conflict | $\textsc{GenerateConflict}(\mathcal{S}, \mathbf{v}, \mathsf{Cmd})$ generates conflicts from counterexamples, i.e., from the set of commands Cmd. We map these commands from $\mathcal{S}[\mathbf{v}]_{|\mathsf{Cmd}}$ to commands Cmd' in $\mathcal{S}$ such that $\{\mathsf{cmd}[\mathbf{v}] \mid \mathsf{cmd} \in \mathsf{Cmd}'\} = \mathsf{Cmd}$. That is, we restore which holes appear in the part of the sketch leading to the CE Cmd for $\mathcal{S}[\mathbf{v}]$. The conflict $\mathit{Conflict}(\mathsf{Cmd}, \mathbf{v})(h) := \mathbf{v}(h)$ contains all $h \in \mathcal{H}$ appearing in Cmd'.

Proposition 10.3 If Cmd is a CE for $\mathcal{S}[\mathbf{v}]$, then for any $\mathbf{v}' \supseteq \mathit{Conflict}(\mathsf{Cmd}, \mathbf{v})$, Cmd is also a CE for $\mathcal{S}[\mathbf{v}']$.

Example 10.15 The CEs in Figure 10.6b contain commands which depend on the realisations for holes X and Y. For these fixed values, the program rejects the specification *independent of the value for Z*, so Z is not in the conflict $\{X \mapsto 1, Y \mapsto 3\}$.

10.2.3 Program-level CEs for CEGIS

We report how to compute program-level CEs in the context of CEGIS. Therefore, we first recollect a MaxSat based approach to computing counterexamples, show how we extend and improve upon that algorithm, and how to integrate it into the conflict computation.

10.2.3.1 Reusing a MaxSat approach

We aim to reuse parts of an MaxSat [BHMW09] approach from [DJWÁ+14] used to compute high-level CEs proposed in [WJVÁ+15].

Program-level CEs versus traditional high-level CEs | Before we may reuse algorithm and results from [DJWÁ+14], we compare our counterexamples with the definition for counterexamples which are the basis of the existing algorithms.

Definition 10.7 (High-level CEs) ([DJWÁ+14]) Given a program $\mathcal{L} = (\mathsf{Var}, \mathsf{Cmd})$ and property $\varphi = \mathbb{P}_{\leq\lambda}(\Diamond T)$ s.t. $\mathcal{L} \not\models \varphi$, the set $\mathsf{Cmd}' \subseteq \mathsf{Cmd}$ is a *high-level counterexample* if $\mathsf{fixdl}(\mathcal{L}_{|\mathsf{Cmd}'}) \not\models \varphi$.

high-level counterexample

We have the following connection between high-level and program-level CEs.

Proposition 10.4 For safety properties, high-level and program-level CEs coincide.

Each program-level CE is trivially a high-level CE. A high-level CE is a program-level CE, as any additional command may only be enabled in unreachable or deadlock states. Otherwise, the program would contain overlapping commands, violating Definition 10.6. Furthermore, unreachable states are irrelevant to any property and, intuitively speaking, adding transitions to deadlocks cannot decrease the probability to reach ψ.

Assuming non-overlapping programs[1] is crucial, as illustrated in the example below.

[1] PRISM semantics for MCs is that overlapping guards are resolved by a uniform distribution instead of non-determinism (which is the standard semantics for MDPs). Assuming non-overlapping programs is thus the same as assuming the MDP semantics for PRISM programs.

Example 10.16 Consider adding the command `s=0 -> s'=2` to the program in Figure 10.6b. The program now has overlapping guards (in state `s=0`). Applying the rescaling semantics, the probability to reach `s=3` in this program, i.e., in the underlying MC, is *reduced* to 1/4, the extended program no longer rejects Φ.

Computing Counterexamples | The technique in [DJWÅ+14] computes *minimal* high-level CEs (i.e. the smallest set of commands) violating a given reachability property with an upper bound λ. The set is not unique in general. It reduces the computation to repeatedly solving `MaxSat` instances over the conjunction of two propositional formulae, Ξ and Υ. The formula Ξ encodes a selection of commands and (via a negation) the `MaxSat` solver minimises the size of the command set. The formula Υ encodes necessary constraints on valid CEs, e.g, it states that a command enabled in the initial state must be selected. In this way, the `MaxSat` solver returns a candidate set Cmd of commands. Using model-checking procedures it is then checked whether the candidate Cmd is sufficient to exceed the probability bound λ, i.e., whether it is a CE. If so, Cmd by construction is a minimal CE. Otherwise, Υ is strengthened to exclude Cmd and (possibly) other candidate sets. Details about this approach are given in [Hen18].

10.2.3.2 More properties

We provide support for CEs beyond upper bounds on reachability probabilities.

Liveness properties | Proposition 10.4 is crucial for the correctness of our approach, but does not hold for liveness properties. Suppose Cmd is a (high-level) CE for $\mathcal{L}$ and φ. Adding commands adds/extends paths and thereby may increase the probability mass reaching the target, as discussed in Example 10.7. Thus, adding commands may yield a program $\mathcal{L}'$ such that $\mathcal{L}' \models \varphi$. However, a high-level CE is a program-level CE if there are no deadlocks reachable in $[\![\mathcal{L}_{|\mathsf{Cmd}}]\!]$.

The idea, analogue to Section 10.1.2, is to trap more probability mass than $1-\lambda$ in the bad states $B \coloneqq S^T_{=0}$ (from which there is no path to the target states T). If there is less than λ probability mass reaching T, then more than $1-\lambda$ probability mass reaches the bad states. To this end, we first compute a high-level CE Cmd for reaching states in B with probability at most $1-\lambda$. Then, we extend Cmd with commands that ensure that the states in B actually may never reach T. We do this by adding commands that fix the behaviour of states in B. Thereby, the program cannot be extended to a program where there are transitions from B to T. The resulting set of commands is a program-level CE as in Definition 10.6. These CEs are no longer necessarily the smallest (in terms of the number of commands) for any given $\mathcal{L}$ and φ.

Example 10.17 Consider the property $\varphi = \mathbb{P}_{>3/5}(\Diamond \texttt{s=2})$. The property is violated by the program from Figure 10.6a. Figure 10.6c constitutes a CE since with probability $1/2$ the system moves to `s=3` that in turn is equipped with a self-loop. Hence, the probability to ever reach `s=2` is guaranteed to be at most $1/2$. Dropping the command with guard `s=3` is insufficient for this guarantee: an additional command could add transitions from `s=3` and `s=1` to `s=2`, thereby increasing the probability mass and exceeding the (lower) bound of φ.

Expected rewards | We adapt ideas of state-level CEs for rewards from [QJDW+15] to the program level. Intuitively, in the underlying MC of the program, we replace the self-loops from the fixdl operation by transitions to a target state. For its soundness, the method has to assume that for each realisation, the probability to reach the target is one. Otherwise, the expected reward is by definition infinite,and the instantiation may be rejected anyways. We simply extend the specification to explicitly check that the expected reward is finite, and we may use counterexamples also for this property. Support for lower bounds on rewards requires additional assumptions and has not been considered.

10.2.3.3 Better integration

So far, we considered the generation of counterexamples independent of their usage within CEGIS. We have made various changes to improve the integration. We report on some of them. The following adaptions allow us to derive conflicts that are in many situations smaller than the conflicts derived from the minimal CEs described above. Smaller conflicts lead to more effective pruning and thus improve the performance of the framework. The conflicts we obtain are always at least as small as before.

Relevant commands | The `MaxSat` approach in [DJWÁ+14] minimises over *all* commands. However, the size of the conflict only depends on the holes contained in the commands. Therefore, we change Ξ to minimise the number of relevant commands. A command is *relevant*, if it contains a hole. Any non-relevant command may be added to a CE without negatively affecting the sizes of the generated conflicts. This optimisation significantly reduces the number of candidates considered during the CE generation. Formula Υ still reasons about all commands, which allows to restrict the candidate command sets further. ·relevant

Example 10.18 In Figure 6.7a on page 174 only the commands with guards `s=0` and `s=1` are relevant. Instead of considering up to 2^4 candidate command sets (all four commands), we only consider up to 2^2 CEs (the two relevant commands).

In the future, we would like to actually change Ξ to minimise the holes that are contained in the commands. Right now, however, the counterexample generation is unaware of the concept of holes. Likewise, currently counterexamples consider commands rather than of branches of a command. The main reason is performance; in the current implementation, commands are assigned to states. Branches, however, would have to be assigned to (higher number of) transitions. However, a branch-based implementation might yield considerably smaller counterexamples.

Multiple CEs | CEs (even with minimal size) are not unique. For the same realisation, multiple CEs may lead to different conflicts. Each of them may be useful to the synthesiser to prune future realisations. Furthermore, our approach does not require to use only the smallest CE, we may want to explore other CEs beyond the first one. However, minimality is crucial: If a strict subset of a CE Cmd is already a CE, conflicts generated by Cmd do not prune additional realisations. We thus extend the `MaxSat` loop. If we find a CE Cmd, we extend Υ by blocking all CEs which include the set of relevant commands in Cmd. We may then continue searching for further minimal CEs. Nevertheless, some of the conflicts generated may not prune any further realisations in the synthesiser, as these realisations might have been considered before. For future work, it may be interesting to use additional knowledge from the synthesiser, thereby focussing on CEs that definitively prune unexplored parts of the family.

CEs for specifications with multiple properties | The description above searches CEs for each property independently. Therefore, the CEs that are computed w.r.t one property might all be subsumed by CEs computed for another property. It is thus beneficial to consider all properties at once, thereby reducing the candidate CEs. We integrated support for CE generation for multiple properties with the same target states. For other combinations, a naive implementation induces a severe performance hit to the `MaxSat` solver. Proper support is left as future work.

10.2.4 Support for flawed programs

We have argued in Section 7.3 that it is beneficial to support sketches that are not well-defined, i.e., that contain invalid programs. Below, we give an overview how we support sketches with invalid programs. Observe that we assume that such programs do not satisfy the specification and should be considered as rejecting.

Out-of-bounds | Variable bounds in programs should be strictly respected. However, constraining the sketch to prevent the generation of realisations that violate these bounds might be complex. Therefore, we extended the semantics such that the underlying MC contains a state reflecting a variable being out-of-bounds (by checking the variable values during the MC construction). Then, the specification is extended with a property that no probability mass should reach these states. This extension means that accepting assignments have no variables that go out of bounds. We may also generate counterexamples for this property.

Overlapping guards | As already pointed out, overlapping guards should lead to nondeterminism—and thus the underlying models are not MC. The assumption of uniform resolution of this nondeterminism (as optional in PRISM) may prevent effective CE generalisation. Therefore, we reject realisations with overlapping guards. In order to mitigate the performance degradation caused by returning trivial conflicts, we implement a similar extension as for out-of-bounds, in which we label states that satisfy guards of multiple commands accordingly, and extend the specification to prevent reaching these states.

Deadlocks | The existence of deadlocks prevents verification of time unbounded properties, therefore most verification tools automatically apply the fixdl operation. While this operation is sensible for a single program, it is not for sketches with holes in guards, as deadlocks are problematic for CE generalisation in combination with lower-bounded properties. Thus, we treat them analogously to overlapping guards by labelling states accordingly.

10.3 Empirical Evaluation

10.3.1 Set-up

Implementation | We evaluate the synthesis framework with a prototype (cf. Chapter 13) using the SMT-solver Z3 [MB08] and an extended model checker STORM [9].

Environment | All results are obtained on a Macbook MF839LL/A, within three hours and using less than 8 GB RAM.

Research questions | We compare the performance of a CEGIS loop with an enumerative approach as baseline. That baseline linearly depends on the number of realisations, and on the underlying MCs' size. We focus on sketches where all realisations are explored, as relevant for optimal synthesis. For concise presentation we use Unsat

variants of feasibility synthesis. Enumerative methods perform mostly independent of the order of enumerating realisations.

Case studies

We consider the following three case studies:

Dynamic power management (DPM) | We consider the DPM problem as discussed in Section 7.3.1.1[1]. The goal of this problem is to trade-off power consumption of complex electronic systems for performance in a controlled fashion. We sketch a controller that decides based on the current workload expressed by the occupancy of low-priority and high-priority request buffers, inspired by [GTC15]. The fixed environment contains no holes. The goal is to synthesise the guards and updates to satisfy a specification with properties such as φ_1: the expected number of lost requests is below λ, and φ_2: the expected energy consumption is below κ.

Network intrusion | This model describes a network (adapted from [KNPV09]), in which the controller tries to infect a target node via intermediate nodes. The node has to be attacked by intruding intermediate nodes. If an attack fails, the node is temporarily on guard and more difficult to intrude. We sketched a partial strategy aiming to minimise the expected time to intrusion based on intruded nodes and the last action. Constraints encode domain specific knowledge, e.g., we never target nodes whose neighbours are already infected.

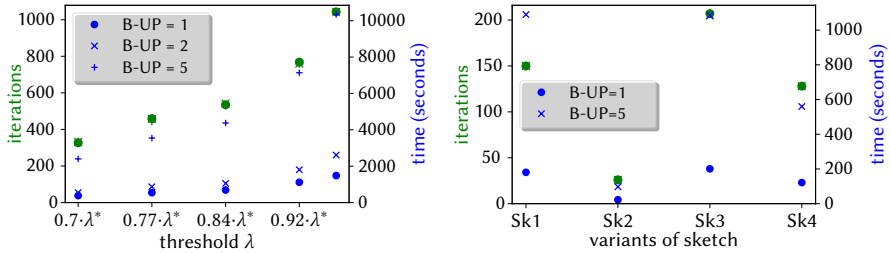
Grid | This model is based on a classical benchmark for solving partially observable MDPs (POMDPs). We exploit the relationship as described in Section 7.1 and focus on finding a deterministic m-state strategy s.t. in expectation, the strategy requires less than λ steps to the target. We create a sketch describing all induced MCs under m-state strategies. Actions are reflected by holes, while same holes ensure that the strategy is observation-based.

10.3.2 Evaluation

We evaluate results for *DPM* and summarise further results.

DPM | *DPM* has 9 holes with 260K realisations, and realisations have 5K (reachable) states on average, ranging from 2K to 8K states. We use this sketch to illustrate some claims. *The performance of CEGIS significantly depends on the specification, namely,*

[1] the actual model used is slightly more complex than what was discussed—the tool includes the raw data for the experiment.



(a) Performance for varying λ

(b) Performance for varying $\mathcal{S}$

Figure 10.7: Performance (runtime and iterations) on DPM

on the thresholds appearing in the properties. Figure 10.7a shows how the number of iterations (left axis, green circle) and the runtime in seconds (right axis, blue) change for varying λ for property φ_1 (stars and crosses are explained later). We obtain a speedup of 100× over the baseline for $\lambda = 0.7{\cdot}\lambda^*$, dropping to 23× for $\lambda = 0.95{\cdot}\lambda^*$, where λ^* is the minimal probabilty over all realisations. The strong dependency between performance and "unsatisfiability" is not surprising. The more unsatisfiable, the smaller the conflicts (as in [DJWÁ+14]). Small conflicts have a double beneficial effect. First, the prototype uses an optimistic verifier searching for minimal conflicts; small conflicts are found faster than large ones. Second, small conflicts prune more realisations. A slightly higher number of small conflicts yields a severe decrease in the number of iterations. Thus *the further the threshold from the optimum, the better the performance.*

Reconsider Figure 10.7a. Crosses and stars correspond to a variant of the sketch in which we have blown up the state space of the underlying MCs by a factor B-UP. Observe that performance degrades similarly for the baseline and our algorithm, which means that *the speedup w.r.t. the baseline is not considerably affected by the size of the underlying MCs.* This observation holds for various models and specifications.

Varying the sketch (slightly) tremendously affects performance, cf. Figure 10.7b for the performance on variants of the original sketch with a single hole substituted by one of its options. The framework performs significantly better on sketches with holes that lie in local regions of the MC. Holes relating to states all-over the MC are harder to prune. Finally, our prototype generally performs better with specifications that have multiple (conflicting) properties: Some realisations may be effectively pruned by conflicts w.r.t. property φ_1, whereas other realisations are easily pruned by conflicts w.r.t., e.g., property φ_2.

Intrusion | This sketch has 26 holes and 6800K realisations, the underlying MCs have only 500 states on average. We observe an even more significant effect of the property

thresholds on the performance than for DPM, as the number of holes is larger (recall the optimistic verifier). We obtain a speedup of factor 2200, 250 and 5 over the baseline, for thresholds $0.7{\cdot}\lambda^*$, $0.8{\cdot}\lambda^*$ and $0.9{\cdot}\lambda^*$, respectively. For $0.7{\cdot}\lambda^*$, many conflicts contain only 8 holes. Blowing up the model does not affect the obtained speedups. Differences among variants are again significant, albeit less extreme than for DPM.

Grid | This sketch is structurally different: only 6 holes in 3 commands and 1800 realisations, but MCs having 100K states on average. Observe that reaching the targets on expectation below some threshold implies that the goal must almost surely be reached. The MCs' topology and the few commands make pruning hard: our algorithm needs more than 400 iterations. Still, we obtain a 3× speedup for $\lambda = 0.98{\cdot}\lambda^*$. Pruning mostly follows from reasoning about realisations that do not reach the target almost surely. Therefore, the speedup is mostly independent of the relation between λ and λ^*.

Comparison with CEGAR | We also compared the approach with a CEGAR-prototype implementing the method of Chapter 11, which applies an abstraction-refinement loop towards, e.g., feasibility synthesis on fMCs[1]. In particular, the abstraction aggregates multiple assignments in a single model to effectively reason about a family. This approach does not support multiple property specifications (as of now), and, more importantly, the algorithm is conceptually not capable of handling constraints. For *DPM* with a single property, the CEGAR-prototype is drastically faster (for unsatisfiable feasibility problems), while on *Grid*, the CEGAR-prototype is typically (often significantly) slower. Comparing prototypes is intricate, but there is a strength and weakness of the CEGIS prototype that explains the different characteristics (for unsatisfiable instances). We conclude with a remark on satisfiable instances.

Weakness: Upon invocation, the CEGIS verifier gets exactly one assignment and is (as of now) *unaware* of other possible assignments. The verifier constructs a CE which is valid for all possible extensions (cf. Definition 10.6), even for extensions which do not correspond to any realisation. It would be better if we compute CEs that are (only) valid for the family. The following example (exaggerating *DPM*) illustrates that considering multiple realisations at once may be helpful: Consider a family with a parametric transition (hole) from the initial state and specification that requires reaching the failure state with probability smaller than $1/10$. Assume that all 100 options lead to a failure states with probability 1. CEGIS never prunes this parameter as it is contained in the CE for every assignment, and thereby considered relevant. However, knowing that all parameter assignments lead to the failure state, makes the hole trivially not relevant. Thus, all corresponding assignments may be pruned.

[1] these families are modelled as sketches. However, in contrast to the method presented here guards may not have holes.

Strength: The weakness is related to its strength: the verifier works with *one* concrete MC. An extreme example (exaggerating *Grid*) is a sketch with holes $h_0, \ldots, h_m$, where hole h_0 has options $1, \ldots, m$, and option i makes holes h_j with $j \neq i$ irrelevant (by the model topology). CEGIS considers a realisation, say $\{h_0 \mapsto i, \ldots, h_i \mapsto x, \ldots\}$, that rejects the specification. As holes h_j with $j \neq i$ are not relevant, CEGIS finds a conflict $\{h_0 \mapsto i, h_i \mapsto x\}$. Indeed, for every realisation, CEGIS is able to prune all but two holes. However, if the verifier would consider many realisations for h_0, it may (without advanced reasoning) generate much larger conflicts. Thus, considering a single realisation naturally fixes the context of the selected options and makes it clearer which holes are not relevant.

Satisfiable instances: For satisfiable instances, CEGIS may start with an accepting realisation and immediately terminate. The probability of this happening depends on the heuristic of the synthesiser and the fraction of satisfiable instances. Further research is required for a proper comparison.

Counterexample-guided inductive synthesis may extend any search-based approach to feasibility synthesis. The key idea is to reason about rejecting instantiations and conclude that other instantiations must be rejecting without analysing these instantiations separately. We have seen how this idea may be realised for fMCs, using counterexamples for MCs, and then lifted the idea to sketches and program-level counterexamples. The empirical evaluation is encouraging, but there is a lot of future work: a synthesiser selecting better instantiations, and a verifier generating small conflicts with less overhead.

11 Abstraction-Refinement for Families

Family verification with MDP model checking

Summary | This chapter considers an abstraction-based technique to analyse subfamilies of an fMC and may be used for a variety of problem statements. The chapter contains three parts: The first part presents the abstraction. The key idea is to loosen the requirement that a parameter has to have the same value in different states. Formally, this idea amounts to taking a quotient of the all-in-one MDP, which aggregates all instantiations of the family in a single MDP. The second part presents two refinement loops for two different problem statements. The refinement loop replaces the parameter partitioning scheme from Section 8.4. Our focus is on tighter integration with the verification method and the particular challenges of families rather than parameter regions. The last part contains an experimental evaluation.

Origins | The section closely resembles [15], but has been reformulated to better match notation and ideas from pMCs, and extended with intuition from [13].

Background | This section depends on the notion of fMCs as introduced in Chapter 6. Being familiar with a parameter space partitioning loop as in Section 8.4 may be helpful, but is not a prerequisite. Furthermore, this technique resembles to some extent parameter lifting for pMCs, which is the topic of Chapter 12. Whereas that chapter focusses purely on the abstraction, we also discuss an tailored refinement loop.

11.1 Abstraction

We propose a sound analysis allowing us to more efficiently analyse fMCs. More precisely, given an fMC $\mathfrak{D}$ and a family F, we focus on family verification: the question whether all $D \in \langle \mathfrak{D} \mid F \rangle$ satisfy some property φ, that is, whether $F \subseteq \mathsf{acc}_{\varphi}^{\mathfrak{D}}$. Likewise, we consider the question whether all $D \in \langle \mathfrak{D} \mid F \rangle$ do reject some property φ, that is, whether $F \subseteq \mathsf{rej}_{\varphi}^{\mathfrak{D}}$. Thus, we consider a setting in which our algorithm outputs either of the following: The family is *accepting*, *rejecting*, or *inconclusive*. The efficiency of the approach is rooted in the sound over-approximation we make: we allow our procedure to return that it does not know, i.e., we add a fourth output *unknown*. The algorithm is sound in the sense that whenever the algorithm outputs either accepting, rejecting or

inconclusive, this statement is true. In this section, we describe how the abstraction may be seen as a quotient of a well-known *all-in-one* MDP with respect to a natural equivalence relation. As a side-product, we obtain a method iterating over so-called consistent strategies.

Related work | CEGAR for probabilistic systems has been considered in [HWZ08; CV10], but they do not consider a notion of families. CEGAR for SPLs has been proposed in [CHLS+14], but in a purely qualitative setting. Further abstraction-based techniques for SPL analysis in a qualitative setting have recently been considered in [DBW19]. CEGAR-based MDP model checking for partial information strategies, a slightly different notion than restricting strategies to consistent ones, has been considered in [GR12]. In contrast to our setting, [GR12] considers history-dependent strategies and in this general setting no guarantee can be given that bounds on suprema converge [GDF14]. Game-based abstraction [KKNP10; HHWZ10]—which tends to be a finer abstraction—for finite memory strategies on POMDPs has been explored in [10].

11.1.1 All-in-one MDP

We first consider a single MDP that subsumes all instantiations of a family $\langle \mathfrak{D} \rangle$, and is equipped with an appropriate action and state labelling to identify the underlying instantiation in every part of the MDP. Indeed, the MDP is the disjoint union of all instantiations, together with an initial state from which the instantiation is nondeterministically chosen.

all-in-one MDP

Definition 11.1 (All-in-one MDP) ([GS13; RANL+15; CDKB18]) The *all-in-one MDP* of an fMC $\mathfrak{D} = (S, \iota, Y, \mathsf{Dom}_Y, \mathbf{P}, \mathsf{rew})$ and a subset of the assignments $F \subseteq \mathcal{F}_{\mathfrak{D}}$, is the MDP

$$\mathsf{AllInOne}(\mathfrak{D}, F) := (S', \iota', Act, P', \mathsf{rew}')$$

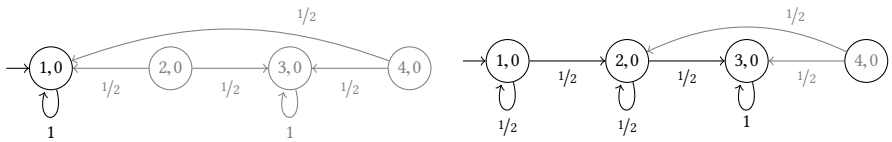
where $S' := S \times F \cup \{\iota'\}$, $Act := \{\alpha_\nu \mid \nu \in F\}$, and P' is defined by:

$$P'(\iota', \alpha_\nu, \langle s_0, \nu \rangle) := 1 \quad \text{and} \quad P'(\langle s, \nu \rangle, \alpha_\nu, \langle s', \nu \rangle) := \mathbf{P}[\nu](s, s'),$$

and rew' defined by

$$\mathsf{rew}'(\langle s, \nu \rangle) := \mathsf{rew}(s) \text{ and } \mathsf{rew}(\iota') := 0.$$

Example 11.1 (All-in-one MDP) We reconsider the fMC $\mathfrak{D}$ from Example 6.3 on page 163 with the subfamily $F := \{\nu_1, \nu_2\}$, the instantiations are also depicted



(a) $\mathfrak{D}[v_1]$ with $v_1(y_1) = s_1, v_1(y_2) = s_3$ (b) $\mathfrak{D}[v_2]$ with $v_2(y_1) = s_2, v_2(y_2) = s_3$

Figure 11.1: Subfamily for the fMC from Example 6.3 repeated. States $\langle s_i, s_j \rangle$ are labelled as i, j for conciseness.

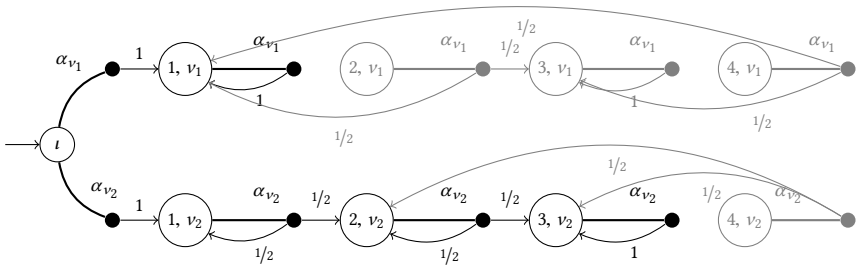


Figure 11.2: The (potentially) reachable fragment of the all-in-one MDP $\mathsf{AllInOne}(\mathfrak{D}, F)$ for $F = \{v_1, v_2\}$. States $\langle\langle s_i, s_0\rangle, v_j\rangle$ are denoted i, v_j.

in Figure 11.1. Figure 11.2 shows the all-in-one MDP $\mathsf{AllInOne}(\mathfrak{D}, F)$ for $\mathfrak{D}$ and F. States which are not reachable from the MDP's initial state are marked grey.

From the (fresh) initial state ι' of the all-in-one MDP, the choice of an action α_v corresponds to choosing the assignment v and entering the concrete MC $\mathfrak{D}[v]$:

Corollary 11.1 For the all-in-one MDP $M := \mathsf{AllInOne}(\mathfrak{D}, F)$ of an $\mathfrak{D}$ and $F \subseteq \mathcal{F}_{\mathfrak{D}}$:

$$\{M[\sigma] \mid \sigma \in \Sigma^M\} \simeq \langle \mathfrak{D} \mid F \rangle.$$

Equality does not hold as the original initial state ι of $\mathfrak{D}$ is not an initial state of $M[\sigma]$, and due to the additional set of unreachable states. For practical purposes, however, one may modify the construction and ensure equality.

Based on Corollary 11.1, the feasibility synthesis problem may be answered by checking $\mathsf{AllInOne}(\mathfrak{D}, F) \exists\models \varphi$: The answer to this query is the answer to the feasibility problem. From the MDP strategy (for the initial state) we may immediately deduce v, if the MDP is accordingly annotated. The validity problem (aka family verification) may be answered by checking $\mathsf{AllInOne}(\mathfrak{D}, F) \forall\models \varphi$.

> **Remark 11.1** Most model checking techniques for (all-in-one) MDPs determine the maximum or minimum probability (or expected reward) for *all* states. Thereby they provide the requested measure for all instantiations, and thus also a solution for, e.g., the optimal feasibility, or the complete partitioning problems.

The all-in-one MDP grows linearly in the size of the subfamily, which makes it prohibitively large. For analysis purposes, it is interesting to combine the all-in-one MDP with a symbolic representation of the state space (e.g., via decision diagrams [AKNP+00]), as the various instantiations often share a lot of structure [CDKB18]. However, when computing measures, even with a symbolic representation the all-in-one MDP is typically too large[1]. We merely use the MDP as formal starting point for the development of our abstraction.

11.1.2 Abstraction by 'forgetting'

We define a predicate abstraction that in each state of the MDP *forgets* to which instantiation a state belongs, i.e., it abstracts the second component of a state $\langle s, v\rangle$.

forgetting

> **Definition 11.2 (Forgetting)** Let $M = (S, \iota, Act, P, \mathsf{rew})$ be an all-in-one MDP for $\mathfrak{D}$ and F. *Forgetting* is the largest equivalence relation $\sim_f \subseteq S \times S$ satisfying
>
> $$\langle s, v\rangle \sim_f \langle s', v'\rangle \text{ iff } s = s', \quad \text{and} \quad \iota \sim_f (\iota^{\mathfrak{D}}, v) \text{ for all } v \in F,$$
>
> where $\iota^{\mathfrak{D}}$ denotes the initial state of $\mathfrak{D}$. Let $[s]_\sim$ denote the equivalence class w.r.t. $\sim_f$ containing state $s \in S$.
>
> quotient MDP
>
> Forgetting induces the *quotient MDP* $\mathsf{quot}(\mathfrak{D}, F) := (S_\sim, [\iota]_\sim, Act, P_\sim)$, with $S_\sim := \{[s]_\sim \mid s \in S\}$, and $P_\sim$ defined by
>
> $$P_\sim([s]_\sim, \alpha_v, [s']_\sim) := \mathbf{P}[v](s, s'),$$
>
> where $\mathbf{P}$ is the transition relation in $\mathfrak{D}$.

At each state of the quotient MDP, the actions correspond to any assignment.

> **Example 11.2** Recall the all-in-one MDP $\mathsf{AllInOne}(\mathfrak{D}, F)$ from Example 11.1. The quotient MDP $\mathsf{quot}(\mathfrak{D}, F)$ is depicted in Figure 11.3. Transitions from previously unreachable states, marked grey in Figure 11.2, are now reachable.

> **Remark 11.2 (Action space)** According to Definition 11.2, for every state $[s]_\sim$ there are $|F|$ actions. Many of these actions lead to the same distributions over successor states. In particular, in a state s, two actions with different assignments v and v' lead to the same distribution over successor states, if $v(y) = v'(y)$ for all

[1] Technically, the problem is the representation of the solution vector that tends to explode.

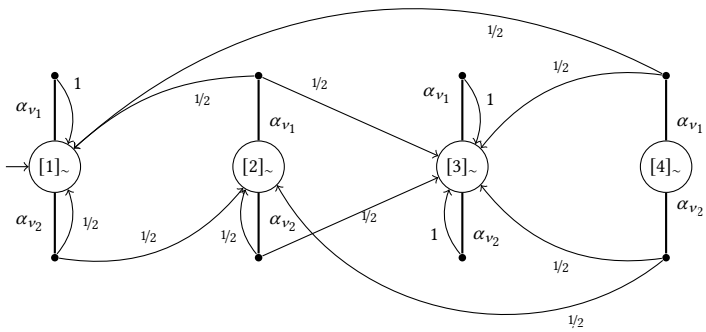
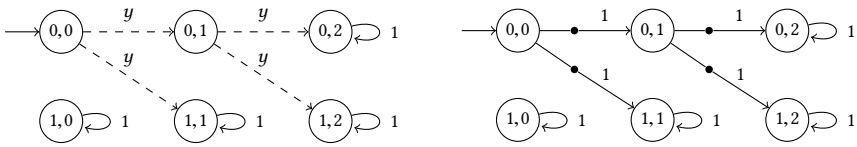


Figure 11.3: The reachable fragment of the quotient MDP $\mathsf{quot}(\mathfrak{D}, F)$ for $F = \{v_1, v_2\}$. States $[\langle\langle s_i, s_0\rangle, v_j\rangle]_\sim$ are denoted $[i]_\sim$.



(a) simplex fMC (b) corresponding quotient MDP

Figure 11.4: A simplex fMC and its quotient.

$y \in \mathsf{occurs}(s)$. To avoid this spurious blow-up of actions, we *a-priori* merge all actions yielding the same distribution. We may formalise this merge operation using the equivalences in this section. In particular, for simplex fMCs and the merge operation, the following holds for all states s in the quotient MDP (see also Figure 11.4):

$$|\mathbb{A}\mathrm{ct}(s)| \leq \max_{y \in Y} |\mathsf{Dom}_Y(y)|.$$

The quotient MDP allows that a strategy may switch assignments and thereby create induced MCs different from any MC in $\langle \mathfrak{D} \mid F \rangle$. We call such strategies inconsistent. States that are unreachable in any instantiation of the family may be reachable in the induced MC of an inconsistent strategy.

Example 11.3 Consider Figure 11.4a and recall from Example 6.5—or observe—that the state $\langle s_1, t_2 \rangle$ is unreachable in both instantiations of $\langle \mathfrak{D} \rangle$. However, in the quotient there is an obvious (but inconsistent) strategy that reaches

this state. In particular, in the quotient four states are reachable, which is more than in any instantiation. We have observed that this larger state space may become drastically larger in sketched families.

We formalise the notion of a consistent assignments with respect to a parameter, i.e., that assignments agree on a particular value for a parameter.

y-assignment-consistency

Definition 11.3 (Consistent assignments) For a family MC $\mathfrak{D}$ and a parameter $y \in Y$, *y-assignment-consistency* is an equivalence relation $\approx_y \subseteq \mathcal{F}_{\mathfrak{D}} \times \mathcal{F}_{\mathfrak{D}}$ satisfying:

$$v \approx_y v' \quad \text{iff} \quad v(y) = v'(y).$$

A strategy in the quotient MDP may select actions encoding the same parameter valuations. Then, the strategy is consistent.

·consistent

Definition 11.4 (Consistent strategy) Let $\mathfrak{D}$ be an fMC with $F \subseteq \mathcal{F}_{\mathfrak{D}}$ and parameter $y \in Y$, and let $M := \mathsf{quot}(\mathfrak{D}, F)$ be a quotient MDP. A strategy $\sigma \in \Sigma^M$ is *y-consistent* if for all $\pi, \pi' \in \Pi^M_{\mathsf{fin}}$:

$$\sigma(\pi) = \alpha_v \wedge \sigma(\pi') = \alpha_{v'} \quad \text{iff} \quad v \approx_y v' \,.$$

A strategy is *Y-consistent* (short: *consistent*) if it is y-consistent for all $y \in Y$.

Example 11.4 Recall the quotient MDP M from Example 11.2. The strategy $\sigma \in \mathfrak{S}^M$ with $\sigma([\iota]_\sim) = \alpha_{v_2}$ and $\sigma([s_2]_\sim) = \alpha_{v_1}$ is *not y_1-consistent* as $v_1(y_1) \neq v_2(y_1)$. In the induced MC $M[\sigma]$, the probability to reach state $[s_3]_\sim$ is one, while in $\mathfrak{D}[v_1]$, state $\langle s_3, s_0 \rangle$ is not reachable.

Lemma 11.2 Let $\mathfrak{D}$ be an fMC with $F \subseteq \mathcal{F}_{\mathfrak{D}}$, and $M := \mathsf{quot}(\mathfrak{D}, F)$. Then:

$$\{M[\sigma] \mid \sigma \text{ consistent strategy } \} = \langle \mathfrak{D} \mid F \rangle.$$

Proof (sketch). For $v \in F$, we construct $\sigma \in \mathfrak{S}^M$ such that $\sigma([s]_\sim) = \alpha_v$ for all s. Clearly σ is consistent and $M[\sigma] = \mathfrak{D}[v]$ is obtained via a homomorphism mapping (s, v) to $[s]_\sim$. For a consistent $\sigma \in \Sigma^M$, we construct $v \in F$. For any fMC state s with $y \in \mathsf{occurs}(s)$ and $\sigma([s]_\sim) = \alpha_{v'}$ for some $v' \in F$. We set $v(y) := v'(y)$. As σ is consistent, the assignment to $v(y)$ is unique. ■

Enumerating all consistent strategies for a quotient and analysing the induced MC provides a solution to both (optimal) feasibility and validity problems. However, there are exponentially many consistent strategies, rendering such an iterative approach inefficient in general. Nevertheless, this approach exploits an often concise representation of a family, and has practical value, as we see later in the experiments.

To complete our abstraction idea, we only need a last corollary that states that the abstraction is a sound overapproximation.

Corollary 11.3 Let $\mathfrak{D}$ be an fMC with $F \subseteq \mathcal{F}_{\mathfrak{D}}$ and φ a property. Then:

$$\mathsf{quot}(\mathfrak{D}, F) \mathrel{\forall\!\models} \varphi \quad \text{implies} \quad D \models \varphi \text{ for all } D \in \langle \mathfrak{D} \mid F \rangle.$$

Thus, analysing $\mathsf{quot}(\mathfrak{D}, F,)$ yields either accepting, rejecting or unknown as outcome, corresponding to either of the following four scenarios (for conciseness, we fix $\varphi = \mathbb{P}_{\leq\lambda}(\Diamond T)$ for target states T):

- $\mathsf{quot}(\mathfrak{D}, F) \mathrel{\forall\!\models} \varphi$, then in particular all consistent strategies yield a reachability probability less or equal λ, so the family is accepting, or
- $\mathsf{quot}(\mathfrak{D}, F) \mathrel{\forall\!\models} \neg\varphi$, then in particular all consistent strategies yield a reachability probability greater than λ, so the family is rejecting, or
- the family is inconsistent, or
- the family is accepting or rejecting, but due to the abstraction we cannot determine which of the two.

The latter scenario is the typical shortcoming of abstraction-based techniques: We have to adequately select the subfamily to consider in order for abstraction to be effective. In the next section, we apply the standard remedy for this shortcoming: refinement.

11.2 Refinement

The abstraction-based family verification may be used in combination with iterative partitioning outlined in Section 8.4 on page 230. However, when considering families, sampling is comparatively more expensive than for pMCs, and as the parameter space is not continuous, some of the assumptions behind the partitioning scheme from Section 8.4 are no longer valid for typical family benchmarks. In particular, for an fMC where each parameter has only two possible assignments, the parameter space is shaped completely different from the parameter spaces for low-dimensional continuous parameter spaces of pMCs. Moreover, parameter space partitioning as in Section 8.4 is presented with a black box verification engine. This section sheds some light on how we may realise a partitioning scheme exploiting knowledge from the verification engine. We first sketch the refinement procedure by example.

Example 11.5 Figure 11.5 illustrates the full exploration of a family using the proposed refinement loop in combination with the previously presented abstraction technique. The grid reflects a family for an fMC with two parameters, each parameter having five possible values. Each cell thus corresponds to an assignment. The various grids then correspond to successive iterations of the refinement

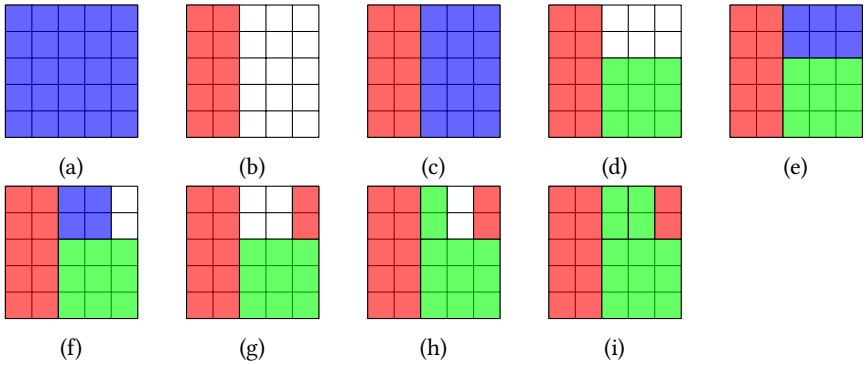


Figure 11.5: The grid depicts a family with two parameters, each with 5 possible values. Thus, each cell corresponds to an instantiation. Blue indicates a verification call that fails, green/red a covered subfamilies accepting/rejecting φ.

loop. The abstraction refinement loop first checks if all assignments accept φ on the sound abstraction. Figure 11.5a shows that the verification fails, i.e. it gives an inconclusive result. This result occurs here because the subfamily consists of both accepting and rejecting assingments. In the next step, we refine the family into two subfamilies and establish that all members in the subfamily represented by the first two columns reject φ (indicated in red in Figure 11.5b), while verifying the remaining subfamily is again inconclusive (Figure 11.5c). Thus the refinement again splits the family, this time by splitting based on the value for the second parameter (yielding a horizontal split). Partitioning that subfamily reveals that the assignments in the lower three rows accept φ (Figure 11.5d). The remaining subfamily is again inconsistent. Then, we get an inconclusive result (Figure 11.5f)—this result is due to spurious over-approximation of the abstraction technique, as indeed all assignments are accepting. The last three iterations on small subfamilies all succeed.

We refine by splitting families into subfamilies, much alike parameter space partitioning, but unlike refinement schemes such as game-based abstraction [KKNP10], where states in the quotient are split.

Splitting families may be done by means of a splitting predicate.

Definition 11.5 (Splitting) Let $\mathfrak{D}$ be an fMC and F a family. For parameter $y \in Y$ and predicate θ_y over $\mathsf{Dom}_Y(y)$, we *split* F into split

$$F_{\top}^{\theta_y} := \{v \in F \mid \theta_y(v(y))\} \quad \text{and} \quad F_{\bot}^{\theta_y} := \{v \in F \mid \neg\theta_y(v(y))\}.$$

Splitting the family and considering the subfamilies separately, rather than splitting states in the quotient MDP, is crucial for the performance of the synthesis process as we avoid rebuilding the quotient MDP in each iteration. Instead, we only restrict the actions of the MDP to the particular subfamily.

Definition 11.6 (Restricting) Let $\mathfrak{D}$ be an fMC with $F \subseteq \mathcal{F}_{\mathfrak{D}}$. Let $\mathsf{quot}(\mathfrak{D}, \mathcal{F}) = (S, \iota, Act, P, \mathsf{rew})$ be a quotient MDP for all assignments. The *restriction* of $\mathsf{quot}(\mathfrak{D}, \mathcal{F})$ to F is the MDP $\mathsf{quot}(\mathfrak{D}, \mathcal{F})_{|F} := (S, \iota, Act', P', \mathsf{rew})$ where $Act' = \{a_v \mid v \in F\}$ and P' is P restricted to $S \times Act'$. restriction

Corollary 11.4 Let $\mathfrak{D}$ be an fMC with $F \subseteq \mathcal{F}_{\mathfrak{D}}$. Then:

$$\mathsf{quot}(\mathfrak{D}, \mathcal{F})_{|F} = \mathsf{quot}(\mathfrak{D}, F).$$

Thus, we can compute the quotient for a family by just disabling some actions.

11.2.1 Refinement loops

We briefly outline two (rather similar) loops for solving the partitioning and the optimal feasibility problem, respectively.

Partitioning loop | Algorithm 11 illustrates the *partitioning* process. Recall that the goal is to partition the family $\mathcal{F}_{\mathfrak{D}}$ into families $\mathsf{acc}_{\varphi}^{\mathfrak{D}}$ and $\mathsf{rej}_{\varphi}^{\mathfrak{D}}$, accepting and rejecting a given a property, respectively. The algorithm uses a set U to store (symbolic descriptions of) subsets of v that have not been yet classified as accepting or rejecting. It starts building the quotient MDP with merged actions. That is, we never construct the all-in-one MDP, and we merge actions as discussed in Remark 11.2. For every $F \in U$, the algorithm restricts the quotient MDP. For the restricted quotient MDP, the algorithm runs standard MDP model checking to compute the max and min probability and corresponding strategies, respectively. Then, the algorithm either classifies F as accepting/rejecting, or splits it based on a suitable predicate, and updates U accordingly. We describe the splitting scheme in Section 11.2.2. The algorithm terminates if U is empty, i.e., all subfamilies have been classified. As only a finite number of subfamilies exist, termination is guaranteed. partitioning

Algorithm 11 Partitioning of F

```
function PARTITION(𝔇, F ⊆ 𝓕_𝔇, T ⊆ S, φ := ℙ_≤λ(◇T))
    rej_φ^𝔇 ← ∅, acc_φ^𝔇 ← ∅, U ← {𝓕_𝔇}
    M_∼ ←BUILDQUOTIENTMDP(𝔇, F)                    ▹ Applying Definitions 11.1 and 11.2
    while U ≠ ∅ do
        select F′ ∈ U and U ← U \ {F′}
        M ←RESTRICT(M_∼, F)                        ▹ Applying Definition 11.6
        (max, σ_max) ←SOLVEMAXMDP(M, ◇T)
        (min, σ_min) ←SOLVEMINMDP(M, ◇T)
        if max < λ then acc_φ^𝔇 ← acc_φ^𝔇 ∪ F′
        else if min > λ then rej_φ^𝔇 ← rej_φ^𝔇 ∪ F′
        else                                       ▹ min ≤ λ ≤ max
            θ_y ←SELPREDICATE(max, σ_max, min, σ_min)   ▹ See Section 11.2.2
            U ← U ∪ SPLIT(F′, θ_y)
    return acc_φ^𝔇, rej_φ^𝔇
```

maximal feasibility

Optimal feasibility loop | Algorithm 12 illustrates the *maximal feasibility* process. Recall that the goal is to find an assignment ν^* such that

$$Pr_{\mathfrak{D}[\nu^*]}(\Diamond T) = \max_{\nu \in \mathcal{F}_{\mathfrak{D}}} Pr_{\mathfrak{D}[\nu]}(\Diamond T).$$

The refinement loop for maximal feasibility in Algorithm 12 is very similar to the partitioning loop. The difference between the algorithms lies in the interpretation of the results of the underlying MDP model checking. In maximal feasibility, if the max probability for F is below the optimal value $\max^*$, F may be safely discarded. Otherwise, we check whether the corresponding strategies $\sigma_{\max}$ is consistent. If the strategy is consistent, the algorithm updates ν^* and $\max^*$, and discards (the rest of) F. If the strategy is not consistent, but $\min > \max^*$ holds, we may still update $\max^*$ and improve the pruning process, as it means that some assignment (we do not know which) in F induces a higher probability than $\max^*$. Regardless whether $\max^*$ has been updated, the algorithm has to split F based on some predicate and analyse its subfamilies as they may include the maximising assignment.

11.2.2 Informed splitting

It remains to discuss how to select the splitting predicate. We focus on the splitting technique in the partitioning loop, and end the section discussing the differences for splitting in an optimal synthesis loop.

If model checking the quotient MDP $\mathsf{quot}(\mathfrak{D}, F)$ yields an inconclusive result, i.e., if it cannot classify the family F as accepting or rejecting, we split F. The splitting

Algorithm 12 Optimal feasibility

```
1:  function MAXFEASIBILITY(𝔇, F ⊆ 𝓕_𝔇, T ⊆ S)
2:      max* ← −∞, U ← {F}
3:      M_∼ ←BUILDQUOTIENTMDP(𝔇, F)                    ▷ Applying Definitions 11.1 and 11.2
4:      while U ≠ ∅ do
5:          select F′ ∈ U and U ← U \ {F′}
6:          M ←RESTRICT(M_∼, F)                        ▷ Applying Definition 11.6
7:          (max, σ_max) ←SOLVEMAXMDP(M, ◊T)
8:          (min, σ_min) ←SOLVEMINMDP(M, ◊T)
9:          if max > max* then
10:             if ISCONSISTENT(σ_max)) then
11:                 v* ← q_max, max* ← max
12:             else
13:                 if min > max* then max* ← min
14:                 θ_y ←SELPREDICATE(max, σ_max, min, σ_min)   ▷ See Section 11.2.2
15:                 U ← U ∪ SPLIT(F′, θ_y)
16:     return v*
```

operation chooses a suitable parameter $y \in Y$ and predicate θ_y that partitions the family F into $F_\top^{\theta_y}$ and $F_\bot^{\theta_y}$ (see Definition 11.5). A good splitting scheme globally reduces the number of model-checking calls required to classify all $v \in F$.

For ease of presentation, we assume that we first select the parameter y and then the predicate. Naturally, these two steps are interdependent. The two key aspects to locally determine a good y are:

- the *variance*, that is, how the splitting may narrow the difference between $\max := Pr^{\max}_{M'}(\Diamond T)$ and $\min := Pr^{\min}_{M'}(\Diamond T)$ for both $M' \in \{\mathsf{quot}(\mathfrak{D}, F_\top^{\theta_y}), \mathsf{quot}(\mathfrak{D}, F_\bot^{\theta_y})\}$, and
- the *consistency*, that is, how the splitting may reduce the inconsistency of the strategies $\sigma_{\max}$ and $\sigma_{\min}$ obtained in future MDPs.

Parameter scoring | The aspects described above cannot be evaluated precisely without applying all the split operations and solving the new MDPs $\mathsf{quot}(\mathfrak{D}, F_\top^{\theta_y})$ and $\mathsf{quot}(\mathfrak{D}, F_\bot^{\theta_y})$ for all kind of predicates θ_y. Therefore, we propose an efficient scheme selecting $y \in Y$ and θ_y based on a light-weight analysis of the model-checking results for $\mathsf{quot}(\mathfrak{D}, F)$. The scheme computes two *scores*—VARIANCE(y) and INCONSISTENCY(y)—that estimate the influence of y on the two key aspects. For any y, the scores are accumulated over all *important states* s (reachable via $\sigma_{\max}$ or $\sigma_{\min}$, respectively) where $y \in \mathsf{occurs}(s)$. A state s is important for F and some $\delta \in \mathbb{R}_{\geq 0}$ if

$$\frac{Pr^{\max}_{\mathsf{quot}(\mathfrak{D},F)}(s \models \Diamond T) - Pr^{\min}_{\mathsf{quot}(\mathfrak{D},F)}(s \models \Diamond T)}{Pr^{\max}_{\mathsf{quot}(\mathfrak{D},F)}(\Diamond T) - Pr^{\min}_{\mathsf{quot}(\mathfrak{D},F)}(\Diamond T)} \geq \delta.$$

To reduce the overhead of computing the scores, we compress the strategy representation. In particular, for $\sigma_{\max}$ and every $y \in Y$, we extract a histogram $C^y_{\max} \colon \mathsf{Dom}_Y(y) \to \mathbb{N}$, where $C^y_{\max}(t)$ is the number of important states for which $\sigma_{\max}(s) = \alpha_\nu$ with $\nu(y) = t$, i.e. the number of states s where the strategy takes an action a_ν that instantiates the parameter y to the value t. The histogram $C^y_{\min}$ analogously compresses $\sigma_{\min}$. We define

$$\textsc{variance}(y) \coloneqq \sum_{y \in \mathsf{Dom}_Y(y)} |C^y_{\max}(t) - C^y_{\min}(t)|,$$

leading to high scores if the two strategies vary a lot. Furthermore, we define

$$\textsc{inconsistency}(y) \coloneqq \textsc{size}\left(C^y_{\max}\right) \cdot \textsc{max}\left(C^y_{\max}\right) + \textsc{size}\left(C^y_{\min}\right) \cdot \textsc{max}\left(C^y_{\min}\right),$$

where

$$\textsc{size}\,(C) \coloneqq |\{y \in \mathsf{Dom}_Y(y) \mid C(y) > 0\}| - 1, \text{ and } \textsc{max}\,(C) \coloneqq \max_{y \in \mathsf{Dom}_Y(y)} C(y),$$

leading to high scores if the parameter has clear favourites for $\sigma_{\max}$ and $\sigma_{\min}$, but simultaneously the strategies also 'select' other values for y.

Which parameter do we use for splitting? | As indicated, we consider different strategies for the different refinement loops: For partitioning, we favour the impact on the variance as we principally do not need consistent strategies. For optimal feasibility, we favour the impact on the consistency, as we need a consistent strategy inducing the max probability.

How to split? | What remains is to select a predicate. We select the predicate hoping to reduce the variance: The scheme selects $S' \subset \mathsf{Dom}_Y(y)$ with $|S'| = \frac{1}{2}\lceil|\mathsf{Dom}_Y(y)|\rceil$, containing those s for which $C^y_{\max}(s) - C^y_{\min}(s)$ is the largest. The goal is to get a family $F^{\theta_y}_{\top}$ such that all members induce a large reachability probability to reach the target (the ones including S' for parameter y) and the complement $F^{\theta_y}_{\bot}$ with all members inducing a small probability.

11.3 Empirical Evaluation

11.3.1 Set-up

Implementation | We implemented the proposed methods as a python prototype using STORM [9]. In particular, we use the STORM python API for model-adaption, -building, and -checking as well as for strategy extraction. For SMT solving, we use

Z3 [MB08]. The tool-chain takes a sketch. Via adequate annotations, we identify the parameter values that yield a particular action. The annotations are key to interpret the strategies and to restrict the quotient without rebuilding. For the all-in-one approach, we construct an MC with several initial states, which allows the usage of the faster equation-system based solvers in STORM.

Environment | All experiments were executed on a Macbook MF839LL/A with 8GB RAM memory limit and a 12h time out. STORM used 4 threads for the symbolic engine and one thread for all sparse computations.

Research questions | The goal of the experimental evaluation is to evaluate: *How does the proposed MDP-based abstraction cope with the inherent complexity of the synthesis problem?* To answer this question, we compare various approaches:

1. iterating over the instantiations (one-by-one).
2. building and analysing the all-in-one MDP.
3. iterating over the consistent strategies of the quotient MDP.
4. encoding consistent strategies into a linear fragment of ETR, similar to Section 4.3, and use an SMT-solver to find such strategies.
5. using the refinement-loop outlined in Section 11.2.

Approaches 1 and 2 [CDKB18], represent state-of-the-art solutions and are our baseline algorithms. Experiments show that the performance of the MDP abstraction significantly varies for different case studies. Thus, we consider benchmarks from various application domains to *identify the key characteristics of the synthesis problems affecting the performance of our approach.* We give a preliminary comparison to the approach from Chapter 10 at the end of that chapter.

Benchmarks description. | *Maze* is a planning problem similar to Example 7.1 and originally formulated as a POMDP. We use the correspondence to fMCs as outlined in Section 7.1 to let the family describe a set of induced MCs for small-memory observation-based strategies. For these strategies, we are interested in the expected time that it takes them to navigate to the exit of the maze.

Pole considers balancing a pole in a noisy and unknown environment (motivated by [ABCK+18; CCMN+12]). At deploy time, the controller has a prior over a finite set of environment behaviours, and should optimise the expected behaviour without depending on the actual (hidden) environment. The family describes strategies for the actuator that do not depend on the hidden information about the environment. We are interested in the expected time until failure.

Herman is an asynchronous encoding of the distributed Herman protocol for self-stabilising rings [Her90; KNP12]. The protocol is extended with a bit of memory for each

Table 11.1: Benchmarks and timings for approaches 1–3

				Member size		Quotient size			Run time		
Bench.	Range	$\lvert Y\rvert$	$\lvert\mathcal{F}_{\mathfrak{D}}\rvert$	Avg. $\lvert S\rvert$	Avg. $\lvert T\rvert$	$\lvert S\rvert$	$\lvert A\rvert$	$\lvert T\rvert$	1-by-1	All-in-1	Sched. Enum.
Pole	[3.35,3.82]	17	1327104	5689	16896	6793	7897	22416	130k*	MO	26k
Maze	[9.8,9800]	20	1048576	134	211	203	277	409	28k	TO	2.7k
Herman	[1.86,2.44]	9	576	5287	6948	21313	102657	184096	55	72	246
DPM	[68,210]	9	32768	5572	18147	35154	66096	160146	2.9k	MO	7.2k
BSN	[0,0.988]	10	1024	116	196	382	457	762	31	2	2

station in the ring and the choice to flip various unfair coins. Nodes in the ring are anonymous, they all behave equivalently (but may change their local memory based on local events). The family describes variations of memory-updates and coin-selection, but preserves anonymity. We are interested in the expected time until stabilisation.

DPM considers a partial information strategy for a disk power manager, as outlined in Section 7.3.1.1. We are interested in the expected energy consumption.

BSN (Body sensor network, [RANL+15]) describes a network of connected sensors that identify health-critical situations, as outlined in Section 7.3.1.2. We are interested in the reliability. We ignore some of the dependencies between features, modelled as constraints in our sketching language, as this approach does not (yet) support constraints. We thereby extended the family.

Benchmark characteristics | Table 11.1 shows the relevant statistics for each benchmark: the benchmark name, the (approximate) range of the min and max probability/reward for the given family, the number of non-singleton parameters $|Y|$, and the number of family members $|\mathcal{F}_{\mathfrak{D}}|$. Then, for the family members the average number of states and transitions of the MCs, and the states, actions ($= \sum_{s\in S} |\mathbb{A}\mathrm{ct}(s)|$), and transitions of the quotient MDP. Finally, the table lists in seconds the run time of the base-line algorithms and the consistent strategy enumeration[1]. The baseline algorithms employ the one-by-one and the all-in-one technique, using either a BDD or a sparse matrix representation. We report the best results. MOs indicate breaking the memory limit. Only the all-in-one approach required significant memory. As expected, the SMT-based implementation provides an inferior performance and thus we do not report its results.

11.3.2 Evaluation

We focus primarily on partitioning as it allows a compact presentation of the key aspects. Below, we provide some remarks about the performance for the max and feasibility synthesis.

[1]Values with a * are estimated by sampling a large fraction of the family.

Table 11.2: Results for partitioning via abstraction-refinement

Inst	λ	#Below	#Subf below	#Above	#Subf above	Singles	#Iter	Time	Build	Check	Analysis	Speedup
Pole	3.37	697	176	1326407	2186	920	4723	308	117	60	118	**421**
	3.73	1307077	7854	20027	3279	1294	22265	1.7k	576	317	396	**77**
	3.76	1322181	3140	4923	1025	1022	8329	584	187	114	197	**222**
	3.79	1326502	572	602	123	74	1389	58	23	10	23	**2.2k**
Maze	10	4	3	1048572	92	4	189	5	<1	3	< 1	**26k**
	20	4247	2297	1044329	4637	3400	13867	114	21	43	29	**246**
	30	18188	9934	1030388	18004	14010	55875	608	80	127	270	**46**
	8000	1046285	846	2291	1125	969	3941	136	9	106	13	**1.0k**
Herman	1.9	6	6	570	368	320	747	333	303	11	18	**0.2**
	1.71	0	0	576	258	184	515	232	206	8	17	**0.3**
DPM	80	160	141	32608	1292	356	2865	1.0k	602	322	64	**3**
	70	6	6	32762	443	40	897	380	190	156	32	**8**
	60	0	0	32768	104	6	207	99	42	48	8	**29**
BSN	.965	544	81	480	81	25	321	2	<1	<1	<1	**1**
	.985	994	41	30	8	5	97	<1	<1	<1	<1	**3**

11.3.2.1 Results

Table 11.2 shows results for partitioning. The first two columns indicate the benchmark and the various thresholds. For each threshold λ, the table lists the number of assignments below (above) λ, each with the number of subfamilies that together contain these assignments, and the number of singleton subfamilies that were considered. The last table part gives the number of iterations of the loop in Algorithm 11 and timing information (total, build/restrict times, model checking times, strategy analysis times). The last column gives the speed-up over the best baseline.

11.3.2.2 Key observations

The speedups drastically vary, showing that the MDP abstraction often achieves a superior performance, but may also lead to a performance degradation. We identify four key factors.

Iterations | As typical for CEGAR approaches, the key characteristic of the benchmark that affects the performance is the number of iterations in the refinement loop. The abstract action introduces an overhead per iteration caused by performing two MDP verification calls and by the strategy analysis. The run time for *BSN*, with a small $|\mathcal{F}_{\mathfrak{D}}|$ is actually significantly affected by the initialisation of various data structures; thus only a small speedup is achieved.

Quotient size | The size of the quotient, compared to the average size of the family members, is relevant too. The quotient includes at least all reachable states of all family members and may be significantly larger if an inconsistent strategies reaches states

which are unreachable under any consistent strategy. The existence of such states is a common artefact from encoding families in sketches. Table 11.1 indicates that we obtain a very compact representation for *Maze* and *Pole*.

Thresholds | The most important aspect is the threshold λ. Clearly, if $\mathcal{F}_{\mathfrak{D}}$ is accepting or rejecting, then the abstraction performs better the further λ is away from either optimum as it increases the chance that even with the over-approximation, the abstraction quickly classifies families as accepting/rejecting. In the remainder, we assume that $\mathcal{F}_{\mathfrak{D}}$ is inconclusive. If λ is between the optima, then the closer it is to either of the optima, the fewer iterations the abstraction-refinement needs, which directly improves the performance. We emphasise that in various domains, thresholds that ask for close-to-optimal solutions are indeed of highest relevance as they typically represent the system designs developers are most interested in [SB10].

Why do thresholds affect the number of iterations? Consider an fMC $\mathfrak{D}$ with $\mathsf{Dom}_Y(y) = \{0, 1\}$ for each $y \in Y$. Geometrically, the set $\mathcal{F}_{\mathfrak{D}}$ may be visualised as $|Y|$-dimensional cube. The cube-vertices reflect family members. Assume for simplicity that one of these vertices is optimal with respect to the specification. Especially in families where parameters are equally relevant[1], the induced probability of a vertex roughly corresponds to the Manhattan distance to the optimal vertex. Thus, vertices above the threshold induce a diagonal hyperplane, which our splitting method approximates with orthogonal splits. Splitting diagonally is not possible, as it would induce optimising over strategies with additional dependencies between actions. Consequently, we need more and more splits the more the diagonal goes through the middle of the cube. *Even when splitting optimally, there is a combinatorial blow-up in the required splits when the threshold is further from the optimal values.* Another effect is that thresholds far from optima are more affected by the over-approximation of the MDP model-checking results and thus yield more inconclusive answers.

Refinement policy | So far, we reasoned about optimal splits. Due to the computational overhead, our policy cannot ensure optimal splits. Instead, the policy depends mostly on information encoded in the computed MDP strategies. *In models where the optimal parameter value heavily depends on the state, the obtained strategies are highly inconsistent and carry only limited information for splitting.* Consequently, in such benchmarks we split sub-optimally. The sub-optimality has a major impact on the performance for *Herman* as all obtained strategies are highly inconsistent – they take a different coin for each node in the network modelled by Herman, which is good

[1] Relevance may be intuitively measured by the regret of switching the parameter value in an assignment, i.e., how much probability mass reaching the target we expect to loose if we take the suboptimal choice for this parameter.

to speed up the stabilisation of the network, but not consistent with the anonymity requirement of the modelled protocol.

11.3.2.3 Summary

Abstraction is not a silver bullet, also not in this setting. It has a lot of potential in partitioning when most (or even all) assignments are accepting or rejecting. Consequently, *feasibility synthesis with unsatisfiable specifications is typically handled well by the abstraction-refinement scheme*, while unsatisfiability is the worst-case for enumeration-based approaches. Likewise, *optimal feasibility synthesis* may be understood as partitioning with a shifting threshold $\max^*$: If the $\max^*$ is quickly set close to max, MDP abstraction yields superior performance. Roughly, we quickly approximate $\max^*$ when some of the parameter values are clearly beneficial for the property.

We propose an abstraction-based approach to family verification, which removes the dependencies between parameters and thereby simplifies an fMC to an MDP. This abstraction is typically too coarse, and in the second part of this chapter, we show how to create a refinement loop for partitioning that uses the MDP model checking results to guide the refinement. The empirical evaluation shows that this approach is promising and a definitive improvement over the state-of-the-art. Still, sometimes the abstraction is too coarse—thereby demanding too many splits,and the abstraction may yield an MDP that is too large, generating significant overhead.

12 Parameter Lifting

Abstraction-based region verification

Summary | Parameter lifting is a technique that abstracts parametric Markov models into parameter-free Markov models with one additional source of nondeterminism. Conceptually, this abstraction consists of two parts: First, the challenging part of parameter synthesis is stripped away: We replace parameter occurrences by fresh copies, thereby eliminating any dependencies between different states. We obtain dependency-free Markov models. Second, we replace the parameters equivalently by nondeterministic actions. We show the approach for pMCs and then for pMDPs.

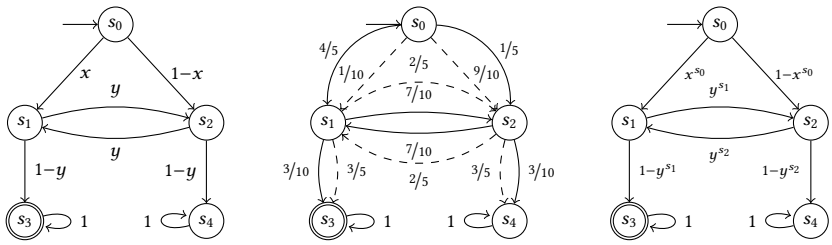
Origins | This chapter describes an abstraction-based technique called parameter lifting, as described in [5]. Various parts of this chapter closely resemble [17], but we adapted the presentation, taking into account all the constructions provided by Chapter 5 and give more examples. We provide an updated experimental section.

Background | We consider pMCs and pMDPs as presented in the first part of Chapter 4, and use results from model checking their parameter-free versions (Chapter 3). We recommend to read Section 5.1, to get a better understanding of some of the essential preprocessing steps and subclasses that we use throughout this chapter. The method shares various ideas with the abstraction in Chapter 11, but the formalisation is quite different.

12.1 Parameter Lifting for Markov chains

We discuss an abstraction-based approach to the region verification problem for pMCs and ε-preserving regions, that is, given a pMC $\mathcal{D}$, an ε-preserving region R, and a property φ, we want to determine whether $\langle \mathcal{D} \mid R \rangle \models \varphi$. We focus on reachability probabilities. The straightforward lifting to expected rewards is treated in [17].

In a nutshell, the idea presented in this section is to abstract a pMC into an MDP whose minimal (maximal) reachability probability under-approximates (over-approximates) the reachability probability of the pMC.



(a) pMC $\mathcal{D}$ (b) MDP $\mathsf{sub}_{\mathcal{D}}(R)$ (c) $\mathsf{rel}(\mathcal{D})$

Figure 12.1: A pMC, the substitution MDP, and the relaxed pMC.

> **Remark 12.1** Equivalently, we abstract pMCs into so-called interval Markov chains—which are special pMCs, recall Section 5.1.1.3. These interval Markov chains may be solved by MDPs by considering the parameter choices the description of the nondeterminism, as in Definition 5.13.

> **Example 12.1** Consider the pMC $\mathcal{D}$ in Figure 12.1a, where each state has a single occurring parameter, and a region $R = [1/10, 4/5] \times [2/5, 7/10]$. The method creates the MDP in Figure 12.1b, where different types of arrows reflect different actions. The MDP is obtained by taking the state space from the pMC, with in each state two actions: One reflecting the lower bound of the interval of the occurring parameter, and the other reflecting the upper bound. Thus, in s_0 we have two actions, one reflecting x being assigned $1/10$, one reflecting x being assigned $4/5$. Model checking on this MDP yields a maximal probability of $47/60$. From this result, we infer that
>
> $$\max_{D \in \langle \mathcal{D} | R \rangle} Pr_D(\Diamond T) \leq 47/60.$$

Details follow below.

> **Remark 12.2** Readers familiar with the POMDP interpretation of pMCs, in Section 7.1, may find that the essential idea is to abstract the POMDP into a (fully observable) MDP. In a refinement loop such as the parameter space partitioning loop (see Section 8.4), the over-approximation induced by the abstraction vanishes.

12.1.1 Idea

Verification as a special form of optimal synthesis | For an ε-preserving closed region R, there is an instantiation $\mathsf{val} \in R$ inducing the maximal (or minimal) reach-

ability probability, cf. Corollary 5.17. Solving the optimal feasibility problem is then sufficient to solve the region verification problem. That is, to infer that R is accepting for $\varphi = \mathbb{P}_{\leq\lambda}(\Diamond T)$, it suffices to show that the *maximal* reachability probability over all valuations is at most λ:

$$R \subseteq \mathsf{acc}_\varphi^{\mathcal{D}} \quad \text{iff} \quad \big(\max_{\mathsf{val}\in R} Pr_{\mathcal{D}[\mathsf{val}]}(\Diamond T)\big) \leq \lambda, \text{and}$$
$$R \subseteq \mathsf{rej}_\varphi^{\mathcal{D}} \quad \text{iff} \quad \big(\min_{\mathsf{val}\in R} Pr_{\mathcal{D}[\mathsf{val}]}(\Diamond T)\big) > \lambda.$$

Example 12.2 Reconsider Example 12.1. Inspection of the pMC in Figure 12.1a yields that the maximum is $22/35$, induced by $\mathsf{val} = (4/5, 2/5)$. We may thus conclude that R accepts any $\varphi := \mathbb{P}_{\leq\lambda}(\Diamond T)$ with $\lambda \geq 22/35$. While this optimal valuation on the vertex of the region may be considered easy to find, the optimal valuation is not always a vertex. If we consider $R' = [1/10, 1/2] \times [2/5, 7/10]$ then the maximum is $1/2$ and obtained by $\mathsf{val}' = (1/2, 9/16)$, which is not a vertex of the region.

Local optimisation | Solving the optimal feasibility problem is hard, even for a pMC with just initial, sink and target state: it amounts to optimising the transition probability, which in general is an arbitrary rational function. Thus, we restrict ourselves to a subclass where the optimal values along a single transition are at the vertices of a linear region.

Definition 12.1 (Locally monotone pMCs) A pMC $\mathcal{D}$ is *locally monotone* iff for all $s \in S$ there is a multilinear polynomial $g_s \in \mathbb{Q}[X]$ satisfying ·locally monotone

$$\mathcal{P}(s, s') \in \{f/g_s \mid f \in \mathbb{Q}[X] \text{ is multilinear}\}$$

for all $s' \in S$.

Lemma 12.1 For a locally monotone pMC $\mathcal{D}$, and a closed rectangular region R we have that for all $s, s' \in S$:

$$\max_{\mathsf{val}\in R} \mathcal{P}(s, s')[\mathsf{val}] = \max_{\mathsf{val}\in\mathcal{B}(R)} \mathcal{P}(s, s')[\mathsf{val}],$$

where $\mathcal{B}(R) = \{\mathsf{val} \mid \mathsf{val}(x) \in \mathcal{B}_R \ \forall x \in X\}$.

Locally monotone pMCs include simple(x) pMCs, as well as the pMCs that occur as the embedded pMCs for parametric CTMCs. (Optimal) feasibility for simplex pMCs—and thus for locally monotone pMCs—is still a hard task, due to dependencies where parameters occur at multiple states. Therefore, observe that Example 12.2, or the illustrative example in Example 4.17 on page 78, consider simple pMCs.

The key idea, similar to [BCDS13], is to locally maximise the probabilities. However, instead of computing the actual optimum, we compute an over-approximation by taking the optimal bounds for each parameter in every state independently.

Example 12.3 Consider the pMC in Figure 12.1a. In state s_0, it is beneficial if $\mathsf{val}(x)$ is large. In state s_1, it is beneficial if $\mathsf{val}(y)$ is small. In state s_2, it is beneficial if $\mathsf{val}(y)$ is large. Such a valuation does not exist—but considering this valuation surely gives an upper bound on the reachability probability of reaching s_3.

Technically, we realise this idea by casting the problem into a substitution MDP, where the strategy now optimises in every state. We construct the MDP in two steps: first we construct the *relaxation*, then the *substitution* (also called *lifting*).

Assumptions | Concluding, we make the following assumptions in this chapter:

- We restrict to a (graph-preserving) region R which is (i) rectangular, and (ii) closed. This restriction makes the bounds of the parameters independent of other parameter valuations, and ensures that the maximum over the region exists.
- We restrict the pMC $\mathcal{D}$ to be locally monotone, to exclude difficulties from analysing single transitions.

Following Corollary 5.19, this method is also applicable for (arbitrary) rectangular well-defined subregions on acyclic models.

12.1.2 Relaxation

Intuitively, the relaxation $\mathsf{rel}(\mathcal{D})$ is a pMC that arises from $\mathcal{D}$ by equipping each state with its own parameters, thereby eliminating parameter dependencies between different states (if any).

Example 12.4 The relaxation of the pMC in Figure 12.1a is given in Figure 12.1c.

relaxation

Definition 12.2 (Relaxation) The *relaxation* of pMC $\mathcal{D} = (S, \iota, X, \mathcal{P})$ is the pMC $\mathsf{rel}(\mathcal{D}) := (S, \iota, \mathsf{rel}_{\mathcal{D}}(X), \mathcal{P}')$ with $\mathsf{rel}_{\mathcal{D}}(X) := \{x_i^s \mid x_i \in X, s \in S\}$ and $\mathcal{P}'(s, s') := \mathcal{P}(s, s')[x_1, \ldots, x_n / x_1^s, \ldots, x_n^s]$.

We extend an instantiation $\mathsf{val} \in \mathsf{Val}_{\mathcal{D}}^{\mathrm{gp}}$ to the *relaxed instantiation* $\mathsf{rel}_{\mathcal{D}}(\mathsf{val}) \in \mathsf{Val}_{\mathsf{rel}(\mathcal{D})}^{\mathrm{gp}}$ by $\mathsf{rel}_{\mathcal{D}}(\mathsf{val})(x_i^s) = \mathsf{val}(x_i)$ for every s. It holds: for all val, $\mathcal{D}[\mathsf{val}] = \mathsf{rel}(\mathcal{D})[\mathsf{rel}_{\mathcal{D}}(\mathsf{val})]$. We lift the relaxation to regions such that $\mathcal{B}(x_i^s) = \mathcal{B}(x_i)$ for all s, i.e., $\mathsf{rel}_{\mathcal{D}}(R) = \bigtimes_{x_i^s \in \mathsf{rel}_{\mathcal{D}}(X)} I(x_i)$. We drop the subscript $\mathcal{D}$, whenever it is clear from the context. Valuations in the relaxation may not correspond to a valuation in the original pMC. However, every instantiation generated by the original pMC is also an instantiation of its relaxation:

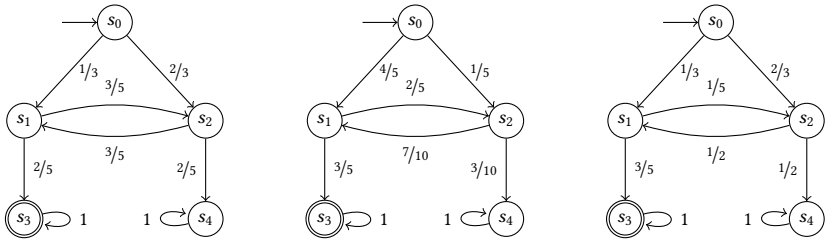


Figure 12.2: Instantiations of $\mathcal{D}$ and $\mathsf{rel}(\mathcal{D})$.

Proposition 12.2 For a pMC $\mathcal{D}$ and a graph-preserving region R:

$$\langle \mathcal{D} \mid R \rangle \subseteq \langle \mathsf{rel}(\mathcal{D}) \mid \mathsf{rel}(R) \rangle.$$

Example 12.5 The MC in Figure 12.2a is an instantiation of $\mathcal{D}$ and $\mathsf{rel}(\mathcal{D})$. The MCs in Figures 12.2b and 12.2c are instantiations of $\mathsf{rel}(\mathcal{D})$, but not of $\mathcal{D}$.

Thus, the maximal reachability probability over all instantiations of $\mathcal{D}$ within R is bounded by the maximum over the instantiations of $\mathsf{rel}(\mathcal{D})$ within $\mathsf{rel}(R)$.

Corollary 12.3 For pMC $\mathcal{D}$ and region R:

$$\max_{D \in \langle \mathcal{D} \mid R \rangle} Pr_D(\Diamond T) \leq \max_{D \in \langle \mathsf{rel}(\mathcal{D}) \mid \mathsf{rel}(R) \rangle} Pr_D(\Diamond T),$$

and consequently:

$$\mathsf{rel}(R) \subseteq \mathsf{acc}_{\varphi}^{\mathsf{rel}(\mathcal{D})} \quad \text{implies} \quad R \subseteq \mathsf{acc}_{\varphi}^{\mathcal{D}}.$$

As $\mathcal{D}$ is locally monotone, and there are no parameter dependencies, the maximum reachability probability is relatively easy to find: We only need to consider valuations val that set the value of each parameter to either the lowest or highest possible value, i.e., $\mathsf{val}(x_i^s) \in \mathcal{B}(x_i^s)$ for all $x_i^s \in \mathsf{rel}(X)$, cf. Section 5.1.1.3. A longer discussion is included in [17]. However, we do not search for this probability directly on the pMC. Instead, we lift this pMC to an MDP.

12.1.3 Lifting

The *lifting* step reformulates the relaxed pMC $\mathsf{rel}(\mathcal{D})$ into an MDP M. lifting

substitution MDP

Definition 12.3 (Substitution (pMCs)) For pMC $\mathcal{D} = (S, \iota, X, \mathcal{P})$ and region R, let the *substitution MDP* $\mathsf{sub}_R(\mathcal{D}) \coloneqq (S, \iota, Act_{\mathsf{sub}}, P_{\mathsf{sub}})$ with

$$Act_{\mathsf{sub}} \coloneqq \biguplus_{s \in S} Act_s \text{ where } Act_s \coloneqq \{u \colon \mathsf{occurs}(s) \to \mathbb{R} \mid u(x) \in \mathcal{B}(x)\ \forall x \in \mathsf{occurs}(s)\},$$

and

$$P_{\mathsf{sub}}(s, u, s') \coloneqq \begin{cases} \mathcal{P}(s, s')[u] & \text{if } u \in Act_s, \\ 0 & \text{otherwise.} \end{cases}$$

·substitution

be the *(parameter-)substitution of $\mathcal{D}$ and R.*

Thus, choosing action u in s corresponds to assigning one of the extremal values $\mathcal{B}(x_i)$ to the parameters x_i^s. The number of outgoing actions from state s is therefore $2^{|\mathsf{occurs}(s)|}$. The nondeterministic choices introduced by the substitution only depend on the bounds $\mathcal{B}(x_i)$ of the parameters x_i. Since the bounds of the parameters x_i^s in $\mathsf{rel}(R)$ agree with the bounds of x_i in R, we have

$$\mathsf{sub}_{\mathsf{rel}(R)}(\mathsf{rel}(\mathcal{D})) = \mathsf{sub}_R(\mathcal{D}) \quad \text{for all graph-preserving } R. \tag{12.1}$$

Thus, the relaxation is a tool for showing the correctness of the approach below, but the equality proves that the substitution MDPs coincide, and thus, relaxation is superfluous.

Example 12.6 Consider the pMC $\mathcal{D}$ in Figure 12.1a with region $R \coloneqq [1/10, 4/5] \times [2/5, 7/10]$ as in Example 12.1. The substitution MDP $\mathsf{sub}_R(\mathcal{D})$ is given in Figure 12.1b.

Proposition 12.4 Let $\mathcal{D}$ be a pMC with region $R \subseteq \mathsf{Val}_{\mathcal{D}}^{\mathsf{gp}}$. Then:

$$\{\mathsf{sub}_R(\mathcal{D})[\sigma] \mid \sigma \in \mathfrak{S}_m^{\mathsf{sub}_R(\mathcal{D})}\} \simeq \langle \mathsf{rel}(\mathcal{D}) \mid \mathsf{rel}(R) \rangle.$$

Although we need all (randomised) memoryless strategies to correspond to the induced MCs of $\mathsf{rel}(\mathcal{D})$, to obtain the maximum, it suffices to consider memoryless deterministic strategies (Proposition 3.6). Together with Proposition 12.2 we obtain:

Corollary 12.5 For a pMC $\mathcal{D}$ and a graph-preserving region R:

$$\langle \mathcal{D} \mid R \rangle \subseteq \{\mathsf{sub}_R(\mathcal{D})[\sigma] \mid \sigma \in \mathfrak{S}_m^{\mathsf{sub}_R(\mathcal{D})}\}.$$

Considering (12.1) and Corollary 12.3 yields:

Algorithm 13 Parameter lifting

```
function REACHABILITY(pMC 𝒟, region R, property φ )
    Construct sub_R(𝒟)
    if sub_R(𝒟) ∀⊨ φ then                    ▹ via standard MDP model checking procedures
        return accept
    else if sub_R(𝒟) ∀⊨ ¬φ then              ▹ via standard MDP model checking procedures
        return reject
    else
        return unknown
```


Corollary 12.6 For a pMC $\mathcal{D}$, a graph-preserving region R, and a set T of target states of $\mathcal{D}$:

$$\max_{\mathsf{val} \in R} Pr_{\mathcal{D}[\mathsf{val}]}(\Diamond T) \;\leq\; \max_{\sigma \in \Sigma} Pr^{\sigma}_{\mathsf{sub}_R(\mathcal{D})}(\Diamond T), \text{ and}$$
$$\min_{\mathsf{val} \in R} Pr_{\mathcal{D}[\mathsf{val}]}(\Diamond T) \;\geq\; \min_{\sigma \in \Sigma} Pr^{\sigma}_{\mathsf{sub}_R(\mathcal{D})}(\Diamond T).$$

A direct consequence of these statements yields:

Theorem 12.7 Let $\mathcal{D}$ be a pMC, R a graph-preserving region, φ a reachability property. Then it holds:

$$\mathsf{sub}_R(\mathcal{D}) \,\forall\!\models \varphi \quad \text{implies} \quad R \subseteq \mathsf{acc}^{\mathcal{D}}_{\varphi}, \text{ and}$$
$$\mathsf{sub}_R(\mathcal{D}) \,\forall\!\models \neg\varphi \quad \text{implies} \quad R \subseteq \mathsf{rej}^{\mathcal{D}}_{\varphi}.$$

Hence, we can deduce via Algorithm 13 whether R is accepting by applying standard MDP model checking, cf. Chapter 3.

Example 12.7 Recall Example 12.1 with the $\mathcal{D}$ in Figure 12.1b and region $R = [1/10, 4/5] \times [2/5, 7/10]$, and let us consider $\varphi = \mathbb{P}_{\leq 3/4}(\Diamond T)$. Recall from Example 12.2 that the region is accepting φ. The MDP in Figure 12.1b corresponds to $\mathsf{sub}_R(\mathcal{D})$; model checking (assuming a maximising strategy) yields that the maximal probability is bounded from above by $47/60$. We cannot conclude that φ is accepting. Model checking assuming a minimising strategy yields that the minimal probability is bounded from below by $23/70$, and thus we cannot conclude that φ is rejecting.

If the over-approximation by region R is too coarse for a conclusive answer, region R may be refined by splitting—as in Section 8.4, meaning that smaller regions are considered. Intuitively, as more potential parameter values are excluded by reducing the region size, the actual choice of the parameter value has less impact on reachability

probabilities. The smaller the region gets, the smaller the over-approximation: The optimal instantiation on the pMC $\mathcal{D}$ is over-approximated by some strategy on $\mathsf{sub}_R(\mathcal{D})$. The approximation error originates from choices where an optimal strategy on $\mathsf{sub}_R(\mathcal{D})$ chooses actions val and val' at states s_1 and s_2, respectively, with $\mathsf{val}(x_i^{s_1}) \neq \mathsf{val}'(x_i^{s_2})$ for some parameter x_i, and therefore intuitively disagree on its value. The probability mass that is affected by these choices decreases the smaller the region is. For infinitesimally small regions, the error from the over-approximation vanishes, as the actions for the upper and the lower bound of a parameter become equal up to an infinitesimal.

Example 12.8 We continue with Example 12.7. As we could not determine whether R is accepting, rejecting or inconsistent, we split R along the y-axis into $R' := [1/10, 4/5] \times [2/5, 3/5]$ and $R'' := [1/10, 4/5] \times [3/5, 7/10]$. For R', we obtain an upper bound of $69/95$, while for R'' we obtain $94/145$ as an upper bound. Both bounds are below $3/4$, and thus R' and R'' are both accepting, and thus R is shown to be accepting.

12.2 Parameter lifting for MDPs

In the last section, we approximated reachability probabilities in (locally-monotone) pMCs by considering its substitution MDP. The nondeterminism in the substitution MDP encodes the parameter valuations over-approximating the reachability probabilities in the pMC. By letting a demonic strategy resolve the nondeterminism in the MDP, we obtain sound bounds on the reachability probabilities in the pMC. These bounds are efficiently computed by standard MDP model checking.

In this section, we generalise the approach to pMDPs, which already contain nondeterminism. The result naturally leads to a 2-player *stochastic game* (SG)—see also Section 3.4. In the SG obtained after relaxation and lifting of a pMDP, one player controls the nondeterminism inherent to the MDP, while the other player controls the (abstracted) parameter values. Letting the two players adequately minimise and/or maximise the reachability probabilities in the SG yields sound bounds on the minimal (and maximal) reachability probabilities in the pMDP. For example, if the player for the original nondeterminism maximises and the parameter player minimises, we obtain a lower bound on the maximal probability. These bounds may be computed by standard stochastic game model checking.

In our presentation below, we discuss the interplay of the two sources of nondeterminism. We formalise the construction of the substitution with nondeterminism, analogous to the pMCs from the previous section. We obtain Definition 12.4 as analogue to Definition 12.3, and Theorem 12.8 as analogue to Theorem 12.7. We do not repeat the concept of relaxation, as we rely on a reduction to the substitution of pMCs.

12.2.1 Idea

In the following, let $\mathcal{M} = (S, \iota, Act, X, \mathcal{P})$ be a locally monotone[1] pMDP and R a graph-preserving, rectangular, closed region.

Angelic nondeterminism | We first turn our attention to the angelic region verification. We recall:

$$R \subseteq \mathsf{acc}^{\mathcal{M}}_{\exists,\varphi} \quad \text{iff} \quad \text{for all } M \in \langle \mathcal{M} \mid R \rangle \text{ exists } \sigma \in \Sigma^{\mathcal{M}} \text{ s.t. } M[\sigma] \models \varphi.$$

The formal idea is again to apply an over-approximation by relaxation, and utilise nondeterminism over the boundaries of the parameter region to obtain a parameter-free model. However, for the proof, we may use another idea that allows to reuse ideas from the pMC case. Therefore, we want to consider all induced pMCs. However, the first quantifier is over the parameters, and we cannot simply reorder the quantifiers. The following implication holds and helps:

$$\text{exists } \sigma \in \Sigma^{\mathcal{M}} \text{ s.t. for all } M \in \langle \mathcal{M} \mid R \rangle \text{ s.t. } M[\sigma] \models \varphi \quad \text{implies} \quad R \subseteq \mathsf{acc}^{\mathcal{M}}_{\exists,\varphi}.$$

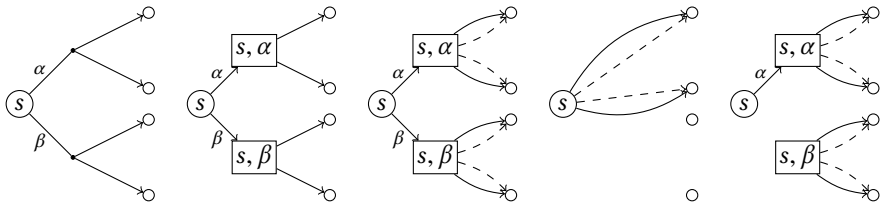
We concisely over-approximate using an SG, which represents the substitution for every induced pMC.

Demonic nondeterminism | For the demonic relation, pMDP region verification may be expressed as a pMC region verification problem (Proposition 5.53) and may be approximated by MDP model checking as shown in the previous section. However, in this chapter we are only concerned with ε-preserving regions, which do not suffice for this reduction. Nevertheless, the demonic case is simpler as there is no quantifier alternation. Indeed, the stochastic game that over-approximates the pMDP then collapses to an MDP that is similar to the MDP from the previous section. We therefore omit further treatment of demonic nondeterminism in this section.

12.2.2 Lifting

The parameter substitution of a pMDP (cf. Figure 12.3a) yields an SG—as in Figure 12.3c. To ensure that each state in the SG can be assigned to a unique player, we first require, w.l.o.g., that the pMDP is alternating, as in Definition 5.9 (cf. Figure 12.3b). The subsequent substitution on the probabilistic states yields the SG $\mathsf{sub}_R(\mathcal{M})$, where one player represents the nondeterminism of the original pMDP $\mathcal{M}$, while the other player decides whether parameters should be set to their lower or upper bound in the region R. More precisely, the substitution SG aggregates the substitution MDPs of all induced

[1] locally monotone pMDPs are a straightforward adaption of locally monotone pMCs.



(a) pMDP $\mathcal{M}$ (b) $\mathsf{alt}(\mathcal{M})$ (c) $\mathsf{sub}_R(\mathcal{M})$ (d) $\mathsf{sub}_R(\mathcal{M}[\sigma])$ (e) $\mathsf{sub}_R(\mathcal{M})[\sigma]$

Figure 12.3: Illustration of the substitution of a pMDP

pMCs, e.g., the substitution of the induced pMC in Figure 12.3d is represented by the induced MDP in Figure 12.3e.

substitution SG

Definition 12.4 (Substitution (pMDPs)) For an alternating pMDP $\mathcal{M} = (S, \iota, Act, X, \mathcal{P})$ and region R, the *substitution SG of $\mathcal{M}$ and R* is the SG

$$\mathsf{sub}_R(\mathcal{M}) \coloneqq (S, \iota, Act_{\mathsf{sub}}, P_{\mathsf{sub}})$$

with $S_{\mathrm{I}} \coloneqq \{s \mid \mathbb{A}\mathrm{ct}(s) > 1\}$, $S_{\mathrm{II}} \coloneqq S \setminus S_{\mathrm{I}}$, $Act_{\mathsf{sub}} \coloneqq Act \uplus \left(\biguplus_{s \in S_{\mathrm{II}}} Act_s \right)$ where

$$Act_s \coloneqq \{u \colon \mathsf{occurs}(s) \to \mathbb{R} \mid u(x) \in \mathcal{B}(x) \text{ for all } x \in \mathsf{occurs}(s)\},$$

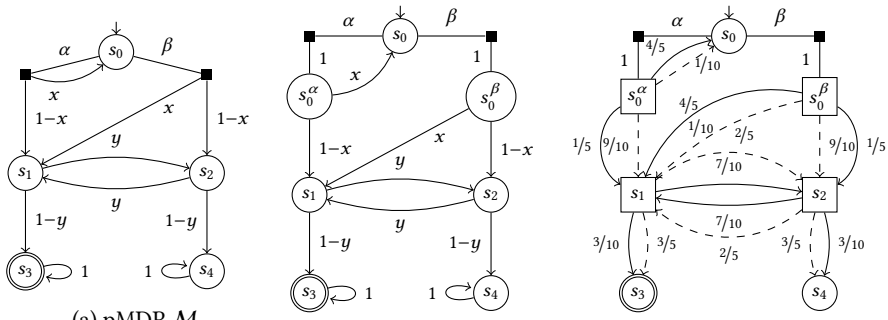
and

$$P_{\mathsf{sub}}(s, \alpha, s') \coloneqq \begin{cases} 1 & \text{if } s \in S_{\mathrm{I}}, \alpha \in \mathbb{A}\mathrm{ct}(s), s' \in S, \\ \mathcal{P}(s, \beta, s')[\alpha] & \text{if } s \in S_{\mathrm{II}}, \alpha \in Act_s, \mathbb{A}\mathrm{ct}(s) = \{\beta\}, s' \in S_{\mathrm{I}}, \\ 0 & \text{otherwise.} \end{cases}$$

We relate the SG $\mathsf{sub}_R(\mathcal{M})$ under different strategies for player I with the substitution in the strategy-induced pMCs of $\mathcal{M}$. We observe that the strategies for player I in $\mathsf{sub}_R(\mathcal{M})$ coincide with strategies in $\mathcal{M}$. Consider the induced MDP $\mathsf{sub}_R(\mathcal{M})[\sigma]$ with a strategy σ for player I. The MDP $\mathsf{sub}_R(\mathcal{M})[\sigma]$ is obtained from $\mathsf{sub}_R(\mathcal{M})$ by erasing transitions not agreeing with σ. In $\mathsf{sub}_R(\mathcal{M})[\sigma]$, player I-state have a single available action, while player II-states have multiple available actions.

Example 12.9 We consider the pMDP in Figure 12.4a. We first construct an alternating pMDP in Figure 12.4b by introducing two auxiliary states. Let region $R = [1/10, 4/5] \times [2/5, 7/10]$ be as before. The corresponding substitution SG is depicted in Figure 12.4c. States that belong to player II, i.e., that correspond to selecting parameter values, are depicted as rectangles.

The remainder of this section is devoted to the correctness as formalised below.



(a) pMDP $\mathcal{M}$ (b) pMDP alt($\mathcal{M}$) (c) SG $\mathrm{sub}_R(\mathcal{M})$

Figure 12.4: Lifting pMDPs

Theorem 12.8 Let $\mathcal{M}$ be a pMDP, R a region, and φ a reachability property. Then:

$$\mathrm{sub}_R(\mathcal{M}) \,{}^{\forall,\forall}\!\models \varphi \text{ implies } R \subseteq \mathsf{acc}^{\mathcal{M}}_{\forall,\varphi}, \text{ and}$$
$$\mathrm{sub}_R(\mathcal{M}) \,{}^{\exists,\forall}\!\models \varphi \text{ implies } R \subseteq \mathsf{acc}^{\mathcal{M}}_{\exists,\varphi}.$$

We focus on the second statement and assume $\varphi = \mathbb{P}_{>\lambda}(\Diamond T)$. Before we summarise the proof, we outline the main steps. We have that $R \subseteq \mathsf{acc}^{\mathcal{M}}_{\exists,\varphi}$ iff for all $\mathsf{val} \in R$ there is a strategy $\sigma \in \Sigma^{\mathcal{M}}$ for which the reachability probability in the MC $\mathcal{M}[\sigma, \mathsf{val}]$ exceeds the threshold λ, i.e.,

$$\begin{aligned} R \subseteq \mathsf{acc}^{\mathcal{M}}_{\exists,\varphi} \quad & \text{iff for all } M \in \langle \mathcal{M} \mid R \rangle \text{ exists } \sigma \in \Sigma^{\mathcal{M}} \text{ s.t. } M[\sigma] \models \varphi \\ & \text{iff } \min_{\mathsf{val} \in R} \max_{\sigma \in \Sigma^{\mathcal{M}}} Pr^{\sigma}_{\mathcal{M}[\mathsf{val}]}(\Diamond T) > \lambda. \end{aligned}$$

To reduce the correctness proof to the pMC case, we utilise the inequality

$$\min_{\mathsf{val} \in R} \max_{\sigma \in \Sigma^{\mathcal{M}}} Pr^{\sigma}_{\mathcal{M}[\mathsf{val}]}(\Diamond T) \geq \max_{\sigma \in \Sigma^{\mathcal{M}}} \min_{\mathsf{val} \in R} Pr^{\sigma}_{\mathcal{M}[\mathsf{val}]}(\Diamond T).$$

Remark 12.3 The right-hand side describes a robust strategy, cf. Section 5.5.

The right-hand side is intuitively a maximum over all induced pMCs: for each pMC, we may use the inequality from Corollary 12.6 and obtain:

$$\min_{\mathsf{val} \in R} \max_{\sigma \in \Sigma^{\mathcal{M}}} Pr^{\sigma}_{\mathcal{M}[\mathsf{val}]}(\Diamond T) \geq \max_{\sigma \in \Sigma^{\mathcal{M}}} \min_{\mathsf{val} \in R} Pr^{\sigma}_{\mathcal{M}[\mathsf{val}]}(\Diamond T) \geq \max_{\sigma \in \Sigma^{\mathcal{M}}} \min_{\tau} Pr^{\tau}_{\mathrm{sub}_R(\mathcal{M}[\sigma])}(\Diamond T).$$

We show that we may compute the right-hand side with the substitution of the pMDP. The corollary formalises that $\mathsf{sub}_R(\mathcal{M})[\sigma]$ (Figure 12.3d) and $\mathsf{sub}_R(\mathcal{M}[\sigma])$ (Figure 12.3e) induce the same reachability probabilities.

Corollary 12.9 For an alternating pMDP $\mathcal{M}$, graph-preserving region R, target states $T \subseteq S$, and strategies $\sigma \in \Sigma_{\mathrm{I}}^{\mathsf{sub}_R(\mathcal{M})}$ and $\tau \in \Sigma^{\mathsf{sub}_R(\mathcal{M}[\sigma])}$, it holds that

$$Pr^{\tau}_{\mathsf{sub}_R(\mathcal{M}[\sigma])}(\Diamond T) = Pr^{\sigma,\widehat{\tau}}_{\mathsf{sub}_R(\mathcal{M})}(\Diamond T)$$

with $\widehat{\tau} \in \Sigma_{\mathrm{II}}^{\mathsf{sub}_R(\mathcal{M})}$ satisfies $\widehat{\tau}(s) = \tau(s)$ for all $s \in S_{\mathrm{II}}$.

Instead of performing the substitution on the pMC induced by $\mathcal{M}$ and σ, we perform the substitution on $\mathcal{M}$ directly and preserve the reachability probability. Consequently, and analogously to the pMC case (cf. Theorem 12.7), we can derive whether R accepts φ by analysing a stochastic game (cf. Section 3.4).

Proof (Theorem 12.8). We only prove the second statement assuming $\varphi = \mathbb{P}_{>\lambda}(\Diamond T)$, other reachability properties are similar. A proof for the (simpler) first statement may be derived in an analogous manner. We have

$$\begin{aligned} R \subseteq \mathsf{acc}^{\mathcal{M}}_{\exists,\varphi} \quad &\text{iff} \quad \text{for all } M \in \langle \mathcal{M} \mid R \rangle \text{ exists } \sigma \in \Sigma^{\mathcal{M}} \text{ s.t. } M[\sigma] \models \varphi \\ &\text{iff} \quad \min_{\mathsf{val} \in R} \max_{\sigma \in \Sigma^{\mathcal{M}}} Pr^{\sigma}_{\mathcal{M}[\mathsf{val}]}(\Diamond T) > \lambda. \end{aligned}$$

A lower bound for this probability is obtained as follows:

$$\begin{aligned} &\min_{\mathsf{val} \in R} \max_{\sigma \in \Sigma^{\mathcal{M}}} Pr^{\sigma}_{\mathcal{M}[\mathsf{val}]}(\Diamond T) \geq \max_{\sigma \in \Sigma^{\mathcal{M}}} \min_{\mathsf{val} \in R} Pr^{\sigma}_{\mathcal{M}[\mathsf{val}]}(\Diamond T) \\ &\overset{*}{\geq} \max_{\sigma \in \Sigma^{\mathcal{M}}} \min_{\tau} Pr^{\tau}_{\mathsf{sub}_R(\mathcal{M}[\sigma])}(\Diamond T) \overset{**}{=} \max_{\sigma \in \Sigma_{\mathrm{I}}^{\mathsf{sub}_R(\mathcal{M})}} \min_{\tau \in \Sigma_{\mathrm{II}}^{\mathsf{sub}_R(\mathcal{M})}} \left(Pr^{\sigma,\tau}_{\mathsf{sub}_R(\mathcal{M})}(\Diamond T) \right). \end{aligned}$$

The inequality $*$ is due to Corollary 12.6. The equality $**$ holds by Corollary 12.9. Together,

$$\begin{aligned} \max_{\sigma \in \Sigma_{\mathrm{I}}^{\mathsf{sub}_R(\mathcal{M})}} \min_{\tau \in \Sigma_{\mathrm{II}}^{\mathsf{sub}_R(\mathcal{M})}} \left(Pr^{\sigma,\tau}_{\mathsf{sub}_R(\mathcal{M})}(\Diamond T) \right) \quad &\text{implies} \quad \min_{\mathsf{val} \in R} \max_{\sigma \in \Sigma^{\mathcal{M}}} Pr^{\sigma}_{\mathcal{M}[\mathsf{val}]}(\Diamond T) > \lambda \\ &\text{iff} \quad R \subseteq \mathsf{acc}^{\mathcal{M}}_{\exists,\varphi}, \end{aligned}$$

and finally:

$$\begin{aligned}
& \max_{\sigma \in \Sigma_{\mathrm{I}}^{\mathrm{sub}_R(\mathcal{M})}} \left(\min_{\tau \in \Sigma_{\mathrm{II}}^{\mathrm{sub}_R(\mathcal{M})}} \left(Pr^{\sigma,\tau}_{\mathrm{sub}_R(\mathcal{M})}(\Diamond T) \right) \right) > \lambda \\
\text{iff} \quad & \exists \sigma \in \Sigma_{\mathrm{I}}^{\mathrm{sub}_R(\mathcal{M})}. \ \forall \tau \in \Sigma_{\mathrm{II}}^{\mathrm{sub}_R(\mathcal{M})} \mathrm{sub}_R(\mathcal{M})[\sigma,\tau] \models \varphi \\
\text{iff} \quad & \mathrm{sub}_R(\mathcal{M}) \, {}^{\exists,\forall}\!\models \varphi.
\end{aligned}$$

Refinement | As before, the sources of over-approximation vanish with decreasing region size.

12.3 Empirical Evaluation

12.3.1 Set-up

Implementation | We evaluate parameter lifting as implemented in STORM. A template for the lifted model is built once, and then instantiated to reduce the building times for checking a single region. Likewise, the model checker uses previous model checking results as a hint to speed up model checking.

Research questions | We consider region verification using parameter lifting. In particular, we consider the verification of accepting or rejecting regions on challenging models, i.e., on models in which the parameter dependencies are relevant. If the dependencies are not relevant, the relaxed model accepts or rejects exactly the accepting/rejecting regions of the original model. Consequently, parameter lifting on accepting or rejecting regions requires a single model checking call, and applying parameter lifting is obviously a good idea. Here, we are more interested in scenarios in which the over-approximation is too coarse. The essential question is then: *How many subregions have to be analysed to show that a region is indeed accepting or rejecting?* and *What is the corresponding run time?*. For regions where we get unknown as result, we use trivial and uninformed splitting in all dimensions to avoid over-engineering. We focus on parameter lifting, and refer to Section 8.4 and in [5] for empirical results showing that parameter lifting is state-of-the-art for region verification.

Environment | All experiments were executed on a Macbook MF839LL/A with 8GB RAM memory limit and a 2h time out. STORM used one thread for all computations.

Benchmark selection | The dependencies between parameters are not relevant for region verification, in, e.g., the common models *BRP*, *Zeroconf* and *consensus*. Such

Table 12.1: Region verification with parameter lifting (pMCs)

Model	Instance	States	Trans	$\lvert X \rvert$	Build t	λ	Reg Checked	% Final	MC t
Crowds	10,5	104512	246082	2	1	$1-(3.000 \cdot 10^{-4})$	13	76.9	1
						$1-(3.150 \cdot 10^{-4})$	21	76.1	2
						$1-(3.190 \cdot 10^{-4})$	37	75.7	3
						$1-(3.195 \cdot 10^{-4})$	— inconsistent —		
Nand	10,3,Rej	21252	31927	2	<1	0.3000	29	75.9	<1
						0.2850	117	75.2	<1
						0.2830	369	75.1	2
						0.2824	4245	75.0	12
						0.2822	— inconsistent —		
Nand	10,3,Acc	21252	31927	2	<1	$2.0 \cdot 10^{-4}$	169	75.1	1
						$4.0 \cdot 10^{-4}$	1281	75.0	6
						$4.5 \cdot 10^{-4}$	6457	75.0	30
						$4.7 \cdot 10^{-4}$	18373	75.0	81
						$4.9 \cdot 10^{-4}$	— inconsistent —		
Repud	K2,std,F2	16	32	14	<1	0.921	16385	100	2
						0.920	540673	100	84
						0.900	1589249	100	212
						0.850	7880705	100	1136
						0.800	23609345	100	4090
						0.490	— inconsistent —		
Maze	m1,sl,rew	16	50	9	<1	10	16897	99.8	6
						12	144385	99.8	53
						13	569345	99.8	209
						14	3027969	99.8	1370
						20	— inconsistent —		

models are easy to analyse and omitted here. We consider simple pMCs and pMDPs, having obvious rectangular regions. We do not include models with more than 15 parameters: In a model with 25 parameters, even splitting a region once in every dimension immediately yields 2^{25} regions. Even when 1000 regions are checked per second, and those regions yield a conclusive answer, region verification still takes over 8 hours. In a model with 15 parameters, splitting every dimension once, and then splitting one percent of the regions a second time requires roughly $0.01 \cdot \left(2^{15}\right)^{15} \approx 2^{120}$ regions to be checked. Splitting heuristics that split only in some dimensions (used in Section 7.1) may perform better, but are beyond the scope of this evaluation.

Concretely, we use parametric *Crowds* from [HHZ11b], *Nand* as in Chapter 8 and *Repud* and *Maze* as in Section 7.1 as pMCs, and as pMDPs, we use a *janitor* model from [5] and the *coinX* consensus model from [11]. We use regions such that for each parameter x, the interval $0.1 \leq \mathcal{I}(x) \leq 0.9$.

12.3.2 Evaluation

Results | Table 12.1 shows some results for pMCs. The analogue for pMDPs is listed in Table 12.2. We list the benchmark and an instance string that refers to the particular instance of the model. The number of states and transitions (*Trans*) indicate the size

Table 12.2: Region verification with parameter lifting (pMDPs)

Model	Instance	States	Act	Trans	$\|X\|$	Build t	λ	Reg Checked	% Final	MC t
Janitor	nocrashtoT	12822	36171	120752	2	2	0.7000	5	80	<1
							0.9000	81	75.3	<1
							0.9120	993	75.0	2
							0.9126	3085	75.0	6
							0.9127	— inconsistent —		
CoinX	proc2,K8,rew	4160	6208	7728	2	< 1	900	37	75.7	50
							800	53	75.5	57
							780	89	75.3	61
							770	205	75.1	78
							769	273	75.1	86
							768	— inconsistent —		
CoinX	proc4,K2,rew	362496	968704	1203712	4	7	280	33	93.9	1377
							250	65	93.8	2532
							230	97	93.8	2917
							210	145	93.8	4036
								— inconsistent —		

of the MC, and for MDP we additionally list the number of actions. The column $|X|$ indicates the number of parameters in the model. The next column (*Build t*) gives the time in seconds that STORM takes to build the model. For each benchmark, we consider various thresholds λ in the property. For each threshold, we show the number of regions that we checked (*Reg Checked*), and the percentage of regions that we successfully classified as accepting/rejecting before covering the full parameter space (*% Final*). The last column gives the run time in seconds for the necessary checks (*MC t*). For each model, we also give a threshold such that the region becomes inconsistent (and where region verification is thus not applicable).

Discussion | The run time roughly scales with the number of regions that are checked, plus an initial overhead from the preprocessing in the first model checking call. The latter time may be substantial, e.g., in the pMDP *CoinX* (with two parameters). Furthermore, the high percentage of final regions shows that typically only few regions need to be refined. Consider the models with two parameters, an inconclusive region is split into four regions, and roughly three out of four are then final, and the consistency in these numbers indicates that many accepting regions are very regular. In regular probabilistic model checking the size of the model description (i.e., the PRISM model) and the number of states of the model are good indicators for the required time, as model building is often a bottleneck. In parameter lifting, model building is performed only once, and thus the structure of the model is essential for the time to check one region, as the rebuilding effort is minimised. A good example is *CoinX* vs. *Janitor*. We see that for models with more parameters, the number of regions we check quickly explodes. Finally, e.g., in *Crowds*, the influence of the parameter dependencies is so

small that even if the threshold is tight, few regions suffice, whereas in, e.g., *Repud*, the influence is significant.

Furthermore, we remark that the run times for parameter space partitioning is often worse: after all, there are now also inconsistent regions which after analysis need to be split. As a consequence, the number of regions increases drastically, and thus the run time. This sharp increase may be alleviated somewhat by the techniques in Section 8.4.

Parameter lifting is an abstraction technique that removes the parameter dependencies and evaluates the resulting model by considering a corresponding MDP (for pMCs) or SG (for pMDPs). The abstraction allows for a straightforward embedding in a divide-and-conquer refinement loop, such as the loop of Section 8.4. It is an excellent approach for region verification whenever the influence of the parameter dependencies is minor, or if the number of parameters is low. For more parameters, the refinement in exponentially many regions often yields a prohibitive number of subregions that have to be analysed.

Part IV

Parameter Synthesis Today and in the Future

13 Tool Support

Summary | We give a brief account of available tool support for parameter synthesis before presenting the ecosystem for parameter synthesis that we[1] have built. All tools and libraries are open-source and available via Github. This overview presents the most relevant pieces of the ecosystem, and a selection of crucial aspects, some of which were previously mentioned in [17]. For empirical evaluations, we refer to the chapters describing the implemented methods.

Related tools | We list a selection of (mature) tools that support parameter synthesis as described in this thesis. Tool support for parameter synthesis as outlined in this thesis started with PARAM [HHZ11b]: and eventually contained the computation of solution functions, (sampling based, unsound) parameter space partitioning, sampling. Most of these functionalities were later integrated into PRISM [KNP11] and to EPMC [HLST+14]. PARAM is no longer actively maintained. The tool COMICS [JÁVW+12] supports the computation of solution functions and of high-level counterexamples, but is no longer maintained. The tools PRISM-PSY [CPPB+16] and Rodes [CCGK+17] have implemented some related ideas for continuous time. There are some tools for the quantitative evaluation of software product lines, e.g., QFLan [VBLL18] and ProFeat [CDKB18], in particular support the enumerative analysis of Markov models, but take their domain specific input languages.

storm | storm [9][2] is a state-of-the-art probabilistic model checker, with support for a variety of input languages and a wide range of advanced operations, including the computation of high-level counterexamples as introduced in [DJWÁ+14] and extended based on Chapter 10, and the translation from POMDPs to pMCs from Section 7.1. storm is written in c++ as a modular library with a set of command-line interfaces on top. Performance and relying on high-performance external libraries are key design principles, an excellent overview of its design has been given in [Hen18].

[1] In this sentence, 'we' means—all in alphabetical order—Harold Bruintjes, Florian Corzilius, Christian Hensel, Tim Quatmann, Matthias Volk and myself, supported and supervised by Erika Ábrahám and Joost-Pieter Katoen, with valuable contributions from Philipp Berger, Murat Cubuktepe, Tom van Dijk, Tom Janson, Gereon Kremer, Joachim Klein, and Lutz Klinkenberg, and with due acknowledgements to our co-authors and anyone who helped shaping the theory, or constructed third-party libraries.

[2] `http://www.stormchecker.org`

We restrict our treatment to the parametric part. As the development for STORM was initiated around the wish to extend MRMC[KZHH+11] with support for interval Markov models, STORM was well-prepared to an extension with other data types as probabilities. Possibly the biggest change has been to integrate model-checking-in-the-loop, which deviated from the early design decision that presenting the model checking result is the last operation before terminating the model checker.

In STORM, parametric models may be represented explicitly using matrices (in different formats), as well as based on a symbolic, decision diagram (dd)-based data structure built on top of SYLVAN [DP17]. All representations support the computation of qualitative properties and bisimulation minimisation as described in [HHZ11b]. STORM supports the various versions of the solution function computation from Section 8.1 and parameter lifting from Chapter 12 on explicitly represented models.

For the computation of the solution function, STORM supports standard Gaussian elimination using efficient routines in EIGEN[1], fraction-free Gaussian elimination on the internal data structures, and (set-based) transition elimination on explicit (symbolic) models. Furthermore, STORM supports efficient sampling and parameter lifting using so-called parameter replacement: it creates a template of the concrete model once and then replaces the parameter values upon request. For parameter lifting, this replacement is done on the substitution MDP. During replacement, it is beneficial to avoid reevaluating the transition probabilities that occur multiple times in the model. Thus, we evaluate each occurring function once and replace the outcome directly at all occurrences. To increase the efficiency of the model checker beyond using it as a black-box, it is beneficial to give the model checker a hint on how the solution may look like. We often use the result from the last model checking call as a hint for the next call. Finally, for a growing number of model checking calls, any one-time preprocessing of the model eventually pays off. In particular, we apply minimisation techniques before construction of the template for substitution. We use both bisimulation minimisation and state-elimination of parameter-free transitions.

CARL | To represent polynomials and rational functions, and crucially, have efficient operations on them, STORM integrates the c++ library for computer arithmetic and logic, abbreviated CARL[2]. CARL was designed as library underneath the ETR support in SMT-RAT [2] and interfaces with low-level libraries for exact numbers such as GMP[3] or CLN[4], and may interface with the algebra libraries GINAC [BFK02] and CoCoA [AB] to, e.g., compute GCDs. In particular, we use a partially factorised representation of rational functions, as introduced by [JCVW+14]: this representation is beneficial for the computation of solution functions on cyclic models.

[1] http://eigen.tuxfamily.org/
[2] http://smtrat.github.io/carl/
[3] https://gmplib.org/
[4] https://www.ginac.de/CLN/

STORMPY | Using an API avoids using the static command-line interface of STORM, it allows to the incremental usage of parameter lifting and sampling, and is essential for the CEGIS and abstraction-refinement loops on sketches. Furthermore, (tool support for) parameter synthesis is yet to get mature. Therefore, support for rapid prototyping is essential, in particular, a need to visualise, heavily interact, and be flexible with the code-base. These requirements are arguably easier to satisfy in the programming language python—and the various libraries available in python—than in c++. This idea has sparked the development of a python API for STORM, which we called STORMPY[1], and a python interface for CARL, called PYCARL[2].

PROPHESY | PROPHESY[3] is a playground for parameter synthesis. Its design aims to provide more flexibility than the performance-optimised routines in probabilistic model checkers. Foremost, PROPHESY is a python library that integrates sampling, parameter space partitioning, SMT-based region verification, utilising convex optimisation for feasibility synthesis, and some dedicated data structures. For the latter, it relies on PYCARL. PROPHESY contains various adapters to interface with (python APIs or command-line interfaces of) probabilistic model checkers, SMT solvers, particle swarm optimisers, and convex optimisation solvers. Together, these interfaces allow integrating the latest advances from these heavily researched topics into parameter synthesis and to compare various approaches and tools via a uniform interface.

The playground is extended by two features on top of the library: a highly configurable command-line interface and a prototype web service. The command-line has a pipeline architecture: we may chain various operations such as simplification, sampling, (partial) parameter space partitioning, etc.—in various orders—each with their own options. These features allow for easy integration of new steps or heuristics in an existing chain of methods. The web service exemplifies that such operations may be flexibly connected with other tools or with user input, e.g., via a web page. The playground currently comes with some overhead, that we aim to reduce in the future.

DYNASTY | DYNASTY[4] targets parameter synthesis for large families and sketches has been used for the empirical evaluation of Chapters 10 and 11 and constitutes of tools which share input formats and some routines. In contrast to PROPHESY, DYNASTY is tightly bound to the internal data structures of STORMPY and the SMT-solver Z3 [MB08]. This design decision makes it an efficient prototype and its python codebase lightweight. To ease development, DYNASTY comes with a variety of analysis tools that allow visualising the performance of the analysis.

[1] https://moves-rwth.github.io/stormpy/
[2] https://moves-rwth.github.io/pycarl/
[3] https://moves-rwth.github.io/prophesy/
[4] https://github.com/moves-rwth/dynasty/

Dynasty takes a project as input. A project contains a sketch – a PRISM-program or a Jani model with undefined (integer) constants as holes, a mapping from holes to options, a list of properties, and constraints on the realisations. Dynasty supports (optimal) feasibility analysis and parameter space partitioning.

Internally, PRISM-programs are converted into Jani models. For the abstraction-refinement, Dynasty creates a Jani model of the abstracted MDP and then builds an annotated MDP using standard routines—the annotations help to restrict the MDP in any refinement step. For inductive synthesis, Dynasty encodes all realisations into Z3 and iterates over the solutions. For each realisation, it passes the realisation and then builds and analyses the MC with respect to the available properties. For properties that are rejected, we let stormpy perform counterexample analysis. Dynasty may also employ an SMT-encoding of the feasibility problem, or construct a Jani model describing the all-in-one MDP. Including a more convenient user interface is work in progress, as is the integration of the various methods.

14 Conclusion and Future Work

Conclusion | Parameter synthesis (of probabilities) in Markov models is a fundamental problem in the verification and synthesis of Markov chains (MCs) and Markov decision processes (MDPs). Basically, parameter synthesis asks for which probabilities (parameter values) an MC or MDP with parameters (pMC, pMDP) satisfies a given property. From a verification point of view, parameter synthesis allows to encode natural uncertainty about the probabilities in the model and thereby gives verification results that are robust to the uncertainty in the model. They are closely related to interval Markov models but feature an analysis which is significantly less pessimistic as parametric models feature dependencies between different states. From a (classical) synthesis point of view, parametric models allow optimising the performance of systems with controllable parameters, e.g., in distributed systems that randomise, or to synthesise the required quality of components in the reliability assessment of hardware systems. Finally, and shown in this thesis, parameter synthesis has tight connections to, e.g., strategy synthesis in POMDPs, and to the quantitative analysis of software product lines. Such applications may be captured in a sketching framework, where a system designer sketches a model and leaves specifics open. Thus, parameter synthesis is a step towards sketching of, e.g., controllers in a stochastic domain.

While parameter synthesis for Markov models has been around for a while, many fundamental questions remained open. Parameter synthesis may be expressed in the existential theory of the reals (ETR) and we established that feasibility synthesis ("does there exist parameter values such that a property is satisfied?") is ETR-complete. Intuitively, it is as hard as finding roots of a multivariate polynomial. For pMCs, the complexity is exponential in the number of parameters, but polynomial in the size of the model. These and further fundamental results about, e.g., continuity help to develop suitable alternatives to an encoding in ETR.

We reimplemented, studied and improved basic methods for various forms of parameter synthesis. Already analysing models with one parameter is significantly more involved than analysing their parameter-free counterpart, and some applications require synthesis in models with many parameters. These observations motivate the use of heuristic methods: We introduced new methods: for verification via an abstraction to parameter-free models and for synthesis via convex optimisation problems.

We additionally studied a restriction of parameter synthesis using family MCs, which encode finite sets of MCs sharing some common structure, but not their complete topology. For family MCs, synthesis may be done using enumeration and we improved this

scheme with standard ideas from formal methods: inductive synthesis and abstraction-refinement. These ideas yield significant improvements over the state-of-the-art.

All aforementioned methods are implemented in open-source frameworks based on the probabilistic model checker STORM and are ready to be applied and improved upon in future work. Below, we discuss how ideas that might help to overcome known shortcomings of the state-of-the-art.

Future in parameter synthesis | The most fundamental open question is the complexity of feasibility in pMDPs with a fixed number of parameters. Further relevant questions include the analysis of pMCs with a particular structure of the parameters. Fundamental results in these areas may spark new and efficient methods.

Another avenue for improvements is a tighter integration, first with SMT solvers for ETR. Complexity-wise, SMT solvers for ETR are an adequate technique, yet the performance of these solvers for parameter synthesis is mostly mediocre. We think that tighter integration with methods in the SMT solver would be beneficial: even though parameter synthesis is ETR-complete, most case studies have promising structural features, and internally, the SMT solver holds much more information than available to, e.g., the parameter space partitioning algorithms built on top of those solvers. Similarly, we think that various heuristic sampling approaches, ranging from hill-climbing to more evolved algorithms, might benefit from tighter integration with model checking—something that we already witnessed during the integration with convex optimisation. Finally, inductive synthesis may be lifted from the discrete case to the general parameter synthesis setting.

Parameter synthesis, particularly sampling, is relying on the analysis of parameter-free models. Some of these models may be analysed quickly as the difference between the threshold in the property and the reachability probability in the model is large—such models are often considered uninteresting, but occur naturally in this setting. Parameter synthesis profits if these models can be analysed more efficiently. More taxing for the analysis are stiff models, which contain very small probabilities, as numerical methods converge slowly on them. Model checking stiff models has received less attention in the literature. It is interesting to consider parametric models to analyse such models, and to improve the analysis of stiff models for the benefit of analysing parametric models. Finally, the incrementality of SAT-based methods is an interesting motivation to study SAT-based methods for model checking in a parameter-synthesis loop.

Besides new methods for reachability in parametric Markov models, it would be worthwhile to extend the existing methods for more expressive properties. Whereas reachability as a property is key for the support of various other properties, relying on algorithms for reachability to analyse other properties often requires a blow-up of the model. To avoid this blow-up, it would be practically relevant to lift and support (reward)bounded and ω-regular objectives directly. Similarly, to avoid investigating

large models, (aggressive) pruning methods with abstraction and static analyses may be worthwhile: such techniques are often not used in probabilistic model checking as they are more expensive than the analysis itself, but in the light of parameter synthesis, such methods may be worthwhile. Finally, specialised problems, originating from various application areas may be improved with domain-specific knowledge. We sketch some ideas below.

Future in applications | The verification of parametric models may be used as a way to verify models that are underspecified. Such problems occur naturally, e.g., in product lines, where different components may be used to realise some function. These components have some semantics that should be exploited by the verification procedure, and thus, parameter synthesis may benefit from and be beneficial to component-based analyses of systems. In the same spirit, the connection to methods that use the iterative verification of finite models to support infinite-state systems (or parameter synthesis therein) seems an interesting area for future work. Another area where parameter synthesis is gaining traction is the verification of models that are extracted from data (measurements) or learning (simulations), which are inherently uncertain. There, the uncertainty may often be quantified, and more research in models that allow attributing parameters with likelihoods is interesting. Likewise, a closer connection to (runtime) monitoring and system identification approaches seems worthwhile.

Parameter synthesis yields methods that are orthogonal to the state-of-the-art in solving POMDPs or concurrent stochastic reachability games. The NP-hardness of those problems is just one witness that having a portfolio of orthogonal methods is beneficial, and combining existing methods with parameter synthesis is an obvious area for future work, just as exploiting the structural properties of those domains in parameter synthesis methods. This direction may be generalised into sketching of probabilistic programs with temporal logic constraints, a research direction that generalises many exciting areas in the verification and synthesis of probabilistic models. For sketching, constructs and methods that allow the designer to concisely feed all relevant information to the synthesiser seem essential, and this question may be tackled first in the light of parameter synthesis.

Part V

Appendices

A Notation

Sets of numbers | We use $\mathbb{R}, \mathbb{Q}, \mathbb{Z}, \mathbb{N}$ to denote the real, rational, integer and natural numbers (with 0 as natural number), respectively. We use subscripts, e.g., $\mathbb{R}_{\geq 0}, \mathbb{R}_{>0}, \mathbb{R}_{<0}$ to restricts these sets to respectively non-negative, positive or negative numbers. We use a superscript to include infinity, e.g., $\mathbb{R}^{\infty}_{\geq 0}$ denotes the non-negative real numbers with infinity.

Functions | Functions relate inputs to unique outputs, i.e., a function f maps from a set V to a set W, denoted $f\colon V \to W$. A function that does not map every input to an output is a partial function, denoted $V \nrightarrow W$. We often denote functions with a set notation: For example, we might denote the function $f\colon \{1,2,3\} \to \{1,2,3\}$ with $f(i) = i$ as $\{1 \mapsto 1, 2 \mapsto 2, 3 \mapsto 3\}$. We use indicators as special functions. For a set $V \subseteq W$ we use $\mathsf{Ind}(x \in V)\colon W \to \{0,1\}$ with $\mathsf{Ind}(x \in V)(v) = 1$ iff $v \in V$.

Distributions | A distribution over some discrete set V is a mapping $\delta\colon V \to \mathbb{R}_{\geq 0}$ such that $\sum_{v \in V} \delta(v) = 1$. For some fixed distribution, we call $\delta(v)$ the probability of v. The set of all distributions over V is denoted $Distr(V)$. The *support* of a distribution, *supp*(δ), is defined as the elements of the set that are assigned to a positive probability, i.e. $supp(\delta) = \{v \in V \mid \delta(v) \neq 0\}$. support

While a distribution maps from the elements of V, we sometimes opt for the reverse notation. For example we may write $1/3 : v_1, 2/3 : v_2$ to denote that v_1 occurs with probability $1/3$ and v_2 with probability $2/3$, i.e., to denote $\{v_1 \mapsto 1/3, v_2 \mapsto 2/3\}$.

Matrices | Matrices and vectors are elementary objects from linear algebra. We assume that the reader is familiar with their basics. We denote a matrix with entries from some set V and r rows and c columns as $A \in V^{r \times c}$. Vectors with r rows have a single column and are denoted $v \in V^r$. $A_{i,j}$ denotes the entry of A in the i'th row and the j'th column, and b_i the entry in the i'th row.

We recall that the *transpose* A^T of a matrix $A \in V^{r \times c}$ is a matrix in $V^{c \times r}$ such $A^T_{i,j} := A_{j,i}$. The matrix $A \in V^{k \times k}$ is symmetric if it is equal to its transpose, and recall that $t \in \mathbb{R}$ is an eigenvalue of a matrix $A \in \mathbb{R}^{k \times k}$ if there exists a vector $v \in \mathbb{R}^k$ such that $A \cdot v = t \cdot v$. Finally, a matrix $A \in \mathbb{R}^{k \times k}$ is *positive semidefinite* (*PSD*) if all eigenvalues of the matrix are non-negative. ·transpose ·PSD

A

·open
·interior

Topology | We define the (Euclidean) distance between two vectors $v, w \in \mathbb{R}^k$ as $\sum_{i=1}^{k} |v_i^2 - w_i^2|$. An *open* set $V \subseteq \mathbb{R}^k$ is a generalisation of an open interval into multiple dimensions. We define the *interior* points of V as

$$\{w \in V \mid \text{exists } r > 0 \text{ such that for all } v \in \mathbb{R}^k : |v - w| < r \text{ implies } v \in V\}.$$

·closed
·closure
·convex

If the interior of a set coincides with the set, the set is open. A set $V \subseteq \mathbb{R}^k$ is *closed*, if its complement is open, i.e., if $\mathbb{R}^k \setminus V$ is open. The *closure* of a set V is the smallest closed set that contains V, i.e., it is the intersection of all closed sets that contain V. A set is *connected*, if it cannot be partitioned into two sets V, W such that none of the points of V lie in the closure of W and vice versa. A set V is *convex*, if for all $v, w \in V$ and $t \in \mathbb{R}$ with $0 \leq t \leq 1$, it holds that $t \cdot v + (1 - t) \cdot w$ is in V. A convex polyhedron is the set of all solutions of finitely many linear inequalities, i.e., a set V is a convex polyhedron if there exists a matrix $A \in \mathbb{R}^{k \times k}$ and a vector $b \in \mathbb{R}^k$ such that $V = \{v \mid Av \leq b\}$.

B Complexity

B.1 Basics

We refrain from giving a thorough and completely formalised introduction to computational complexity, cf. [Pap94] for details.

Reductions | Decision problems can be considered as formal languages describing all positive instances. A language $L_1 \subseteq \{0,1\}^*$ is *polynomially (many-one or Karp) reducible* to $L_2 \subseteq \{0,1\}^*$. if there exists a polynomial-time computable function $f\colon \{0,1\}^* \to \{0,1\}^*$ such that for all $w \in \{0,1\}^*$, $w \in L_1$ iff $f(w) \in L_2$. Polynomial reductions are essential to characterise complexity classes.

Complexity classes | For this thesis, some essential complexity classes are:

- P, the set of problems that may be solved in polynomial time by a deterministic Turing machine,
- NP, the set of problems that may be solved in polynomial time by a nondeterministic Turing machine,
- PSPACE, the set of problems that may be solved using polynomial space by a (non)deterministic Turing machine.

A problem L is also in NP, if for a given certificate that describes the solution, i.e., for a polynomially long w, a deterministic Turing machine may check in polynomial time if this certificate is indeed a solution, i.e., if $w \in L$. If a problem L is in either of these complexity classes, and there exists a polynomial reduction from L' to L, then L' is also in the same complexity class.

Hardness and completeness | A problem L is hard for a complexity class C and polynomial reductions, if there is a polynomial reduction from every problem in C to L. Problems in a class C that are hard for C and polynomial reductions are called C-complete.

B.2 ETR Completeness

Definition (repeats Definition 5.21) The decision problem *modified-closed-bounded-4-feasibility* (mb4FEAS-c) asks: Given a non-negative quartic polynomial f,

$$\text{exists } \mathsf{val}\colon X \to [0,1] \text{ s.t. } f[\mathsf{val}] \leq 0?$$

The *modified-open-bounded-4-feasibility* (mb4FEAS-o) is analogously defined with val ranging over $(0,1)$.

This problem easily reduces to its $\geq$-variant by multiplying f with -1.

Lemma (repeats Lemma 5.50) The problems mb4FEAS-c and mb4FEAS-o are ETR-hard.

We start our reduction from the following ETR-complete problem:

Definition B.1 The decision problem *bounded-closed quadratic conjunction* (bQUAD-c) asks: Given quadratic polynomials $f_1, \ldots, f_m$, $\exists \mathsf{val}\colon \boldsymbol{x} \to [-1,1]$ s.t. $\bigwedge_{i=1}^m f_i[\mathsf{val}] = 0$? The open variant of the decision problem (bQUAD-o) is defined analogously, where the range of val is $(-1,1)$.

Theorem B.1 (From [Sch13, Lemma 3.9][a]) The bQUAD-o/c problems are ETR-complete.

[a] Strictly, the lemma is an immediate corollary to (the proof of) [Sch13, Lemma 3.9]. Although Schaefer states the result for the Euclidean norm and a closed ball, the proof ensures that if there is a solution to the original constraint system in the unbounded space, there is also a solution to the constructed constraint system in the open ball. Therefore, there is also a solution in the closed ball or the open/closed box. This adaption also occurs implicitly in [SS17], where a closed box is assumed.

We first reduce bQUAD to b4FEAS (like mb4FEAS, but with different range, and equality).

Definition B.2 The decision problem closed-bounded-4-feasibility (b4FEAS-c) asks: Given a (non-negative) polynomial f of degree at most 4, $\exists \mathsf{val}\colon \boldsymbol{x} \to [-1,1]$ s.t. $f[\mathsf{val}] = 0$? The open-bounded-4-feasibility (b4FEAS-o) may be defined analogously where $\mathsf{val}\colon \boldsymbol{x} \to (0,1)$.

Lemma B.2 bQUAD-c polynomially reduces to b4FEAS-c, and bQUAD-o polynomially reduces to b4FEAS-o.

Proof. The proof is a straightforward adaption from the proof of [SS17, Lemma 3.2]. For quadratic polynomials $f_1, \ldots, f_m$, consider the quartic polynomial $f = \sum_{i=1}^m (f_i)^2$. Clearly, each $(f_i)^2$ is non-negative and has the same roots as f_i. As the sum of non-negative polynomials is non-negative, f is non-negative, and has roots exactly where

the f_i have common roots. As it preserves all roots, it in particular preserves roots in any bounded domain such as $[-1,1]^{\boldsymbol{x}}$. The bitlength of the encoding at most doubles [SS17, Lemma 3.2]. ■

Lemma B.3 b4FEAS-c polynomially reduces to mb4FEAS-c and b4FEAS-o polynomially reduces to mb4FEAS-o.

Proof. We show this lemma in three steps:

1. We reduce b4FEAS-c to the problem

   P1-c: given a quartic[1] (non-negative) polynomial f',

   $$\exists \mathsf{val}\colon \boldsymbol{x} \to [0,2] \text{ s.t. } f'[\mathsf{val}] = 0.$$

   Let f be the input to b4FEAS-c. We construct f' as $f[x_1 \mapsto x_1 - 1, \ldots, x_n \mapsto x_n - 1]$.

   - **Example:** For $f = -x_1^2 x_2^2 + x_1^2$, we obtain $f' = -(x_1-1)^2(x_2-1)^2 + (x_1-1)^2$. Consider $\mathsf{val} = [x_1 \mapsto 1, x_2 \mapsto -1]$: We have that $f[\mathsf{val}] = 0$.
     Now consider $\mathsf{val}' = [x_1 \mapsto 2, x_2 \mapsto 0]$: We have $f'[\mathsf{val}'] = -1 \cdot 1 + 1 = 0$.
   - **Correctness:** For each $\mathsf{val} \in [-1,1]^X$, it holds that $f[\mathsf{val}] = f'[\mathsf{val}']$ for val' with $\mathsf{val}'(x_i) = \mathsf{val}(x_i) + 1$. Observe that $\mathsf{val}' \in [0,2]^X$. *There is a bijection between roots of f and f'*: Assume val to be a root of f. Then val' is a root of f' with val' as above. Analogously, any root val' of f' corresponds to a root of f. Each term remains (at most) quartic, thus f' is also quartic. As f is non-negative, so is f'.
   - **Complexity:** The substitution may be applied on a per-term basis: Consider the term $t_j = c_j \cdot x_{j_1} x_{j_2} x_{j_3} x_{j_4}$: It becomes the polynomial $f'_j = c_j \cdot (x_{j_1} - 1) \cdot (x_{j_2} - 1) \cdot (x_{j_3} - 1) \cdot (x_{j_4} - 1)$ which in its expanded form contains 16 terms $t'_{j,1}, \ldots, t'_{j,16}$ with (for each j) pairwise different monomials, thus all coefficients in f'_j have the same magnitude as the coefficient of t_j. We obtain $f' \sum_j \sum_i t_{j,i}$.

2. We reduce **P1**-c to the problem

   P2-c: given a quartic (non-neative) polynomial $\hat{f}$,

   $$\exists \mathsf{val}\colon \boldsymbol{x} \to [0,1] \text{ s.t. } \hat{f}[\mathsf{val}] = 0.$$

   Let f' be the input to **P1**-c. We construct $\hat{f}$ by $f'[x_1 \mapsto 2 \cdot x_1, \ldots, x_k \mapsto 2 \cdot x_k]$.

   - **Example:** For $f' = x_1^2 - 4x_1 + x_2 + 4$, we construct $\hat{f} = 4x_1^2 - 8x_1 + 2x_2 + 4$. Consider $\mathsf{val}' = [x_1 \mapsto 2, x_2 \mapsto 0]$: We have that $f'[\mathsf{val}'] = 4 - 8 + 0 + 4 = 0$. Now consider $\hat{\mathsf{val}} = [x_1 \mapsto 1, x_2 \mapsto 0]$: We have $\hat{f}[\hat{\mathsf{val}}] = 4 - 8 + 0 + 4 = 0$.

[1] degree at most 4

B

- **Correctness:** For each $\mathsf{val} \in [0,2]^X$, it holds that $f[\mathsf{val}] = f'[\mathsf{val}']$ for val' with $\mathsf{val}'(x_i) = \frac{\mathsf{val}(x_i)}{2}$. Observe that $\mathsf{val}' \in [0,1]^X$. *There is a bijection between roots of f and f'*: Assume val to be a root of f. Then val' is a root of f' with val' as above. Analogously, any root val' of f' corresponds to a root of f. *The polynomial remains non-negative.* The monomials (and thus the degree of the polynomial) remains unchanged.
- **Complexity:** Each coefficient is multiplied at most four times by two; and requires thus only constantly many bits more.

3. We reduce the problem **P2**-c to mb4FEAS-c: This reduction is straightforward, just observe that for a non-negative polynomial f, it holds that for any val, $f[\mathsf{val}] \leq 0$ iff $f[\mathsf{val}] = 0$.

The reduction chain reduces b4FEAS-c to mb4FEAS-c, and may be analogously applied to reduce b4FEAS-o to mb4FEAS-o. ■

Proof (Lemma 5.50). Immediate corollary to Theorem B.1, Lemmas B.2 and B.3. ■

Definition (repeats Definition 5.22) The decision problem *bounded-conjunction-of-inequalities* (bcon4INEQ-c) asks: Given a family of quartic polynomials $f_1, \ldots, f_m$:

$$\text{exists } \mathsf{val} : X \to [0,1] \text{ s.t. } \bigwedge_{i=1}^{m} f_i[\mathsf{val}] < 0?$$

The open variant (bcon4INEQ-o) may be defined analogously.

By a reduction from mb4FEAS, adapted from [SS17, Thm 4.1]:

Lemma (repeats Lemma 5.57) The bcon4INEQ-o/c problems are ETR-hard.

Proof (Lemma 5.57). (adapted from [SS17, Thm 4.1]) We reduce from mb4FEAS-c (ETR-hard by Lemma 5.50) to bcon4INEQ-c, using an adaption of the construction of the proof of [SS17, Thm. 4.1].

We ask for an $\mathsf{val} \in [0,1]^{\boldsymbol{x}}$ such that $f[\mathsf{val}] \leq 0$. As we can assume f to be non-negative, this is equivalent to asking for $f[\mathsf{val}] = 0$ (cf. **P2**-c in the proof of Lemma B.2). We introduce a fresh variable z. Consider the two semi-algebraic sets

$$A = \{\mathsf{val}' \in \mathbb{R}^{|\boldsymbol{x}|+1} \mid \mathsf{val} \models \big(f \doteq z \wedge \bigwedge_{x \in \boldsymbol{x}} 0 \leqslant x \leqslant 1\big)\},$$

and

$$B = \{\mathsf{val}' \in \mathbb{R}^{|\boldsymbol{x}|+1} \mid \mathsf{val} \models \big(z \doteq 0 \wedge \bigwedge_{x \in \boldsymbol{x}} 0 \leqslant x \leqslant 1\big)\}.$$

Observe that A and B are disjoint, iff no $\mathsf{val} \in \mathbb{R}^{\boldsymbol{x}}$ exists such that $f[\mathsf{val}] = 0$. We now construct an bcon4INEQ-c instance that is satisfiable if A and B are not disjoint.

The sets A and B are compact[1]. If A and B are disjoint, then there is a positive (minimal) distance δ between A and B. By [SS17, Corollary 3.8], δ is at least $2^{-2^{L+5}}$, where L is the minimal number of bits to encode f in a sum-of-terms fashion. Thus, if there exists a point where the distance is between A and B would be smaller than δ, then A and B cannot be disjoint, and hence they must overlap. By construction, the distance between A and B is given by difference between of f and 0. This observation yields the following constraints

$$|f - 0| \lessdot \delta \quad \wedge \quad \delta \lessdot 2^{-2^{L+5}} \quad \wedge \quad \mathsf{val} \in [0,1]^{\boldsymbol{x}}.$$

Clearly, if there is a satisfying assignment, then $0 < \delta < 1$, and as f is non-negative, we have $|f| = f$. To construct (in polynomial time) an equivalent ETR formula, have to reformulate the constant in the second constraint: Therefore, we reformulate $\delta < 2^{-2^{L+5}}$ by iterative squaring (or power iteration) to

$$\delta \lessdot \delta_1 \cdot \delta_1 \quad \wedge \quad \delta_1 \lessdot \delta_2 \cdot \delta_2 \quad \wedge \quad \ldots \quad \wedge \quad \delta_{L+3} \lessdot \delta_{L+4} \cdot \delta_{L+4} \quad \wedge \quad \delta_{L+4} \lessdot \frac{1}{2},$$

which is trivially to reformulate as

$$\Delta = \left(\delta - \delta_1 \cdot \delta_1 \lessdot 0 \quad \wedge \quad \ldots \quad \wedge \quad \delta_{L+3} - \delta_{L+4} \cdot \delta_{L+4} \lessdot 0 \quad \wedge \quad \delta_{L+4} - \frac{1}{2} \lessdot 0\right).$$

Together we obtain the following input to bcon4INEQ-c:

$$f - \delta \lessdot 0 \wedge \Delta,$$

which has a satisfying $\mathsf{val}' \colon \boldsymbol{x} \cup \{\delta, \delta_1, \ldots, \delta_{L+4}\} \to [0,1]$ iff there exists $\mathsf{val} \in [0,1]^{\boldsymbol{x}}$ s.t. $f[\mathsf{val}] \leq 0$. All constraints have degree at most degree 4. ■

[1]Notice that we use the same sets A, B also for reducing the open variant, to ensure compactness.

B

Declaration of Authorship

In accordance with the regulations regarding doctoral studies at the RWTH Aachen University Faculty of Mathematics, Computer Science and Natural Sciences, I include this statement as provided by the faculty:

I, Sebastian Junges,
declare that this thesis and the work presented in it are my own and have been generated by me as the result of my own original research.

Hiermit erkläre ich an Eides statt / I do solemnly swear that:

1. This work was done wholly or mainly while in candidature for the doctoral degree at this faculty and university;
2. Where any part of this thesis has previously been submitted for a degree or any other qualification at this university or any other institution, this has been clearly stated;
3. Where I have consulted the published work of others or myself, this is always clearly attributed;
4. Where I have quoted from the work of others or myself, the source is always given. This thesis is entirely my own work, with the exception of such quotations;
5. I have acknowledged all major sources of assistance;
6. Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
7. Parts of this work have been published before as detailed in Section 1.4 on page 20.

Sebastian Junges, September 2019, Aachen.

Bibliography

References[1]

[802a] 802 Committee. '*Standard for Ethernet*'. Technical report 802.3 (cited on page 6).

[ÁABB+16] E. Ábrahám, J. Abbott, B. Becker, A. M. Bigatti, M. Brain, B. Buchberger, A. Cimatti, J. H. Davenport, M. England, P. Fontaine, S. Forrest, A. Griggio, D. Kroening, W. M. Seiler and T. Sturm. '*Satisfiability checking and symbolic computation*'. *ACM Comm. Computer Algebra* 50.4 (2016), pages 145–147 (cited on page 227).

[AADF+06] D. Angluin, J. Aspnes, Z. Diamadi, M. J. Fischer and R. Peralta. '*Computation in networks of passively mobile finite-state sensors*'. *Distributed Computing* 18.4 (2006), pages 235–253 (cited on page 15).

[AAER07] D. Angluin, J. Aspnes, D. Eisenstat and E. Ruppert. '*The computational power of population protocols*'. *Distributed Computing* 20.4 (2007), pages 279–304 (cited on page 15).

[AB] J. Abbott and A. M. Bigatti. '*CoCoALib: a C++ library for doing Computations in Commutative Algebra*'. Available at `cocoa.dima.unige.it/cocoalib` (cited on page 320).

[ABCK+18] S. Arming, E. Bartocci, K. Chatterjee, J.-P. Katoen and A. Sokolova. '*Parameter-Independent Strategies for pMDPs via POMDPs*'. *QEST*. Volume 11024. LNCS. Springer, 2018, pages 53–70 (cited on pages 13, 157, 295).

[ÁBDJ+14] E. Ábrahám, B. Becker, C. Dehnert, N. Jansen, J.-P. Katoen and R. Wimmer. '*Counterexample Generation for Discrete-Time Markov Models: An Introductory Survey*'. *SFM*. Volume 8483. LNCS. Springer, 2014, pages 65–121 (cited on page 264).

[ABZ10] C. Amato, D. S. Bernstein and S. Zilberstein. '*Optimizing fixed-size stochastic controllers for POMDPs and decentralized POMDPs*'. *Autonomous Agents and Multi-Agent Systems* 21.3 (2010), pages 293–320 (cited on page 178).

[1] Papers co-authored by me are labelled with numbers and listed separately on Page 363.

[ADKK+18] A. Abate, C. David, P. Kesseli, D. Kroening and E. Polgreen. '*Counter-example Guided Inductive Synthesis Modulo Theories*'. *CAV*. Volume 10981. LNCS. Springer, 2018, pages 270–288 (cited on page 259).

[AG03] F. Alizadeh and D. Goldfarb. '*Second-order cone programming*'. *Math. Program.* 95.1 (2003), pages 3–51 (cited on page 245).

[AHK03] S. Andova, H. Hermanns and J.-P. Katoen. '*Discrete-Time Rewards Model-Checked*'. *FORMATS*. Volume 2791. LNCS. Springer, 2003, pages 88–104 (cited on page 43).

[AHK07] L. de Alfaro, T. A. Henzinger and O. Kupferman. '*Concurrent reachability games*'. *Theor. Comput. Sci.* 386.3 (2007), pages 188–217 (cited on pages 13, 193).

[AHLN+08] A. Antonik, M. Huth, K. G. Larsen, U. Nyman and A. Wasowski. '*20 Years of Modal and Mixed Specifications*'. *Bulletin of the EATCS* 95 (2008), pages 94–129 (cited on page 161).

[AHM00] L. de Alfaro, T. A. Henzinger and F. Y. C. Mang. '*The Control of Synchronous Systems*'. *CONCUR*. Volume 1877. LNCS. Springer, 2000, pages 458–473 (cited on pages 13, 193).

[AHV15] R. Alur, T. A. Henzinger and M. Y. Vardi. '*Theory in practice for system design and verification*'. *SIGLOG News* 2.1 (2015), pages 46–51 (cited on page 5).

[AJNd02] P. A. Abdulla, B. Jonsson, M. Nilsson and J. d'Orso. '*Regular Model Checking Made Simple and Efficient*'. *CONCUR*. Volume 2421. LNCS. Springer, 2002, pages 116–130 (cited on page 15).

[AJNS04] P. A. Abdulla, B. Jonsson, M. Nilsson and M. Saksena. '*A Survey of Regular Model Checking*'. *CONCUR*. Volume 3170. LNCS. Springer, 2004, pages 35–48 (cited on page 15).

[AK86] K. R. Apt and D. Kozen. '*Limits for Automatic Verification of Finite-State Concurrent Systems*'. *Inf. Process. Lett.* 22.6 (1986), pages 307–309 (cited on page 15).

[AKNP+00] L. de Alfaro, M. Z. Kwiatkowska, G. Norman, D. Parker and R. Segala. '*Symbolic Model Checking of Probabilistic Processes Using MTBDDs and the Kronecker Representation*'. *TACAS*. Volume 1785. LNCS. Springer, 2000, pages 395–410 (cited on page 286).

[APZ03] T. Arons, A. Pnueli and L. D. Zuck. '*Parameterized Verification by Probabilistic Abstraction*'. *FoSSaCS*. Volume 2620. LNCS. Springer, 2003, pages 87–102 (cited on page 15).

[AR16] S. Arora and M. V. P. Rao. '*Probabilistic Model Checking of Incomplete Models*'. *ISoLA*. Volume 9952. LNCS. 2016, pages 62–76 (cited on page 16).

[ASB95] A. Aziz, V. Singhal and F. Balarin. '*It Usually Works: The Temporal Logic of Stochastic Systems*'. *CAV*. Volume 939. LNCS. Springer, 1995, pages 155–165 (cited on page 47).

[ASFS18] R. Alur, R. Singh, D. Fisman and A. Solar-Lezama. '*Search-based program synthesis*'. *Commun. ACM* 61.12 (2018), pages 84–93 (cited on pages 173, 259).

[AST18] P. A. Abdulla, A. P. Sistla and M. Talupur. '*Model Checking Parameterized Systems*'. *Handbook of Model Checking*. Springer, 2018, pages 685–725 (cited on page 15).

[Åst65] K. Åström. '*Optimal control of Markov processes with incomplete state information*'. *Journal of Mathematical Analysis and Applications* 10.1 (1965), pages 174–205 (cited on pages 13, 178).

[Aur91] F. Aurenhammer. '*Voronoi Diagrams - A Survey of a Fundamental Geometric Data Structure*'. *ACM Comput. Surv.* 23.3 (1991), pages 345–405 (cited on page 229).

[AVBK+17] S. Aflaki, M. Volk, B. Bonakdarpour, J.-P. Katoen and A. Storjohann. '*Automated Fine Tuning of Probabilistic Self-Stabilizing Algorithms*'. *SRDS*. IEEE Computer Society, 2017, pages 94–103 (cited on pages 12, 108, 220, 223).

[BAFK18] C. Baier, L. de Alfaro, V. Forejt and M. Kwiatkowska. '*Model Checking Probabilistic Systems*'. *Handbook of Model Checking*. Springer, 2018, pages 963–999 (cited on page 5).

[Bar72] E. H. Bareiss. '*Computational Solutions of Matrix Problems over an Integral Domain*'. *IMA Journal of Applied Mathematics* 10.1 (1972), pages 68–104 (cited on pages 210, 211).

[BB04] D. Braziunas and C. Boutilier. '*Stochastic Local Search for POMDP Controllers*'. *AAAI*. AAAI Press / The MIT Press, 2004, pages 690–696 (cited on page 178).

[BBPM99] L. Benini, A. Bogliolo, G. A. Paleologo and G. D. Micheli. '*Policy optimization for dynamic power management*'. *IEEE Trans. on CAD of Integrated Circuits and Systems* 18.6 (1999), pages 813–833 (cited on page 198).

[BCDS13] L. Brim, M. Ceska, S. Drazan and D. Safránek. '*Exploring Parameter Space of Stochastic Biochemical Systems Using Quantitative Model Checking*'. *CAV*. Volume 8044. LNCS. Springer, 2013, pages 107–123 (cited on pages 15, 304).

[BCHK+97] C. Baier, E. M. Clarke, V. Hartonas-Garmhausen, M. Z. Kwiatkowska and M. Ryan. '*Symbolic Model Checking for Probabilistic Processes*'. *ICALP*. Volume 1256. LNCS. Springer, 1997, pages 430–440 (cited on page 217).

[BDDK+14] C. Baier, M. Daum, C. Dubslaff, J. Klein and S. Klüppelholz. '*Energy-Utility Quantiles*'. *NASA Formal Methods*. Volume 8430. LNCS. Springer, 2014, pages 285–299 (cited on page 7).

[BDFL+18] A. Bart, B. Delahaye, P. Fournier, D. Lime, E. Monfroy and C. Truchet. '*Reachability in parametric Interval Markov Chains using constraints*'. *Theor. Comput. Sci.* 747 (2018), pages 48–74 (cited on page 16).

[BEK18] M. Blondin, J. Esparza and A. Kucera. '*Automatic Analysis of Expected Termination Time for Population Protocols*'. *CONCUR*. Volume 118. LIPIcs. Schloss Dagstuhl - LZI, 2018, 33:1–33:16 (cited on page 15).

[BEKK13] T. Brázdil, J. Esparza, S. Kiefer and A. Kucera. '*Analyzing probabilistic pushdown automata*'. *Formal Methods in System Design* 43.2 (2013), pages 124–163 (cited on page 15).

[Ber89] J. Bertrand. '*Calcul des probabilités*'. Gauthier-Villars, 1889 (cited on page 161).

[BFK02] C. Bauer, A. Frink and R. Kreckel. '*Introduction to the GiNaC Framework for Symbolic Computation within the C++ Programming Language*'. *J. Symb. Comput.* 33.1 (2002), pages 1–12 (cited on page 320).

[BG06] F. Benhamou and L. Granvilliers. '*Continuous and Interval Constraints*'. *Handbook of Constraint Programming*. Volume 2. Foundations of Artificial Intelligence. Elsevier, 2006, pages 571–603 (cited on page 224).

[BGKR+11] E. Bartocci, R. Grosu, P. Katsaros, C. R. Ramakrishnan and S. A. Smolka. '*Model Repair for Probabilistic Systems*'. *TACAS*. Volume 6605. LNCS. Springer, 2011, pages 326–340 (cited on pages 11, 76).

[BHHK10] C. Baier, B. R. Haverkort, H. Hermanns and J.-P. Katoen. '*Performance evaluation and model checking join forces*'. *Commun. ACM* 53.9 (2010), pages 76–85 (cited on page 7).

[BHRV12] A. Bouajjani, P. Habermehl, A. Rogalewicz and T. Vojnar. '*Abstract regular (tree) model checking*'. *STTT* 14.2 (2012), pages 167–191 (cited on page 15).

[BJKK+15] R. Bloem, S. Jacobs, A. Khalimov, I. Konnov, S. Rubin, H. Veith and J. Widder. '*Decidability of parameterized verification*'. *Synthesis Lectures on Distributed Computing Theory* 6.1 (2015), pages 1–170 (cited on page 15).

[BJMR05] J. L. Bresina, A. K. Jónsson, P. H. Morris and K. Rajan. '*Activity Planning for the Mars Exploration Rovers*'. *ICAPS*. AAAI, 2005, pages 40–49 (cited on page 7).

[BK08] C. Baier and J.-P. Katoen. '*Principles of model checking*'. MIT Press, 2008 (cited on pages 1, 4, 5, 35, 40, 42, 43, 46, 47, 51, 54, 84, 147, 219).

[BKLM+12] N. Benes, J. Křetínský, K. G. Larsen, M. H. Møller and J. Srba. '*Dual-Priced Modal Transition Systems with Time Durations*'. *LPAR*. Volume 7180. LNCS. Springer, 2012, pages 122–137 (cited on page 161).

[BKLM+15] N. Benes, J. Kretínský, K. G. Larsen, M. H. Møller, S. Sickert and J. Srba. '*Refinement checking on parametric modal transition systems*'. *Acta Inf.* 52.2-3 (2015), pages 269–297 (cited on page 161).

[BKLP+17] C. Baier, J. Klein, L. Leuschner, D. Parker and S. Wunderlich. '*Ensuring the Reliability of Your Model Checker: Interval Iteration for Markov Decision Processes*'. *CAV*. Volume 10426. LNCS. Springer, 2017, pages 160–180 (cited on page 230).

[BMS16] L. Bortolussi, D. Milios and G. Sanguinetti. '*Smoothed model checking for uncertain Continuous-Time Markov Chains*'. *Inf. Comput.* 247 (2016), pages 235–253 (cited on page 16).

[BPR06] S. Basu, R. Pollack and M.-F. Roy. '*Existential theory of the reals*'. *Algorithms in Real Algebraic Geometry* (2006) (cited on pages 25, 32, 223).

[Bro04] C. W. Brown. '*QEPCAD B: a system for computing with semi-algebraic sets via cylindrical algebraic decomposition*'. *ACM SIGSAM Bulletin* 38.1 (2004), pages 23–24 (cited on page 224).

[BS18] L. Bortolussi and S. Silvetti. '*Bayesian Statistical Parameter Synthesis for Linear Temporal Properties of Stochastic Models*'. *TACAS*. Volume 10806. LNCS. Springer, 2018, pages 396–413 (cited on page 16).

[BSHV03] H. C. Bohnenkamp, P. van der Stok, H. Hermanns and F. W. Vaandrager. '*Cost-Optimization of the IPv4 Zeroconf Protocol*'. *DSN*. IEEE Computer Society, 2003, pages 531–540 (cited on page 6).

[Buc65] B. Buchberger. '*Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal*'. PhD thesis. 1965 (cited on page 224).

[BV04] S. Boyd and L. Vandenberghe. '*Convex Optimization*'. Cambridge University Press, 2004 (cited on pages 25, 110, 245).

[CÁ11] F. Corzilius and E. Ábrahám. '*Virtual Substitution for SMT-Solving*'. *FCT*. Volume 6914. LNCS. Springer, 2011, pages 360–371 (cited on page 224).

[CAH04] K. Chatterjee, L. de Alfaro and T. A. Henzinger. '*Trading Memory for Randomness*'. *QEST*. IEEE Computer Society, 2004, pages 206–217 (cited on page 178).

[CAH13] K. Chatterjee, L. de Alfaro and T. A. Henzinger. '*Strategy improvement for concurrent reachability and turn-based stochastic safety games*'. *J. Comput. Syst. Sci.* 79.5 (2013), pages 640–657 (cited on page 193).

[Can88] J. F. Canny. '*Some Algebraic and Geometric Computations in PSPACE*'. *STOC*. ACM, 1988, pages 460–467 (cited on page 32).

[CCD16] K. Chatterjee, M. Chmelik and J. Davies. '*A Symbolic SAT-Based Algorithm for Almost-Sure Reachability with Small Strategies in POMDPs*'. *AAAI*. AAAI Press, 2016, pages 3225–3232 (cited on page 87).

[CCGK+17] R. Calinescu, M. Ceska, S. Gerasimou, M. Kwiatkowska and N. Paoletti. '*RODES: A Robust-Design Synthesis Tool for Probabilistic Systems*'. *QEST*. Volume 10503. LNCS. Springer, 2017, pages 304–308 (cited on pages 174, 319).

[CCGK+18] R. Calinescu, M. Ceska, S. Gerasimou, M. Kwiatkowska and N. Paoletti. '*Efficient synthesis of robust models for stochastic systems*'. *Journal of Systems and Software* 143 (2018), pages 140–158 (cited on pages 12, 15, 174).

[CCGK15] K. Chatterjee, M. Chmelik, R. Gupta and A. Kanodia. '*Qualitative analysis of POMDPs with temporal logic specifications for robotics applications*'. *ICRA*. IEEE, 2015, pages 325–330 (cited on page 180).

[CCGK16] K. Chatterjee, M. Chmelik, R. Gupta and A. Kanodia. '*Optimal cost almost-sure reachability in POMDPs*'. *Artif. Intell.* 234 (2016), pages 26–48 (cited on page 178).

[CCHR+11] P. Cerný, K. Chatterjee, T. A. Henzinger, A. Radhakrishna and R. Singh. '*Quantitative Synthesis for Concurrent Programs*'. *CAV*. Volume 6806. LNCS. Springer, 2011, pages 243–259 (cited on page 177).

[CCMN+12] I. Chades, J. Carwardine, T. G. Martin, S. Nicol, R. Sabbadin and O. Buffet. '*MOMDPs: A Solution for Modelling Adaptive Management Problems*'. *AAAI*. AAAI Press, 2012 (cited on pages 16, 178, 295).

[CD14] K. Chatterjee and L. Doyen. '*Partial-Observation Stochastic Games: How to Win when Belief Fails*'. *ACM Trans. Comput. Log.* 15.2 (2014), 16:1–16:44 (cited on page 193).

[CDKB18] P. Chrszon, C. Dubslaff, S. Klüppelholz and C. Baier. '*ProFeat: feature-oriented engineering for family-based probabilistic model checking*'. *Formal Asp. Comput.* 30.1 (2018), pages 45–75 (cited on pages 13, 175, 197, 199, 200, 284, 286, 295, 319).

[CDKP14] M. Ceska, F. Dannenberg, M. Z. Kwiatkowska and N. Paoletti. '*Precise Parameter Synthesis for Stochastic Biochemical Systems*'. *CMSB*. Volume 8859. LNCS. Springer, 2014, pages 86–98 (cited on page 15).

[CDLL+11] B. Caillaud, B. Delahaye, K. G. Larsen, A. Legay, M. L. Pedersen and A. Wasowski. '*Constraint Markov Chains*'. *Theor. Comput. Sci.* 412.34 (2011), pages 4373–4404 (cited on page 16).

[CES09] E. M. Clarke, E. A. Emerson and J. Sifakis. '*Model checking: algorithmic verification and debugging*'. *Commun. ACM* 52.11 (2009), pages 74–84 (cited on page 4).

[CFRS14] T. Chen, Y. Feng, D. S. Rosenblum and G. Su. '*Perturbation Analysis in Verification of Discrete-Time Markov Chains*'. *CONCUR*. Volume 8704. LNCS. Springer, 2014, pages 218–233 (cited on page 78).

[CGIR+17] A. Cimatti, A. Griggio, A. Irfan, M. Roveri and R. Sebastiani. '*Invariant Checking of NRA Transition Systems via Incremental Reduction to LRA with EUF*'. *TACAS*. Volume 10205. LNCS. 2017, pages 58–75 (cited on page 222).

[CGJP+16] R. Calinescu, C. Ghezzi, K. Johnson, M. Pezzè, Y. Rafiq and G. Tamburrelli. '*Formal Verification With Confidence Intervals to Establish Quality of Service Properties of Software Systems*'. *IEEE Trans. Reliability* 65.1 (2016), pages 107–125 (cited on page 12).

[CGKM12] R. Calinescu, C. Ghezzi, M. Z. Kwiatkowska and R. Mirandola. '*Self-adaptive software needs quantitative verification at runtime*'. *Commun. ACM* 55.9 (2012), pages 69–77 (cited on page 7).

[CH91] G. E. Collins and H. Hong. '*Partial Cylindrical Algebraic Decomposition for Quantifier Elimination*'. *J. Symb. Comput.* 12.3 (1991), pages 299–328 (cited on page 224).

[CHHK+13] T. Chen, E. M. Hahn, T. Han, M. Z. Kwiatkowska, H. Qu and L. Zhang. '*Model Repair for Markov Decision Processes*'. *TASE*. IEEE Computer Society, 2013, pages 85–92 (cited on pages 11, 227).

[CHI17] K. Chatterjee, K. A. Hansen and R. Ibsen-Jensen. '*Strategy Complexity of Concurrent Safety Games*'. *MFCS*. Volume 83. LIPIcs. Schloss Dagstuhl - LZI, 2017, 55:1–55:13 (cited on page 193).

[CHK13] T. Chen, T. Han and M. Z. Kwiatkowska. '*On the complexity of model checking interval-valued discrete time Markov chains*'. *Inf. Process. Lett.* 113.7 (2013), pages 210–216 (cited on page 16).

[CHLS+14] M. Cordy, P. Heymans, A. Legay, P.-Y. Schobbens, B. Dawagne and M. Leucker. '*Counterexample guided abstraction refinement of product-line behavioural models*'. *SIGSOFT FSE*. ACM, 2014, pages 190–201 (cited on page 284).

[Cho17] V. Chonev. '*Reachability in Augmented Interval Markov Chains*'. *CoRR* abs/1701.02996 (2017) (cited on pages 16, 127, 129, 134, 143).

[Chu67] K. L. Chung. '*Markov chains*'. Springer, 1967 (cited on pages 2, 35).

[CK18] G. Chatzieleftheriou and P. Katsaros. '*Abstract model repair for probabilistic systems*'. *Inf. Comput.* 259.1 (2018) (cited on page 11).

[CLO97] D. Cox, J. Little and D. O'Shea. '*Ideals, varieties, and algorithms: an introduction to computational algebraic geometry and commutative algebra*'. Springer, 1997 (cited on page 224).

[CLRS09] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein. '*Introduction to Algorithms, 3rd Edition*'. MIT Press, 2009 (cited on page 217).

[Col75] G. E. Collins. '*Hauptvortrag: Quantifier elimination for real closed fields by cylindrical algebraic decomposition*'. *Automata Theory and Formal Languages*. Volume 33. LNCS. Springer, 1975, pages 134–183 (cited on page 224).

[Con90] A. Condon. '*On Algorithms for Simple Stochastic Games*'. *Advances In Computational Complexity Theory*. Volume 13. DIMACS Series in Discrete Mathematics and Theoretical Computer Science. DIMACS/AMS, 1990, pages 51–72 (cited on pages 54, 55).

[Coo71] S. A. Cook. '*The Complexity of Theorem-Proving Procedures*'. *STOC*. ACM, 1971, pages 151–158 (cited on page 32).

[Cou09] D. Cousineau. '*Fitting the three-parameter Weibull distribution: review and evaluation of existing and new methods*'. *IEEE Transactions on Dielectrics and Electrical Insulation* 16.1 (2009), pages 281–288 (cited on page 12).

[CPPB+16] M. Ceska, P. Pilar, N. Paoletti, L. Brim and M. Z. Kwiatkowska. '*PRISM-PSY: Precise GPU-Accelerated Parameter Synthesis for Stochastic Systems*'. *TACAS*. Volume 9636. LNCS. Springer, 2016, pages 367–384 (cited on pages 15, 319).

[CS13] A. Chakarov and S. Sankaranarayanan. '*Probabilistic Program Analysis with Martingales*'. *CAV*. Volume 8044. LNCS. Springer, 2013, pages 511–526 (cited on page 15).

[CSH08] K. Chatterjee, K. Sen and T. A. Henzinger. '*Model-Checking omega-Regular Properties of Interval Markov Chains*'. *FoSSaCS*. Volume 4962. LNCS. Springer, 2008, pages 302–317 (cited on page 16).

[CT18] M. Cubuktepe and U. Topcu. '*Verification of Markov Decision Processes with Risk-Sensitive Measures*'. *ACC*. IEEE, 2018, pages 2371–2377 (cited on page 157).

[CV10] R. Chadha and M. Viswanathan. '*A counterexample-guided abstraction-refinement framework for markov decision processes*'. *ACM Trans. Comput. Log.* 12.1 (2010), 1:1–1:49 (cited on page 284).

[CY88] C. Courcoubetis and M. Yannakakis. '*Verifying Temporal Properties of Finite-State Probabilistic Programs*'. *FOCS*. IEEE Computer Society, 1988, pages 338–345 (cited on page 5).

[Daw04] C. Daws. '*Symbolic and Parametric Model Checking of Discrete-Time Markov Chains*'. *ICTAC*. Volume 3407. LNCS. Springer, 2004, pages 280–294 (cited on pages 8, 64, 213).

[DB16] S. Diamond and S. P. Boyd. '*CVXPY: A Python-Embedded Modeling Language for Convex Optimization*'. *Journal of Machine Learning Research* 17 (2016), 83:1–83:5 (cited on page 254).

[DBW19] A. S. Dimovski, C. Brabrand and A. Wasowski. '*Finding suitable variability abstractions for lifted analysis*'. *Formal Asp. Comput.* 31.2 (2019), pages 231–259 (cited on page 284).

[DFT16] R. Dimitrova, J. Fu and U. Topcu. '*Robust optimal policies for Markov decision processes with safety-threshold constraints*'. *CDC*. IEEE, 2016, pages 7081–7086 (cited on pages 13, 157).

[DJJL01] P. R. D'Argenio, B. Jeannet, H. E. Jensen and K. G. Larsen. '*Reachability Analysis of Probabilistic Systems by Successive Refinements*'. *PAPM-PROBMIV*. Volume 2165. LNCS. Springer, 2001, pages 39–56 (cited on page 6).

[DJWÁ+14] C. Dehnert, N. Jansen, R. Wimmer, E. Ábrahám and J.-P. Katoen. '*Fast Debugging of PRISM Models*'. *ATVA*. Volume 8837. LNCS. Springer, 2014, pages 146–162 (cited on pages 273–275, 279, 319).

[DKB14] C. Dubslaff, S. Klüppelholz and C. Baier. '*Probabilistic model checking for energy analysis in software product lines*'. *MODULARITY*. ACM, 2014, pages 169–180 (cited on page 197).

[DKLL+13] B. Delahaye, J.-P. Katoen, K. G. Larsen, A. Legay, M. L. Pedersen, F. Sher and A. Wasowski. '*Abstract Probabilistic Automata*'. *Inf. Comput.* 232 (2013), pages 66–116 (cited on page 161).

[DKNP06] M. Duflot, M. Z. Kwiatkowska, G. Norman and D. Parker. '*A formal analysis of bluetooth device discovery*'. *STTT* 8.6 (2006), pages 621–632 (cited on page 6).

[DLLP+12] B. Delahaye, K. G. Larsen, A. Legay, M. L. Pedersen and A. Wasowski. '*New results for Constraint Markov Chains*'. *Perform. Eval.* 69.7-8 (2012), pages 379–401 (cited on page 16).

[DLP16] B. Delahaye, D. Lime and L. Petrucci. '*Parameter Synthesis for Parametric Interval Markov Chains*'. *VMCAI*. Volume 9583. LNCS. Springer, 2016, pages 372–390 (cited on page 16).

[DP17] T. van Dijk and J. van de Pol. '*Sylvan: multi-core framework for decision diagrams*'. *STTT* 19.6 (2017), pages 675–696 (cited on page 320).

[DR18] R. Dureja and K. Y. Rozier. '*More Scalable LTL Model Checking via Discovering Design-Space Dependencies*'. *TACAS*. Volume 10805. LNCS. Springer, 2018, pages 309–327 (cited on page 262).

[Dut14] B. Dutertre. '*Yices 2.2*'. *CAV*. Volume 8559. LNCS. Springer, 2014, pages 737–744 (cited on page 223).

[EC80] E. A. Emerson and E. M. Clarke. '*Characterizing Correctness Properties of Parallel Programs Using Fixpoints*'. *ICALP*. Volume 85. LNCS. Springer, 1980, pages 169–181 (cited on page 4).

[EFM99] J. Esparza, A. Finkel and R. Mayr. '*On the Verification of Broadcast Protocols*'. *LICS*. IEEE Computer Society, 1999, pages 352–359 (cited on page 15).

[EG14] C. von Essen and D. Giannakopoulou. '*Analyzing the Next Generation Airborne Collision Avoidance System*'. *TACAS*. Volume 8413. LNCS. Springer, 2014, pages 620–635 (cited on page 7).

[EGK12] J. Esparza, A. Gaiser and S. Kiefer. '*Proving Termination of Probabilistic Programs Using Patterns*'. *CAV*. Volume 7358. LNCS. Springer, 2012, pages 123–138 (cited on page 15).

[EGL85] S. Even, O. Goldreich and A. Lempel. '*A Randomized Protocol for Signing Contracts*'. *Commun. ACM* 28.6 (1985), pages 637–647 (cited on page 6).

[EGLM17] J. Esparza, P. Ganty, J. Leroux and R. Majumdar. '*Verification of population protocols*'. *Acta Inf.* 54.2 (2017), pages 191–215 (cited on page 15).

[EK95] R. Eberhart and J. Kennedy. '*Particle swarm optimization*'. *IEEE International Conference on Neural Networks*. Volume 4. 1995, pages 1942–1948 (cited on page 228).

[ERW19] J. Esparza, M. Raskin and C. Weil-Kennedy. '*Parameterized Analysis of Immediate Observation Petri Nets*'. *Petri Nets*. Volume 11522. LNCS. Springer, 2019, pages 365–385 (cited on page 15).

[ES98] R. C. Eberhart and Y. Shi. '*Comparison between Genetic Algorithms and Particle Swarm Optimization*'. *Evolutionary Programming*. Volume 1447. LNCS. Springer, 1998, pages 611–616 (cited on page 228).

[FA09] L. Fribourg and É. André. '*An Inverse Method for Policy-Iteration Based Algorithms*'. *INFINITY*. Volume 10. EPTCS. 2009, pages 44–61 (cited on page 136).

[FGL15] A. Filieri, L. Grunske and A. Leva. '*Lightweight Adaptive Filtering for Efficient Learning and Updating of Probabilistic Models*'. *ICSE*. IEEE Computer Society, 2015, pages 200–211 (cited on page 7).

[FGT12] A. Filieri, C. Ghezzi and G. Tamburrelli. '*A formal approach to adaptive software: continuous assurance of non-functional requirements*'. *Formal Asp. Comput.* 24.2 (2012), pages 163–186 (cited on page 7).

[FKNP11] V. Forejt, M. Z. Kwiatkowska, G. Norman and D. Parker. '*Automated Verification Techniques for Probabilistic Systems*'. *SFM*. Volume 6659. LNCS. Springer, 2011, pages 53–113 (cited on page 5).

[FLW06] H. Fecher, M. Leucker and V. Wolf. '*Don't Know in Probabilistic Systems*'. *SPIN*. Volume 3925. LNCS. Springer, 2006, pages 71–88 (cited on page 16).

[FS01] A. Finkel and P. Schnoebelen. '*Well-structured transition systems everywhere!*' *Theor. Comput. Sci.* 256.1-2 (2001), pages 63–92 (cited on page 15).

[FTG16] A. Filieri, G. Tamburrelli and C. Ghezzi. '*Supporting Self-Adaptation via Quantitative Verification and Sensitivity Analysis at Run Time*'. *IEEE Trans. Software Eng.* 42.1 (2016), pages 75–99 (cited on pages 7, 10, 223).

[FV97] J. Filar and K. Vrieze. '*Competitive Markov Decision Processes*'. Springer, 1997 (cited on page 193).

[GDF14] S. Giro, P. R. D'Argenio and L. M. F. Fioriti. '*Distributed probabilistic input/output automata: Expressiveness, (un)decidability and algorithms*'. *Theor. Comput. Sci.* 538 (2014), pages 84–102 (cited on page 284).

[GF15] J. García and F. Fernández. '*A comprehensive survey on safe reinforcement learning*'. *J. Mach. Learn. Res.* 16 (2015), pages 1437–1480 (cited on page 7).

[GHS18] P. Gainer, E. M. Hahn and S. Schewe. '*Accelerated Model Checking of Parametric Markov Chains*'. *ATVA*. Volume 11138. LNCS. Springer, 2018, pages 300–316 (cited on page 230).

[GLD00] R. Givan, S. M. Leach and T. L. Dean. '*Bounded-parameter Markov decision processes*'. *Artif. Intell.* 122.1-2 (2000), pages 71–109 (cited on page 16).

[GPS17] S. Gulwani, O. Polozov and R. Singh. '*Program Synthesis*'. *Foundations and Trends in Programming Languages* 4.1-2 (2017), pages 1–119 (cited on page 173).

[GR12] S. Giro and M. N. Rabe. '*Verification of Partial-Information Probabilistic Systems Using Counterexample-Guided Refinements*'. *ATVA*. Volume 7561. LNCS. Springer, 2012, pages 333–348 (cited on page 284).

[GS13] C. Ghezzi and A. M. Sharifloo. '*Model-based verification of quantitative non-functional properties for software product lines*'. *Information & Software Technology* 55.3 (2013), pages 508–524 (cited on pages 13, 197, 284).

[GS92] S. M. German and A. P. Sistla. '*Reasoning about Systems with Many Processes*'. *J. ACM* 39.3 (1992), pages 675–735 (cited on page 15).

[GST19] A. Gouberman, M. Siegle and B. Tati. '*Markov chains with perturbed rates to absorption: Theory and application to model repair*'. *Perform. Eval.* 130 (2019), pages 32–50 (cited on page 16).

[GTC15] S. Gerasimou, G. Tamburrelli and R. Calinescu. '*Search-Based Synthesis of Probabilistic Models for Quality-of-Service Software Engineering (T)*'. *ASE*. IEEE Computer Society, 2015, pages 319–330 (cited on pages 197, 198, 278).

[Gur13] Gurobi Optimization, Inc. '*Gurobi Optimizer Reference Manual*'. Available at `www.gurobi.com`. 2013 (cited on pages 243, 254).

[Häg02] O. Häggström. '*Finite Markov chains and algorithmic applications*'. Volume 52. London Mathematical Society Student Texts. Cambridge University Press, 2002 (cited on pages 2, 35).

[Han13] Y. Han. '*State Elimination Heuristics for Short Regular Expressions*'. *Fundam. Inform.* 128.4 (2013), pages 445–462 (cited on pages 215, 216).

[Han88] D. Handelman. '*Representing polynomials by positive linear functions on compact convex polyhedra*'. *Pacific Journal of Mathematics* 132.1 (1988), pages 35–62 (cited on page 130).

[HBFK+18] L. Herrmann, C. Baier, C. Fetzer, S. Klüppelholz and M. Napierkowski. '*Formal Parameter Synthesis for Energy-Utility-Optimal Fault Tolerance*'. *EPEW*. Volume 11178. LNCS. Springer, 2018, pages 78–93 (cited on pages 7, 11).

[HBK17] L. Hutschenreiter, C. Baier and J. Klein. '*Parametric Markov Chains: PCTL Complexity and Fraction-free Gaussian Elimination*'. *GandALF*. Volume 256. EPTCS. 2017, pages 16–30 (cited on pages 10, 21, 125–127, 141, 146, 155).

[HBZ04] E. A. Hansen, D. S. Bernstein and S. Zilberstein. '*Dynamic Programming for Partially Observable Stochastic Games*'. *AAAI*. AAAI Press / The MIT Press, 2004, pages 709–715 (cited on page 193).

[HCA17] S. Haesaert, N. Cauchi and A. Abate. '*Certified policy synthesis for general Markov decision processes: An application in building automation systems*'. *Perform. Eval.* 117 (2017), pages 75–103 (cited on page 7).

[Hen13] T. A. Henzinger. '*Quantitative reactive modeling and verification*'. *Computer Science - R&D* 28.4 (2013), pages 331–344 (cited on page 7).

[Hen18] C. Hensel. '*The probabilistic model checker Storm: symbolic methods for probabilistic model checking*'. PhD thesis. RWTH Aachen University, Germany, 2018 (cited on pages 274, 319).

[Her90] T. Herman. '*Probabilistic Self-Stabilization*'. *Inf. Process. Lett.* 35.2 (1990), pages 63–67 (cited on pages 6, 295).

[HH14] A. Hartmanns and H. Hermanns. '*The Modest Toolset: An Integrated Environment for Quantitative Modelling and Verification*'. *TACAS*. Volume 8413. LNCS. Springer, 2014, pages 593–598 (cited on page 5).

[HHHL+17] E. M. Hahn, V. Hashemi, H. Hermanns, M. Lahijanian and A. Turrini. '*Multi-objective Robust Strategy Synthesis for Interval Markov Decision Processes*'. *QEST*. Volume 10503. LNCS. Springer, 2017, pages 207–223 (cited on pages 13, 16, 157).

[HHWZ09] E. M. Hahn, H. Hermanns, B. Wachter and L. Zhang. '*INFAMY: An Infinite-State Markov Model Checker*'. *CAV*. Volume 5643. LNCS. Springer, 2009, pages 641–647 (cited on page 15).

[HHWZ10] E. M. Hahn, H. Hermanns, B. Wachter and L. Zhang. '*PASS: Abstraction Refinement for Infinite Probabilistic Models*'. *TACAS*. Volume 6015. LNCS. Springer, 2010, pages 353–357 (cited on page 284).

[HHZ11a] E. M. Hahn, T. Han and L. Zhang. '*Synthesis for PCTL in Parametric Markov Decision Processes*'. *NASA Formal Methods*. Volume 6617. LNCS. Springer, 2011, pages 146–161 (cited on pages 8, 10, 64, 137).

[HHZ11b] E. M. Hahn, H. Hermanns and L. Zhang. '*Probabilistic reachability for parametric Markov models*'. *STTT* 13.1 (2011), pages 3–19 (cited on pages 10, 83, 115, 213, 216, 219, 314, 319, 320).

[HJ94] H. Hansson and B. Jonsson. '*A Logic for Reasoning about Time and Reliability*'. *Formal Asp. Comput.* 6.5 (1994), pages 512–535 (cited on pages 4, 47).

[HKM08] T. Han, J.-P. Katoen and A. Mereacre. '*Approximate Parameter Synthesis for Probabilistic Time-Bounded Reachability*'. *RTSS*. IEEE Computer Society, 2008, pages 173–182 (cited on page 15).

[HKPQ+19] A. Hartmanns, M. Klauck, D. Parker, T. Quatmann and E. Ruijters. '*The Quantitative Verification Benchmark Set*'. *TACAS*. Volume 11427. LNCS. Springer, 2019, pages 344–350 (cited on pages 5, 56).

[HLST+14] E. M. Hahn, Y. Li, S. Schewe, A. Turrini and L. Zhang. '*iscasMc: A Web-Based Probabilistic Model Checker*'. *FM*. Volume 8442. LNCS. Springer, 2014, pages 312–317 (cited on page 319).

[HM18] S. Haddad and B. Monmege. '*Interval iteration algorithm for MDPs and IMDPs*'. *Theor. Comput. Sci.* 735 (2018), pages 111–131 (cited on page 230).

[HMU03] J. E. Hopcroft, R. Motwani and J. D. Ullman. '*Introduction to Automata Theory, Languages, and Computation*'. Addison-Wesley, 2003 (cited on page 213).

[How71] R. Howard. '*Dynamic probabilistic systems: Semi-Markov and decision processes*'. Series in decision and control Part 2. John Wiley & Sons, 1971 (cited on pages 2, 35).

[HPRV76] C. Himmelberg, T. Parthasarathy, T. Raghavan and F. Van Vleck. '*Existence of p-equilibrium and optimal stationary strategies in stochastic games*'. *Proceedings of the American Mathematical Society* 60.1 (1976), pages 245–251 (cited on page 195).

[HWZ08] H. Hermanns, B. Wachter and L. Zhang. '*Probabilistic CEGAR*'. *CAV*. Volume 5123. LNCS. Springer, 2008, pages 162–175 (cited on page 284).

[JÁVW+12] N. Jansen, E. Ábrahám, M. Volk, R. Wimmer, J.-P. Katoen and B. Becker. '*The COMICS Tool - Computing Minimal Counterexamples for DTMCs*'. *ATVA*. Volume 7561. LNCS. Springer, 2012, pages 349–353 (cited on page 319).

[JCVW+14] N. Jansen, F. Corzilius, M. Volk, R. Wimmer, E. Ábrahám, J.-P. Katoen and B. Becker. '*Accelerating Parametric Probabilistic Verification*'. *QEST*. Volume 8657. LNCS. Springer, 2014, pages 404–420 (cited on pages 210, 216, 320).

[JL91] B. Jonsson and K. G. Larsen. '*Specification and Refinement of Probabilistic Processes*'. *LICS*. IEEE Computer Society, 1991, pages 266–277 (cited on pages 16, 117).

[JM12] D. Jovanovic and L. de Moura. '*Solving non-linear arithmetic*'. *ACM Comm. Computer Algebra* 46.3/4 (2012), pages 104–105 (cited on page 224).

[JS17] S. Jha and S. A. Seshia. '*A theory of formal synthesis via inductive learning*'. *Acta Inf.* 54.7 (2017), pages 693–726 (cited on page 259).

[Kal11] L. Kallenberg. '*Markov decision processes*'. *Lecture Notes. University of Leiden* (2011) (cited on pages 2, 5, 35).

[Kat16] J.-P. Katoen. '*The Probabilistic Model Checking Landscape*'. *LICS*. ACM, 2016, pages 31–45 (cited on page 5).

[Ken10] J. Kennedy. '*Particle Swarm Optimization*'. *Encyclopedia of Machine Learning*. Springer, 2010, pages 760–766 (cited on page 228).

[Kha79] L. G. Khachiyan. '*A polynomial algorithm in linear programming*'. *Doklady Academii Nauk SSSR.* Volume 244. 1979, pages 1093–1096 (cited on page 32).

[KKMO18] B. L. Kaminski, J.-P. Katoen, C. Matheja and F. Olmedo. '*Weakest Precondition Reasoning for Expected Runtimes of Randomized Algorithms*'. *J. ACM* 65.5 (2018), 30:1–30:68 (cited on page 15).

[KKNP10] M. Kattenbelt, M. Z. Kwiatkowska, G. Norman and D. Parker. '*A game-based abstraction-refinement framework for Markov decision processes*'. *Formal Methods in System Design* 36.3 (2010), pages 246–280 (cited on pages 284, 290).

[KLC98] L. P. Kaelbling, M. L. Littman and A. R. Cassandra. '*Planning and Acting in Partially Observable Stochastic Domains*'. *Artif. Intell.* 101.1-2 (1998), pages 99–134 (cited on pages 13, 178).

[KM18] J. Kretínský and T. Meggendorfer. '*Conditional Value-at-Risk for Reachability and Mean Payoff in Markov Decision Processes*'. *LICS.* ACM, 2018, pages 609–618 (cited on page 157).

[KMLB+18] J. A. Kromer, S. Märcker, S. Lange, C. Baier and B. M. Friedrich. '*Decision making improves sperm chemotaxis in the presence of noise*'. *PLoS computational biology* 14.4 (2018), e1006109 (cited on page 7).

[KMMP+01] Y. Kesten, O. Maler, M. Marcus, A. Pnueli and E. Shahar. '*Symbolic model checking with rich assertional languages*'. *Theor. Comput. Sci.* 256.1-2 (2001), pages 93–112 (cited on page 15).

[KMW12] A. Kolobov, Mausam and D. S. Weld. '*A Theory of Goal-Oriented MDPs with Dead Ends*'. *UAI.* AUAI Press, 2012, pages 438–447 (cited on page 188).

[KNP08] M. Z. Kwiatkowska, G. Norman and D. Parker. '*Using probabilistic model checking in systems biology*'. *SIGMETRICS Performance Evaluation Review* 35.4 (2008), pages 14–21 (cited on page 7).

[KNP11] M. Z. Kwiatkowska, G. Norman and D. Parker. '*PRISM 4.0: Verification of Probabilistic Real-Time Systems*'. *CAV.* Volume 6806. LNCS. Springer, 2011, pages 585–591 (cited on pages 5, 319).

[KNP12] M. Z. Kwiatkowska, G. Norman and D. Parker. '*Probabilistic verification of Herman's self-stabilisation algorithm*'. *Formal Asp. Comput.* 24.4-6 (2012), pages 661–670 (cited on pages 6, 220, 295).

[KNPS06] M. Z. Kwiatkowska, G. Norman, D. Parker and J. Sproston. '*Performance analysis of probabilistic timed automata using digital clocks*'. *Formal Methods in System Design* 29.1 (2006), pages 33–78 (cited on page 6).

[KNPS18] M. Kwiatkowska, G. Norman, D. Parker and G. Santos. '*Automated Verification of Concurrent Stochastic Games*'. *QEST*. Volume 11024. LNCS. Springer, 2018, pages 223–239 (cited on pages 64, 193).

[KNPV09] M. Z. Kwiatkowska, G. Norman, D. Parker and M. G. Vigliotti. '*Probabilistic Mobile Ambients*'. *Theor. Comput. Sci.* 410.12-13 (2009), pages 1272–1303 (cited on page 278).

[KNS03] M. Z. Kwiatkowska, G. Norman and J. Sproston. '*Probabilistic Model Checking of Deadline Properties in the IEEE 1394 FireWire Root Contention Protocol*'. *Formal Asp. Comput.* 14.3 (2003), pages 295–318 (cited on page 6).

[Koc15] M. J. Kochenderfer. '*Decision Making Under Uncertainty: Theory and Application*'. 1st. The MIT Press, 2015 (cited on page 178).

[Kre17] J. Kretínský. '*30 Years of Modal Transition Systems: Survey of Extensions and Analysis*'. *Models, Algorithms, Logics and Tools*. Volume 10460. LNCS. Springer, 2017, pages 36–74 (cited on page 161).

[Kri63] S. Kripke. '*Semantical Consideration on Modal Logic.*' *Acta Philosophica Fennica* 16 (1963), pages 83–94 (cited on page 1).

[KS76] J. G. Kemeny and J. L. Snell. '*Markov Chains*'. Springer, 1976 (cited on pages 2, 35).

[KT97] I. Karafyllidis and A. Thanailakis. '*A model for predicting forest fire spreading using cellular automata*'. *Ecological Modelling* 99.1 (1997), pages 87–97 (cited on page 7).

[KU02] I. Kozine and L. V. Utkin. '*Interval-Valued Finite Markov Chains*'. *Reliable Computing* 8.2 (2002), pages 97–113 (cited on page 16).

[KY76] D. Knuth and A. Yao. '*Algorithms and Complexity: New Directions and Recent Results*'. Academic Press, 1976. Chapter The complexity of nonuniform random number generation (cited on pages 3, 62).

[KZHH+11] J.-P. Katoen, I. S. Zapreev, E. M. Hahn, H. Hermanns and D. N. Jansen. '*The ins and outs of the probabilistic model checker MRMC*'. *Perform. Eval.* 68.2 (2011), pages 90–104 (cited on page 320).

[LB16] T. Lipp and S. Boyd. '*Variations and extension of the convex–concave procedure*'. *Optimization and Engineering* 17.2 (2016), pages 263–287 (cited on pages 247, 249, 253).

[LCK95] M. L. Littman, A. R. Cassandra and L. P. Kaelbling. '*Learning Policies for Partially Observable Environments: Scaling Up*'. *ICML*. Morgan Kaufmann, 1995, pages 362–370 (cited on page 178).

[LMT05] R. Lanotte, A. Maggiolo-Schettini and A. Troina. '*Automatic Analysis of a Non-Repudiation Protocol*'. *Electr. Notes Theor. Comput. Sci.* 112 (2005), pages 113–129 (cited on page 6).

[LMT07] R. Lanotte, A. Maggiolo-Schettini and A. Troina. '*Parametric probabilistic transition systems for system design and analysis*'. *Formal Asp. Comput.* 19.1 (2007), pages 93–109 (cited on pages 8, 64, 107).

[LR16] A. W. Lin and P. Rümmer. '*Liveness of Randomised Parameterised Systems under Arbitrary Schedulers*'. *CAV*. Volume 9780. LNCS. Springer, 2016, pages 112–133 (cited on page 15).

[LS16] E. A. Lee and S. A. Seshia. '*Introduction to embedded systems: A cyber-physical systems approach*'. Mit Press, 2016 (cited on page 1).

[LSCÁ+13] U. Loup, K. Scheibler, F. Corzilius, E. Ábrahám and B. Becker. '*A Symbiosis of Interval Constraint Propagation and Cylindrical Algebraic Decomposition*'. *CADE*. Volume 7898. LNCS. Springer, 2013, pages 193–207 (cited on page 224).

[LT88] K. G. Larsen and B. Thomsen. '*A Modal Process Logic*'. *LICS*. IEEE Computer Society, 1988, pages 203–210 (cited on page 161).

[LWDA18] Y. Z. Lun, J. Wheatley, A. D'Innocenzo and A. Abate. '*Approximate Abstractions of Markov Chains with Interval Decision Processes*'. *ADHS*. Volume 51. IFAC-PapersOnLine 16. Elsevier, 2018, pages 91–96 (cited on page 16).

[LXM16] S. H. Lim, H. Xu and S. Mannor. '*Reinforcement Learning in Robust Markov Decision Processes*'. *Math. Oper. Res.* 41.4 (2016), pages 1325–1353 (cited on pages 13, 157).

[MB01] M. J. McGlynn and S. A. Borbash. '*Birthday protocols for low energy deployment and flexible neighbor discovery in ad hoc wireless networks*'. *MobiHoc*. ACM, 2001, pages 137–145 (cited on page 6).

[MB08] L. M. de Moura and N. Bjørner. '*Z3: An Efficient SMT Solver*'. *TACAS*. Volume 4963. LNCS. Springer, 2008, pages 337–340 (cited on pages 223, 277, 295, 321).

[MB89] M. Mushkin and I. Bar-David. '*Capacity and coding for the Gilbert-Elliot channels*'. *IEEE Trans. Information Theory* 35.6 (1989), pages 1277–1290 (cited on page 6).

[McC92] R. A. McCallum. '*First Results with Utile Distinction Memory for Reinforcement Learning*'. Technical report. 1992 (cited on page 178).

[MEO19] P. J. Meyer, J. Esparza and P. Offtermatt. '*Computing the Expected Execution Time of Probabilistic Workflow Nets*'. *TACAS*. Volume 11428. LNCS. Springer, 2019, pages 154–171 (cited on page 7).

[MHC03] O. Madani, S. Hanks and A. Condon. '*On the undecidability of probabilistic planning and related stochastic optimization problems*'. *Artif. Intell.* 147.1-2 (2003), pages 5–34 (cited on pages 13, 178).

[MJ13] L. M. de Moura and D. Jovanovic. '*A Model-Constructing Satisfiability Calculus*'. *VMCAI.* Volume 7737. LNCS. Springer, 2013, pages 1–12 (cited on page 224).

[MKKC99] N. Meuleau, K. Kim, L. P. Kaelbling and A. R. Cassandra. '*Solving POMDPs by Searching the Space of Finite Policies*'. *UAI.* Morgan Kaufmann, 1999, pages 417–426 (cited on pages 178, 189).

[MM05] A. McIver and C. Morgan. '*Abstraction, Refinement and Proof for Probabilistic Systems*'. Monographs in Computer Science. Springer, 2005 (cited on page 15).

[MMAG14] I. Meedeniya, I. Moser, A. Aleti and L. Grunske. '*Evaluating probabilistic models with uncertain model parameters*'. *Software and System Modeling* 13.4 (2014), pages 1395–1415 (cited on page 12).

[MMX12] S. Mannor, O. Mebel and H. Xu. '*Lightning Does Not Strike Twice: Robust MDPs with Coupled Uncertainty*'. *ICML.* icml.cc / Omnipress, 2012 (cited on pages 13, 157).

[Moo79] R. E. Moore. '*Methods and applications of interval analysis*'. SIAM studies in applied mathematics. SIAM, 1979 (cited on page 224).

[MP13] L. M. de Moura and G. O. Passmore. '*The Strategy Challenge in SMT Solving*'. *Automated Reasoning and Mathematics.* Volume 7788. LNCS. Springer, 2013, pages 15–44 (cited on page 224).

[MWLR+07] R. Muneepeerakul, J. S. Weitz, S. A. Levin, A. Rinaldo and I. Rodriguez-Iturbe. '*A neutral metapopulation model of biodiversity in river networks*'. *Journal of Theoretical Biology* 245.2 (2007), pages 351–363 (cited on page 7).

[NB15] S. Navlakha and Z. Bar-Joseph. '*Distributed information processing in biological and computational systems*'. *Commun. ACM* 58.1 (2015), pages 94–102 (cited on page 15).

[Neu56] J. von Neumann. '*Probabilistic logics and synthesis of reliable organisms from unreliable components*'. *Automata Studies.* Princeton University Press, 1956, pages 43–98 (cited on page 7).

[NG05] A. Nilim and L. E. Ghaoui. '*Robust Control of Markov Decision Processes with Uncertain Transition Matrices*'. *Operations Research* 53.5 (2005), pages 780–798 (cited on pages 13, 157).

[NM47] J. von Neumann and O. Morgenstern. '*Theory of games and economic behavior*'. Princeton University Press, 1947 (cited on page 193).

[NORV15] A. V. Nori, S. Ozair, S. K. Rajamani and D. Vijaykeerthy. '*Efficient synthesis of probabilistic programs*'. *PLDI*. ACM, 2015, pages 208–217 (cited on pages 174, 197).

[NPKS05] G. Norman, D. Parker, M. Z. Kwiatkowska and S. K. Shukla. '*Evaluating the reliability of NAND multiplexing with PRISM*'. *IEEE Trans. on CAD of Integrated Circuits and Systems* 24.10 (2005), pages 1629–1637 (cited on pages 7, 219).

[NPZ17] G. Norman, D. Parker and X. Zou. '*Verification and control of partially observable probabilistic systems*'. *Real-Time Systems* 53.3 (2017), pages 354–402 (cited on pages 178, 188, 189, 220, 255).

[NS06] G. Norman and V. Shmatikov. '*Analysis of probabilistic contract signing*'. *Journal of Computer Security* 14.6 (2006), pages 561–589 (cited on page 6).

[OCPB16] B. O'Donoghue, E. Chu, N. Parikh and S. P. Boyd. '*Conic Optimization via Operator Splitting and Homogeneous Self-Dual Embedding*'. *J. Optimization Theory and Applications* 169.3 (2016), pages 1042–1068 (cited on page 254).

[Oso12] T. Osogami. '*Robustness and risk-sensitivity in Markov decision processes*'. *NIPS*. 2012, pages 233–241 (cited on page 157).

[PÁJT+15] S. Pathak, E. Ábrahám, N. Jansen, A. Tacchella and J.-P. Katoen. '*A Greedy Approach for the Efficient Repair of Stochastic Models*'. *NFM*. Volume 9058. LNCS. Springer, 2015, pages 295–309 (cited on pages 11, 229).

[Pap94] C. H. Papadimitriou. '*Computational Complexity*'. Addison-Wesley, 1994 (cited on page 331).

[Par71] T. Parthasarathy. '*Discounted and positive stochastic games*'. *Bulletin of the American Mathematical Society* 77.1 (1971), pages 134–136 (cited on page 195).

[PB17] J. Park and S. Boyd. '*General Heuristics for Nonconvex Quadratically Constrained Quadratic Programming*'. *CoRR* abs/1703.07870 (2017) (cited on page 254).

[PLSS13] A. Puggelli, W. Li, A. L. Sangiovanni-Vincentelli and S. A. Seshia. '*Polynomial-Time Verification of PCTL Properties of MDPs with Convex Uncertainties*'. *CAV*. Volume 8044. LNCS. Springer, 2013, pages 527–542 (cited on page 16).

[PT16] H. A. Poonawala and U. Topcu. '*Filter-based stochastic abstractions for constrained planning with limited sensing*'. *CDC*. IEEE, 2016, pages 3319–3324 (cited on page 16).

[Put94] M. L. Puterman. '*Markov Decision Processes: Discrete Stochastic Dynamic Programming*'. John Wiley & Sons, 1994 (cited on pages 2, 35).

[PWHA17] E. Polgreen, V. B. Wijesuriya, S. Haesaert and A. Abate. '*Automated Experiment Design for Data-Efficient Verification of Parametric Markov Decision Processes*'. *QEST*. Volume 10503. LNCS. Springer, 2017, pages 259–274 (cited on page 12).

[PXZ02] A. Pnueli, J. Xu and L. D. Zuck. '*Liveness with (0, 1, infty)-Counter Abstraction*'. *CAV*. Volume 2404. LNCS. Springer, 2002, pages 107–122 (cited on page 15).

[PZ86] A. Pnueli and L. D. Zuck. '*Verification of Multiprocess Probabilistic Protocols*'. *Distributed Computing* 1.1 (1986), pages 53–72 (cited on page 15).

[QJDW+15] T. Quatmann, N. Jansen, C. Dehnert, R. Wimmer, E. Ábrahám, J.-P. Katoen and B. Becker. '*Counterexamples for Expected Rewards*'. *FM*. Volume 9109. LNCS. Springer, 2015, pages 435–452 (cited on pages 264, 266, 275).

[QK18] T. Quatmann and J.-P. Katoen. '*Sound Value Iteration*'. *CAV*. Volume 10981. LNCS. Springer, 2018, pages 643–661 (cited on page 230).

[RANL+15] G. N. Rodrigues, V. Alves, V. Nunes, A. Lanna, M. Cordy, P. Schobbens, A. M. Sharifloo and A. Legay. '*Modeling and Verification for Probabilistic Properties in Software Product Lines*'. *HASE*. IEEE Computer Society, 2015, pages 173–180 (cited on pages 12, 13, 197, 199, 200, 284, 296).

[Rei85] W. Reisig. '*Petri Nets: An Introduction*'. Volume 4. EATCS Monographs on Theoretical Computer Science. Springer, 1985 (cited on page 1).

[Ren92] J. Renegar. '*On the Computational Complexity and Geometry of the First-Order Theory of the Reals, Part I: Introduction. Preliminaries. The Geometry of Semi-Algebraic Sets. The Decision Problem for the Existential Theory of the Reals*'. *J. Symb. Comput.* 13.3 (1992), pages 255–300 (cited on page 32).

[RGT05] N. Roy, G. J. Gordon and S. Thrun. '*Finding Approximate POMDP solutions Through Belief Compression*'. *J. Artif. Intell. Res.* 23 (2005), pages 1–40 (cited on page 180).

[RJB99] J. E. Rumbaugh, I. Jacobson and G. Booch. '*The unified modeling language reference manual*'. Addison-Wesley-Longman, 1999 (cited on page 1).

[RN10] S. J. Russell and P. Norvig. '*Artificial Intelligence – A Modern Approach (3. ed.)*' Pearson Education, 2010 (cited on pages 13, 178, 189).

[Ros16] D. S. Rosenblum. '*The power of probabilistic thinking*'. *ASE*. ACM, 2016, page 3 (cited on pages 7, 199).

[RR98] M. K. Reiter and A. D. Rubin. '*Crowds: Anonymity for Web Transactions*'. *ACM Trans. Inf. Syst. Secur.* 1.1 (1998), pages 66–92 (cited on page 6).

[RS14] J. Raskin and O. Sankur. '*Multiple-Environment Markov Decision Processes*'. *FSTTCS*. Volume 29. LIPIcs. Schloss Dagstuhl - LZI, 2014, pages 531–543 (cited on page 16).

[RVWD13] D. M. Roijers, P. Vamplew, S. Whiteson and R. Dazeley. '*A Survey of Multi-Objective Sequential Decision-Making*'. *J. Artif. Intell. Res.* 48 (2013), pages 67–113 (cited on page 12).

[Sak05] J. Sakarovitch. '*The Language, the Expression, and the (Small) Automaton*'. *CIAA*. Volume 3845. LNCS. Springer, 2005, pages 15–30 (cited on page 216).

[SAM17] S. E. Z. Soudjani, A. Abate and R. Majumdar. '*Dynamic Bayesian networks for formal verification of structured stochastic processes*'. *Acta Inf.* 54.2 (2017), pages 217–242 (cited on page 15).

[SB10] J. Skaf and S. Boyd. '*Techniques for exploring the suboptimal set*'. *Optimization and Engineering* 11.2 (2010), pages 319–337 (cited on page 298).

[SB98] R. S. Sutton and A. G. Barto. '*Reinforcement learning - an introduction*'. Adaptive computation and machine learning. MIT Press, 1998 (cited on pages 7, 157).

[SC05] D. Szer and F. Charpillet. '*An Optimal Best-First Search Algorithm for Solving Infinite Horizon DEC-POMDPs*'. *ECML*. Volume 3720. LNCS. Springer, 2005, pages 389–399 (cited on page 178).

[SCFR17] G. Su, T. Chen, Y. Feng and D. S. Rosenblum. '*ProEva: runtime proactive performance evaluation based on continuous-time Markov chains*'. *ICSE*. IEEE, 2017, pages 484–495 (cited on page 7).

[Sch13] M. Schaefer. '*Realizability of Graphs and Linkages*'. *Thirty Essays on Geometric Graph Theory*. Springer New York, 2013, pages 461–482 (cited on pages 150, 245, 332).

[SDM08] A. Sesic, S. Dautovic and V. Malbasa. '*Dynamic Power Management of a System With a Two-Priority Request Queue Using Probabilistic-Model Checking*'. *IEEE Trans. on CAD of Integrated Circuits and Systems* 27.2 (2008), pages 403–407 (cited on page 198).

[SFCR16] G. Su, Y. Feng, T. Chen and D. S. Rosenblum. '*Asymptotic Perturbation Bounds for Probabilistic Model Checking with Empirically Determined Probability Parameters*'. *IEEE Trans. Software Eng.* 42.7 (2016), pages 623–639 (cited on page 78).

[Sha53] L. Shapley. '*Stochastic Games*'. *PNAS* 39.10 (1953), pages 1095–1100 (cited on page 193).

[Sif82] J. Sifakis. '*A Unified Approach for Studying the Properties of Transition Systems*'. *Theor. Comput. Sci.* 18 (1982), pages 227–258 (cited on page 4).

[Sku06] D. Skulj. '*Finite Discrete Time Markov Chains with Interval Probabilities*'. *SMPS*. Volume 37. Advances in Soft Computing. Springer, 2006, pages 299–306 (cited on page 16).

[Sol13] A. Solar-Lezama. '*Program sketching*'. *STTT* 15.5-6 (2013), pages 475–495 (cited on pages 173, 197).

[Son78] E. J. Sondik. '*The Optimal Control of Partially Observable Markov Processes over the Infinite Horizon: Discounted Costs*'. *Operations Research* 26.2 (1978), pages 282–304 (cited on page 178).

[SPK13] G. Shani, J. Pineau and R. Kaplow. '*A survey of point-based POMDP solvers*'. *Autonomous Agents and Multi-Agent Systems* 27.1 (2013), pages 1–51 (cited on pages 178, 180).

[SS17] M. Schaefer and D. Stefankovic. '*Fixed Points, Nash Equilibria, and the Existential Theory of the Reals*'. *Theory Comput. Syst.* 60.2 (2017), pages 172–193 (cited on pages 32, 150, 153, 157, 332–335).

[SST18] S. A. Seshia, N. Sharygina and S. Tripakis. '*Modeling for Verification*'. *Handbook of Model Checking*. Springer, 2018, pages 75–105 (cited on page 1).

[ST05] R. Segala and A. Turrini. '*Comparative Analysis of Bisimulation Relations on Alternating and Non-Alternating Probabilistic Models*'. *QEST*. IEEE Computer Society, 2005, pages 44–53 (cited on page 113).

[STBS+06] A. Solar-Lezama, L. Tancau, R. Bodík, S. A. Seshia and V. A. Saraswat. '*Combinatorial sketching for finite programs*'. *ASPLOS*. ACM, 2006, pages 404–415 (cited on page 259).

[Stu18] T. Sturm. '*Thirty Years of Virtual Substitution: Foundations, Techniques, Applications*'. *ISSAC*. ACM, 2018, pages 11–16 (cited on page 224).

[SV99] M. Stoelinga and F. W. Vaandrager. '*Root Contention in IEEE 1394*'. *ARTS*. Volume 1601. LNCS. Springer, 1999, pages 53–74 (cited on page 6).

[SVA06] K. Sen, M. Viswanathan and G. Agha. '*Model-Checking Markov Chains in the Presence of Uncertainties*'. *TACAS*. Volume 3920. LNCS. Springer, 2006, pages 394–410 (cited on page 16).

[TB17] K. S. Trivedi and A. Bobbio. '*Reliability and Availability Engineering - Modeling, Analysis, and Applications*'. Cambridge University Press, 2017 (cited on page 7).

[TBF05] S. Thrun, W. Burgard and D. Fox. '*Probabilistic Robotics*'. The MIT Press, 2005 (cited on pages 7, 177).

[TPCZ18] A. Tirinzoni, M. Petrik, X. Chen and B. D. Ziebart. '*Policy-Conditioned Uncertainty Sets for Robust Markov Decision Processes*'. *NeurIPS*. 2018, pages 8953–8963 (cited on pages 13, 157).

[Var99] M. Y. Vardi. '*Probabilistic Linear-Time Model Checking: An Overview of the Automata-Theoretic Approach*'. *ARTS*. Volume 1601. LNCS. Springer, 1999, pages 265–276 (cited on page 47).

[VBLL18] A. Vandin, M. H. ter Beek, A. Legay and A. Lluch-Lafuente. '*QFLan: A Tool for the Quantitative Analysis of Highly Reconfigurable Systems*'. *FM*. Volume 10951. LNCS. Springer, 2018, pages 329–337 (cited on pages 13, 197, 319).

[VK13] M. Varshosaz and R. Khosravi. '*Discrete time Markov chain families: modeling and verification of probabilistic software product lines*'. *SPLC Workshops*. ACM, 2013, pages 34–41 (cited on pages 13, 197).

[VKRJ+18] A.-F. Viet, S. Krebs, O. Rat-Aspert, L. Jeanpierre, C. Belloc and P. Ezanno. '*A modelling framework based on MDP to coordinate farmers' disease control decisions at a regional scale*'. *PloS one* 13.6 (2018), e0197612 (cited on page 7).

[VLB12] N. Vlassis, M. L. Littman and D. Barber. '*On the Computational Complexity of Stochastic Controller Optimization in POMDPs*'. *TOCT* 4.4 (2012), 12:1–12:8 (cited on page 178).

[VS02] W. Vesely and M. Stamatelatos. '*Fault Tree Handbook with Aerospace Applications*'. Technical report. NASA Headquarters, USA, 2002 (cited on pages 1, 220).

[VW86] M. Y. Vardi and P. Wolper. '*An Automata-Theoretic Approach to Automatic Program Verification (Preliminary Report)*'. *LICS*. IEEE Computer Society, 1986, pages 332–344 (cited on page 4).

[VWKN19] M. Volk, N. Weik, J.-P. Katoen and N. Nießen. '*A DFT Modeling Approach for Infrastructure Reliability Analysis of Railway Station Areas*'. *FMICS*. Volume 11687. LNCS. Springer, 2019, pages 40–58 (cited on page 7).

[Wei97] V. Weispfenning. '*Quantifier elimination for real algebra-the quadratic case and beyond*'. *Applicable Algebra in Engineering, Communication and Computing* 8.2 (1997), pages 85–101 (cited on page 224).

[WF12] T. Wongpiromsarn and E. Frazzoli. '*Control of probabilistic systems under dynamic, partially known environments with temporal logic specifications*'. *CDC*. IEEE, 2012, pages 7644–7651 (cited on page 177).

[WJÁB+12] R. Wimmer, N. Jansen, E. Ábrahám, B. Becker and J.-P. Katoen. '*Minimal Critical Subsystems for Discrete-Time Markov Models*'. *TACAS*. Volume 7214. LNCS. Springer, 2012, pages 299–314 (cited on pages 263, 264).

[WJÁK+14] R. Wimmer, N. Jansen, E. Ábrahám, J.-P. Katoen and B. Becker. '*Minimal counterexamples for linear-time probabilistic verification*'. *Theor. Comput. Sci.* 549 (2014), pages 61–100 (cited on page 85).

[WJVÁ+15] R. Wimmer, N. Jansen, A. Vorpahl, E. Ábrahám, J.-P. Katoen and B. Becker. '*High-level Counterexamples for Probabilistic Automata*'. *Logical Methods in Computer Science* 11.1 (2015) (cited on pages 271–273).

[WKR13] W. Wiesemann, D. Kuhn and B. Rustem. '*Robust Markov Decision Processes*'. *Math. Oper. Res.* 38.1 (2013), pages 153–183 (cited on pages 13, 157).

[WS17] E. Walraven and M. T. J. Spaan. '*Accelerated Vector Pruning for Optimal POMDP Solvers*'. *AAAI*. AAAI Press, 2017, pages 3672–3678 (cited on pages 178, 188, 189).

[WTM12] E. M. Wolff, U. Topcu and R. M. Murray. '*Robust control of uncertain Markov Decision Processes with temporal logic specifications*'. *CDC*. IEEE, 2012, pages 3372–3379 (cited on pages 13, 157).

[YLWA05] H. L. S. Younes, M. L. Littman, D. Weissman and J. Asmuth. '*The First Probabilistic Track of the International Planning Competition*'. *J. Artif. Intell. Res.* 24 (2005), pages 851–887 (cited on page 6).

[YMZ11] L. Yang, S. Murugesan and J. Zhang. '*Real-Time Scheduling over Markovian Channels: When Partial Observability Meets Hard Deadlines*'. *GLOBECOM*. IEEE, 2011, pages 1–5 (cited on page 220).

[ZMT18] L. D. Zuck, K. L. McMillan and J. Torf. '*P^5 : Planner-less Proofs of Probabilistic Parameterized Protocols*'. *VMCAI*. Volume 10747. LNCS. Springer, 2018, pages 336–357 (cited on page 15).

[ZS06] A. Zadorojniy and A. Shwartz. '*Robustness of policies in constrained Markov decision processes*'. *IEEE Trans. Automat. Contr.* 51.4 (2006), pages 635–638 (cited on pages 13, 157).

Co-authored Publications

A statement regarding my contributions and the origins of the chapters may be found in Sect. 1.4.

Cited publications

[1] S. Junges, U. Loup, F. Corzilius and E. Ábrahám. '*On Gröbner Bases in the Context of Satisfiability-Modulo-Theories Solving over the Real Numbers*'. *CAI*. Volume 8080. LNCS. Springer, 2013, pages 186–198 (cited on page 224).

[2] F. Corzilius, G. Kremer, S. Junges, S. Schupp and E. Ábrahám. '*SMT-RAT: An Open Source C++ Toolbox for Strategic and Parallel SMT Solving*'. *SAT*. Volume 9340. LNCS. Springer, 2015, pages 360–368 (cited on pages 223, 320).

[3] C. Dehnert, S. Junges, N. Jansen, F. Corzilius, M. Volk, H. Bruintjes, J.-P. Katoen and E. Ábrahám. '*PROPhESY: A PRObabilistic ParamEter SYnthesis Tool*'. *CAV*. Volume 9206. LNCS. Springer, 2015, pages 214–231 (cited on pages 20, 219, 232, 237).

[4] C. Dombrowski, S. Junges, J.-P. Katoen and J. Gross. '*Model-Checking Assisted Protocol Design for Ultra-reliable Low-Latency Wireless Networks*'. *SRDS*. IEEE Computer Society, 2016, pages 307–316 (cited on page 6).

[5] T. Quatmann, C. Dehnert, N. Jansen, S. Junges and J.-P. Katoen. '*Parameter Synthesis for Markov Models: Faster Than Ever*'. *ATVA*. Volume 9938. LNCS. Springer, 2016, pages 50–67 (cited on pages 20, 301, 313, 314).

[6] M. Volk, S. Junges and J.-P. Katoen. '*Advancing Dynamic Fault Tree Analysis - Get Succinct State Spaces Fast and Synthesise Failure Rates*'. *SAFECOMP*. Volume 9922. LNCS. Springer, 2016, pages 253–265 (cited on page 220).

[7] C. E. Budde, C. Dehnert, E. M. Hahn, A. Hartmanns, S. Junges and A. Turrini. '*JANI: Quantitative Model and Tool Interaction*'. *TACAS*. Volume 10206. LNCS. 2017, pages 151–168 (cited on page 56).

[8] M. Cubuktepe, N. Jansen, S. Junges, J.-P. Katoen, I. Papusha, H. A. Poonawala and U. Topcu. '*Sequential Convex Programming for the Efficient Verification of Parametric MDPs*'. *TACAS*. Volume 10206. LNCS. Springer, 2017, pages 133–150 (cited on pages 21, 254).

[9] C. Dehnert, S. Junges, J.-P. Katoen and M. Volk. '*A storm is Coming: A Modern Probabilistic Model Checker*'. *CAV*. Volume 10427. LNCS. Springer, 2017 (cited on pages 5, 189, 254, 277, 294, 319).

[10] L. Winterer, S. Junges, R. Wimmer, N. Jansen, U. Topcu, J.-P. Katoen and B. Becker. '*Motion planning under partial observability using game-based abstraction*'. *CDC*. IEEE, 2017, pages 2201–2208 (cited on pages 21, 284).

[11] M. Cubuktepe, N. Jansen, S. Junges, J.-P. Katoen and U. Topcu. '*Synthesis in pMDPs: A Tale of 1001 Parameters*'. *ATVA*. Volume 11138. LNCS. Springer, 2018, pages 160–176 (cited on pages 12, 21, 243, 251, 314).

[12] S. Junges, N. Jansen, R. Wimmer, T. Quatmann, L. Winterer, J.-P. Katoen and B. Becker. '*Finite-State Controllers of POMDPs using Parameter Synthesis*'. *UAI*. AUAI Press, 2018, pages 519–529 (cited on pages 21, 177, 188).

[13] M. Ceska, C. Hensel, N. Jansen, S. Junges and J.-P. Katoen. '*Model Repair Revamped*'. *From Reactive Systems to Cyber-Physical Systems*. Volume 11500. LNCS. Springer, 2019, pages 107–125 (cited on pages 22, 177, 259, 283).

[14] M. Ceska, C. Hensel, S. Junges and J.-P. Katoen. '*Counterexample-Driven Synthesis for Probabilistic Program Sketches*'. *FM*. Volume 11800. LNCS. Springer, 2019, pages 101–120 (cited on pages 22, 159, 164, 177, 259).

[15] M. Ceska, N. Jansen, S. Junges and J.-P. Katoen. '*Shepherding Hordes of Markov Chains*'. *TACAS*. Volume 11428. LNCS. Springer, 2019, pages 172–190 (cited on pages 22, 159, 164, 172, 283).

[16] M. Ghadhab, S. Junges, J. Katoen, M. Kuntz and M. Volk. '*Safety analysis for vehicle guidance systems with dynamic fault trees*'. *Rel. Eng. & Sys. Safety* 186 (2019), pages 37–50 (cited on page 7).

[17] S. Junges, E. Ábrahám, C. Hensel, N. Jansen, J.-P. Katoen, T. Quatmann and M. Volk. '*Parameter Synthesis for Markov Models*'. *CoRR* abs/1903.07993 (2019). under review (cited on pages 11, 21, 25, 61, 207, 219, 222, 301, 305, 319).

[18] J. Spel, S. Junges and J.-P. Katoen. '*Are Parametric Markov Chains Monotonic?*' *ATVA*. Volume 11781. LNCS. Springer, 2019, pages 479–496 (cited on pages 78, 229).

[19] T. Winkler, S. Junges, G. A. Pérez and J.-P. Katoen. '*On the Complexity of Reachability in Parametric Markov Decision Processes*'. *CONCUR*. Volume 140. LIPIcs. Schloss Dagstuhl - LZI, 2019, 14:1–14:17 (cited on pages 21, 25, 61, 105, 107, 132, 150, 157, 177, 196).

[20] C. Baier, C. Hensel, L. Hutschenreiter, S. Junges, J.-P. Katoen and J. Klein. '*Parametric Markov Chains: PCTL Complexity and Fraction-free Gaussian Elimination*'. *Inf. Comp.* (2020). to appear (cited on pages 21, 105, 126, 155, 207, 213, 219).

Further peer-reviewed publications

F. Corzilius, U. Loup, S. Junges and E. Ábrahám. '*SMT-RAT: An SMT-Compliant Nonlinear Real Arithmetic Toolbox - (Tool Presentation)*'. *SAT*. Volume 7317. LNCS. Springer, 2012, pages 442–448.

D. Bohlender, H. Bruintjes, S. Junges, J. Katelaan, V. Y. Nguyen and T. Noll. '*A Review of Statistical Model Checking Pitfalls on Real-Time Stochastic Models*'. *ISoLA*. Volume 8803. LNCS. Springer, 2014, pages 177–192.

S. Junges, D. Guck, J.-P. Katoen, A. Rensink and M. Stoelinga. '*Fault Trees on a Diet - - Automated Reduction by Graph Rewriting -*'. *SETTA*. Volume 9409. LNCS. Springer, 2015, pages 3–18.

C. Dehnert, S. Junges, N. Jansen, F. Corzilius, M. Volk, J.-P. Katoen, E. Ábrahám and H. Bruintjes. '*Parameter Synthesis for Probabilistic Systems*'. *MBMV*. Albert-Ludwigs-Universität Freiburg, 2016, pages 72–74.

S. Junges, D. Guck, J.-P. Katoen and M. Stoelinga. '*Uncovering Dynamic Fault Trees*'. *DSN*. IEEE Computer Society, 2016, pages 299–310.

S. Junges, N. Jansen, C. Dehnert, U. Topcu and J.-P. Katoen. '*Safety-Constrained Reinforcement Learning for MDPs*'. *TACAS*. Volume 9636. LNCS. Springer, 2016, pages 130–146.

S. Junges, N. Jansen, J.-P. Katoen and U. Topcu. '*Probabilistic Verification for Cognitive Models*'. AAAI Technical Reports FS-16. AAAI Press, 2016.

M. Ghadhab, S. Junges, J.-P. Katoen, M. Kuntz and M. Volk. '*Model-Based Safety Analysis for Vehicle Guidance Systems*'. *SAFECOMP*. Volume 10488. LNCS. Springer, 2017, pages 3–19.

S. Junges, D. Guck, J.-P. Katoen, A. Rensink and M. Stoelinga. '*Fault trees on a diet: automated reduction by graph rewriting*'. *Formal Asp. Comput.* 29.4 (2017), pages 651–703.

T. Quatmann, S. Junges and J.-P. Katoen. '*Markov Automata with Multiple Objectives*'. *CAV*. Volume 10426. LNCS. Springer, 2017.

A. Hartmanns, S. Junges, J.-P. Katoen and T. Quatmann. '*Multi-cost Bounded Reachability in MDP*'. *TACAS*. Volume 10806. LNCS. Springer, 2018, pages 320–339.

S. Junges, N. Jansen, J.-P. Katoen, U. Topcu, R. Zhang and M. M. Hayhoe. '*Model Checking for Safe Navigation Among Humans*'. *QEST*. Volume 11024. LNCS. Springer, 2018, pages 207–222.

S. Junges, J.-P. Katoen, M. Stoelinga and M. Volk. '*One Net Fits All - A Unifying Semantics of Dynamic Fault Trees Using GSPNs*'. *Petri Nets*. Volume 10877. LNCS. Springer, 2018, pages 272–293.

M. Volk, S. Junges and J.-P. Katoen. '*Fast Dynamic Fault Tree Analysis by Model Checking Techniques*'. *IEEE Trans. Industrial Informatics* 14.1 (2018), pages 370–379.

M. Cubuktepe, N. Jansen, S. Junges, J.-P. Katoen and U. Topcu. '*Scenario-Based Verification of Uncertain MDPs*'. *TACAS*. to appear. 2020.

Index

3SAT, 32

A
abstraction-refinement, 242
all-in-one MDP, 284
assignment
    accepting, 170
    fMC, 162
    rejecting, 170

B
bad states, 41
belief MDP, 178
BSCC, 39

C
CAD, 224
candidate hypothesis, 232
candidate region, 231
CCP, 247
CEGIS, 241, 259
command
    relevant, 275
complete partitioning, 77
conflict, 262
    trivial, 262
conflict set, 262
constraint, 27
constraint MC, 16
constraint system, 29
convex polytope, 30
counterexample, 264
    high-level, 273
    program-level, 272
critical states, 263
CSRG, 193
    corresponding pMDP, 196
    induced MDP, 194
    play, 194
    round, 194
    strategy, 194
    value, 195

D
DC, 248
DIR, 222

E
ETR, 32
exist-one states, 93
exist-zero states, 51, 93
expected reward, 44
expected steps, 44

F
family, 162
    accepting, 170
    rejecting, 170
family verification, 170
feasibility, 9
    encoding, 222
    fMC, 170
    sampling, 227
    using convex optimisation, 243

finite optimal reward, 48
fMC, 162
    from pMC, 167
    path, 165
    simplex, 165
    sub-fMC, 263
    to pMC, 166
forall-one states, 93
forall-zero states, 51, 93
forgetting, 286
formula, 28
    equisatisfiable, 29
    equivalent, 29
    propositional, 29
FSC, 178, 185
    action mapping, 185
    control update, 185

G
Gaussian elimination, 208
    fraction-free, 210
    one-step fraction-free, 210
generator, 66
    fMC, 162
Gröbner bases, 224
grouped parameters, 110

H
holes, 173
horizon, 42

I
ICP, 224
instantiation, 66
    fMC, 162
    sketch, 174
interval MC, 16, 117

J
Jani, 56

L
lifting, 305
literals, 29
LP, 31

M
matrix
    PSD, 329
    transpose, 329
maximal feasibility, 292
MaxSat, 31
MC, 2, 36
    subMC, 263
MDP, 2, 35
    available actions, 36
    equivalence, 50
    induced MC, 40
    underlying graph, 38
minimising strategy set, 138
Modest, 56
monomial, 25
    linear, 26
    multi-linear, 26
    quadratic, 26
MOSS, 138
multivariate, 26

N
nondeterminism, 36

O
observation function, 180
observations, 178
occurs
    fMC, 165
    pMDP, 67
one-states, 51, 82
optimal feasibility, 75, 170
option, 173
OSS, 137

P
parameter lifting, 242
parameter space, 9
parameter space partitioning, 230
parameter-cost function, 76, 170
partial assignment, 262
partial partitioning, 77
partitioning, 171, 291
path, 37
    first state, 37
    last state, 37
    length, 37
    mass, 42
    visits, 37
pCTMC, 15
permissive valuations, 75
PGCL, 56
pMC, 8, 64
    corresponding POMDP, 184
    locally monotone, 303
    substitution, 306
pMDP, 8, 64
    action-univariate, 109
    affine, 108
    alt, 113
    alternating, 112
    constant-reward, 106
    dependency-free, 117
    expanded simplex, 115
    multivariate, 109
    ntp, 116
    polynomial, 78
    realisable, 66
    simple, 112, 114
    univariate, 109
polynomial, 26
    adequate, 123
    bilinear, 26
    convex, 27
    gradient, 27
    Hessian, 27
    linear, 26
    quadratic, 26
    root, 27
    univariate, 26
POMDP, 13, 180
    corresponding pMC, 182
    underlying MDP, 180
    unfolding, 187
predecessors, 38
PRISM, 56
probabilistic model checking, 5
probability
    has, 42
program, 56
    command, 56
    fixdl, 56
    guard, 56
    module, 56
    update, 56
    variable, 56
property, 4
    bounded reachability, 46
    complement, 46
    expected reward, 46
    liveness, 46
    lower bounded, 46
    safety, 46
    unbounded reachability, 45
    upper bounded, 46
PSO, 227

Q
QCQP, 245
quant-states, 51
quotient MDP, 286

R
rational function, 26
reachability
    almost sure, 140
    positive, 140

unsure, 140
reachability probability, 42
bounded, 42
reachable states, 38
realisability, 107
region, 9, 70
affine, 72
graph-consistent, 118
inconsistent, 69
well-defined, 79
region verification, 9, 75
encoding, 222
relaxation, 304
restriction, 291
rewards, 35

S

sample, 227, 234
satisfaction relation, 45
SG, 55
satisfying valuation, 29
satisfying valuations, 28
SCC, 39
self-loop, 36
semi-algebraic set, 30
set
closed, 330
closure, 330
convex, 330
interior, 330
open, 330
SF, 222
SG, 54
sketch, 173
realisation, 174
sketching, 197
SMT, 32
solution, 29
solution function, 67
computation, 207
specification, 46
SPL, 13, 197
split, 291
state
absorbing, 39
nondeterministic, 36
state elimination, 213
for expected rewards, 216
using BDDs, 217
strategy, 39
consistent, 288
deterministic, 39
memoryless, 39
observation-based, 181
optimal, 48
robust, 13, 157
SG, 55
somewhere optimal, 135
strongly connected, 39
substitution MDP, 306
substitution SG, 310
successors, 38
support, 329
synthesiser, 260

T

target states, 41
term, 26
threshold, 45

U

underlying MC, 56
unknown region, 77
unreachability, 140

V

validity
fMC, 170
valuation, 26, 65
accepting, 69
border, 233
graph-preserving, 73

ε-preserving, 120
pure, 120
rejecting, 69
well-defined, 65
vanishing, 73
variable, 25
assignment, 26
occurs, 26
virtual substitution, 224

Y
y-assignment-consistency, 288

Z
zero-states, 50, 82