Counterexamples in Probabilistic Verification

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

Diplom-Informatiker

Nils Jansen

aus Simmerath

Berichter	Prof. Dr. Erika Ábrahám
	Prof. Dr. Bernd Becker
	Prof. Dr. Ir. Joost-Pieter Katoen
Tag der mündlichen Prüfung	26.03.2015

Abstract

The topic of this thesis is roughly to be classified into the *formal verification* of *probabilistic systems*. In particular, the generation of *counterexamples* for *discrete-time Markov Models* is investigated. A counterexample for discrete-time Markov Chains (DTMCs) is classically defined as a (finite) set of paths. In this work, this set of paths is represented *symbolically* as a critical part of the original system, a so-called *critical subsystem*. This notion is extended to Markov decision processes (MDPs) and probabilistic automata (PAs). The results are introduced in four parts:

1. *A model checking algorithm for DTMCs based on a decomposition of the system's graph in strongly connected components (SCCs).* This approach is extended to parametric discrete-time Markov Chains.

2. *The generation of counterexamples for DTMCs and reachability properties based on graph algorithms.* A *hierarchical abstraction scheme* to compute abstract counterexamples is presented, followed by a general framework for both *explicitly represented* systems and *symbolically* represented systems using binary decision diagrams (BDDs).

3. *The computation of minimal critical subsystems using SAT modulo theories (SMT) solving and mixed integer linear programming (MILP).* This is investigated for reachability properties and ω-regular properties on DTMCs, MDPs, and PAs.

4. *A new concept of high-level counterexamples for PAs.* Markov models can be specified by means of a probabilistic programming language. An approach for computing critical parts of such a symbolic description of a system is presented, yielding human-readable counterexamples.

All results have been published in conference proceedings or journals. A thorough evaluation on common benchmarks is given comparing all methods and also competing with available implementations of other approaches.

Zusammenfassung

Das Thema dieser Doktorarbeit kann grob in die *formale Verifikation probabilistischer Systeme* eingeordnet werden. Genauer gesagt wird die Generierung von *Gegenbeispielen* für *Markow-Modelle mit diskreter Zeit* untersucht. Ein Gegenbeispiel für Markow-Ketten mit diskreter Zeit (DTMCs) ist ursprünglich als eine (endliche) Menge von Pfaden definiert. In dieser Arbeit repräsentieren wir diese Menge von Pfaden *symbolisch* durch einen kritischen Teil des originalen Systems, ein sogenanntes *kritisches Teilsystem*. Dieses Konzept wird erweitert auf Markow-Entscheidungsprozesse (MDPs) und probabilistische Automaten (PAs). Die Resultate werden unterteilt in vier Abschnitte vorgestellt:

1. *Ein Algorithmus zur Modellüberprüfung für DTMCs basierend auf der Zerlegung des Systemgraphen in starke Zusammenhangskomponenten.* Dieser Ansatz wird auf parametrische Markow-Ketten mit diskreter Zeit erweitert.

2. *Die Generierung von Gegenbeispielen für DTMCs und Erreichbarkeitsbedingungen basierend auf Graphalgorithmen.* Ein *hierarchisches Abstraktionsschema* zur Berechnung abstrakter Gegenbeispiele wird präsentiert, gefolgt von einem generellen Rahmenwerk für sowohl *explizit repräsentierte* Systeme als auch *symbolisch repräsentierte* Systeme unter Verwendung von binären Entscheidungsdiagrammen (BDDs).

3. *Die Berechnung minimaler kritischer Teilsysteme unter Nutzung von Erfüllbarkeitsüberpüfung erweitert um Theorien (SMT) und gemischt ganzzahlig-linearer Optimierung (MILP).* Dies wird untersucht für Erreichbarkeitsbedingungen und ω-reguläre Eigenschaften für DTMCs, MDPs und PAs.

4. *Ein neues Konzept für high-level Gegenbeispiele für PAs.* Markow-Modelle können mit Hilfe einer probabilistischen Programmiersprache spezifiziert werden. Ein Ansatz zur Berechnung kritischer Teile solch einer symbolischen Systembeschreibung wird vorgestellt, resultierend in für Menschen lesbaren Gegenbeispielen.

Alle Resultate wurden in Konferenzbänden oder Fachzeitschriften publiziert. Eine ausführliche Evaluierung anhand bekannter Benchmarks vergleicht alle Methoden untereinander und mit verfügbaren Implementierungen anderer Ansätze.

Tomorrow may be hell,
but today was a good writing day,
and on the good writing days nothing else matters.

NEIL GAIMAN

Acknowledgements

Now that finally my thesis is finished, my first and deepest thanks go to my supervisor Erika Ábrahám for giving me the opportunity to do my PhD in Aachen, for introducing me to this interesting field of research, for teaching me a formal way of thinking, and last but not least for all the great night sessions having wine when the paper deadlines approached. The first years with the Theory of Hybrid Systems group were amazing!

I also want to thank my second supervisor, Joost-Pieter Katoen for all the input, help, and discussions during my PhD, and for giving me the opportunity to continue doing research in Aachen in the perfect environment of the MOVES group.

Special thanks go to my friend and colleague Ralf Wimmer from Freiburg for all the very productive discussions, help, and for the awesome travels, to—only to mention a few—Argentina, Estonia, or the USA. He was a great support and guidance right from the start. To be continued! I am also much obliged to Bernd Becker from Freiburg, who not only strongly supported the joint work of Aachen and Freiburg and who guided many discussions to the right directions with his experience, but who also was a great host every time I have been to Freiburg.

I am very grateful for the years of working with the team Erika Ábrahám assembled to the THS group. It was an enjoyable experience to help initiating the group together with Xin Chen, Florian Corzilius, Ulrich Loup, Johanna Nellen, and later on with Stefan Schupp and Gereon Kremer. Furthermore, big thanks go to my team of student assistants, especially Matthias Volk, Andreas Vorpahl, Barna Zajzon, Maik Scheffler, Jens Katelaan, and Tim Quatmann for the great work on our projects and for lots of fun. Special thanks are in order for Matthias and Barni who helped me a lot conducting the final experiments for this thesis. Moreover, big thanks go to the guys who helped me correct many errors, namely Ralf Wimmer, Dennis Komm, Jörg Olschewski, Christian Dehnert, and of course Erika Ábrahám, and to Bjorn Feldmann who designed the nice cover of this thesis.

I enjoyed working with my co-authors and collaboration partners Erika Ábrahám, Bernd Becker, Bettina Braitling, Harold Bruintjes, Florian Corzilius, Christian Dehnert, Friedrich Gretz, Sebastian Junges, Benjamin Kaminski, Jens Katelaan, Joost-Pieter Katoen, Daniel Neider, Federico Olmedo, Shashank Pathak, Tim Quatmann, Johann Schuster, Armando Tacchella, Matthias Volk, Andreas Vorpahl, Ralf Wimmer, and Barna Zajzon. This is a personal "Thanks, and let's do more!"

My work was supported by the German Research Council (DFG) as part of the DFG project "CEBug" (AB 461/1-1) and the Research Training Group "AlgoSyn" (1298) and by the Excellence Initiative of the German federal and state government.

One of the things I enjoyed most about doing a PhD was the traveling to many conferences and meetings at awesome places all over the world. Saying this, I want to thank Pedro d'Argenio for being such a good host in Córdoba several times.

I want to thank Alexa for all the support over the last years. Finally, I am always deeply grateful to have my parents and my friends.

This thesis is dedicated to my grandfather.

Contents

CHAPTER 1

Introduction

Over the last decades, the number of computer-controlled systems has been growing rapidly. Nowadays, in almost every aspect of our lives we find some sort of complex computer system, often in safety critical scenarios of essential importance such as airbags, autopilots, or train scheduling, only to mention a few examples from the transport area. Even a very small mistake in the underlying software can lead to disastrous consequences, consider, e. g., the crash of the Ariane-5 rocket, which happened in 1996 due to an unintended floating point conversion. Several popular examples report on very serious events resulting from errors like this; ranging from mere nuisances over financial breakdowns to live-threatening disasters.

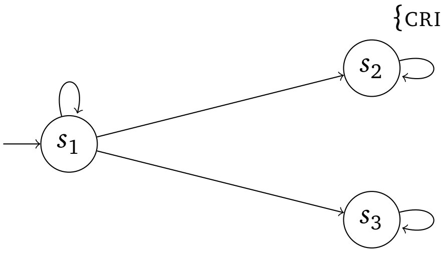
Although it must be ensured that systems work *correctly* in a sense that no undesired behavior occurs, it is often not feasible to apply test-runs to a satisfiable extent. This might for instance be due to financial reasons or tight time schedules. It is therefore of utmost importance to ensure a certain *safety* already during the early stage of system development. One possibility is to *simulate* a computer-based representation, a *model* of the system, for different scenarios. While this might be a reasonable and economic way to rule out many errors, in theory, one would have to do this for *every possible* scenario, i. e., for every possible input the model could be facing, in order to rule out any possible error.

An important discipline in computer science is the *formal verification* of systems, where certain properties can be *proven*. In the mid of the nineties, a new branch called *model checking* was established [JGP99]. Having a reliable model of the system at hand together with a formal specification of desired or undesired behavior, one can *prove* that the desired behavior occurs for all possible scenarios or—in other words—that the undesired behavior can be excluded. For a thorough introduction we refer to [BK08].

If an error is revealed, it is desirable to get some sort of *diagnostic information* in order to trace the problem. This information is called a *counterexample*. This might, e. g., be an erroneous *run* of the system. A famous quotation from 2008 by one of the founders of the concept of model checking, Edmund Clarke, stresses the importance of counterexamples:

> *"It is impossible to overestimate the importance of the counterexample feature. The counterexamples are invaluable in debugging complex systems. Some people use model checking just for this feature. [Cla08]"*

To give a first intuition, consider a so-called *Kripke structure* [Kri63] which is a simple model to represent the behavior of a system. Intuitively, the states of the system are modeled by nodes of a graph. Possibilities to move between the states are indicated by transitions. Moreover, every state has a certain set of properties. A toy example can be found below, where from the initial state s_1 transitions lead to the states s_2 and s_3. Additionally, every state is equipped with a self-loop. The state s_2 is labeled with the property CRITICAL indicating an undesired behavior.



The specification we want to investigate is that a critical state is never reached; formally we have the LTL formula $\neg\Diamond$CRITICAL, i. e., "it is not possible to finally reach a state which is labeled with CRITICAL". This property is clearly violated by the path leading from s_1 to s_2, which also serves as a counterexample for this property.

Classic model checking is based on rigorous exploration of the state space, which leads to serious problems considering the large size of models for real-world scenarios. This is often referred to as the *state space explosion problem*. Several approaches were developed to overcome this problem. An important one is *symbolic model checking* which was introduced in 1992 [BCM$^+$92] and which is based on a *symbolic representation* of the state space, e. g., by *binary decision diagrams* (BDDs) [Bry86]. Dedicated algorithms for these symbolic representations were complemented by SAT-based *bounded model checking* [CBRZ01], where a SAT solver is used to prove or disprove properties by means of bounded system runs. Another technique is the *counterexample-guided abstraction refinement* framework (CEGAR) [CGJ$^+$00]. Here, verification is performed on abstract systems which—if the abstraction is too coarse—may be refined with the help of counterexamples that are *spurious*, which means that they do not form a counterexample for the original system. This procedure goes on until a counterexample is found that is not spurious or until the property can be proven to be true.

Besides being a guide in *debugging* a system, counterexamples play an important role in automated verification techniques. As mentioned before, the key ingredient for CEGAR approaches, besides a feasible abstraction, are counterexamples. Another application is in *model-based testing* [FWA09]. A system is tested via a model, a so-called *blueprint* of the system. Counterexamples which are obtained during this procedure can be used to correct the original system.

Amongst others, these applications, which are of great relevance in practical settings, led to active research on the *generation* and the *representation* of counterexamples [GMZ04, CGMZ95, BP12, SB05]. As illustrated by the example above, if a system is simple enough to be modeled sufficiently accurate as a Kripke structure, and if the violated specification is given as a *linear-time property*, a counterexample is given by a *path* through the Kripke structure inducing the violation. These counterexamples can be generated *on-the-fly* during the model checking procedure. For CTL properties, the representation of counterexamples requires a more complex tree-like structure [CJLV02]. For further information, we refer to [CV03].

All the above-mentioned methods were mainly developed for mere *digital circuits*. The real-world behavior of many processes is inherently *stochastic*, take for instance randomized algorithms, fault-tolerant systems, or communication networks where certain aspects can only be captured via probabilities. For a network protocol, a useful property would be "The probability of having a message delivered is at least 99.9%." Referring to a benchmark we use in this thesis, assume a *crowd* of nodes in a network [RR98], where each member has a probabilistic choice of delivering a message or routing it to another—again randomly determined—node of the network in order to establish anonymity. Assuming "bad" members of the crowd that want to identify the sender of the message, a question would then be "What is the probability that the sender of a message is identified by a bad member?".

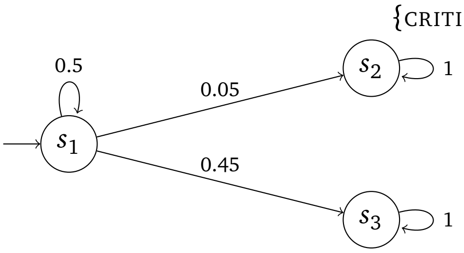
Answering such *quantitative* questions is the comprehensive topic of this thesis. *Probabilistic model checking* summarizes techniques to analyze systems where transitions are augmented with probabilities. *Discrete-time Markov chains* (*DTMCs*) are a popular model to represent probabilistic behavior. Their invention can be traced back to the Russian mathematician *Andrey Andreyevich Markov* in 1906. For standard textbooks we refer to [KS69, Kul95].

In a nutshell, DTMCs are Kripke structures whose transitions induce a discrete probability distribution over successor states. For DTMCs, properties like "The probability of reaching a critical state is at most 10^{-5}" are considered. *Markov decision processes* (*MDPs*) and *probabilistic automata* (*PAs*) add the feature of a nondeterministic choice of probability distributions to DTMCs. To verify properties, a *scheduler* is needed to resolve this nondeterminism yielding a DTMC. Such a scheduler then induces a probability measure on this DTMC, for instance the *maximal probability* for a certain property. We formulate properties like "What is the maximal probability of reaching a critical state"? Standard approaches reduce model checking for DTMCs to solving a linear equation system, while for MDPs an optimization problem is used to compute maximal or minimal probabilities depending on the scheduler, e. g., using linear programming. A detailed description of the basic techniques can be found in [BK08]. For an overview of the state-of-the-art we refer to [Kat13, KNP07].

As for other models, the state space explosion is also a problem for probabilistic systems. Much effort has been put into adapting well-known concepts to this setting. The usage of symbolic data structures via BDDs and *multi-terminal binary decision diagrams* (MTBDDs) [FMY97] was proposed [BCH$^+$97]. For an overview of efficient algorithms and a discussion about hybrid approaches regarding a mixture between symbolic and explicit approaches we refer to [Par02]. For many large or even infinite systems, these methods do still not work. Therefore, one research branch is to develop dedicated abstraction techniques such as probabilistic CEGAR [HWZ08, CV10] or game-based abstraction [KKNP10, Wac10].

The most prominent probabilistic model checker available for the largest variety of models and properties is `PRISM` [KNP11] developed at the University of Oxford, UK. A very competitive tool is `MRMC` [KZH$^+$11] which has been developed at RWTH Aachen University, Germany. For a comparison, see [JKO$^+$07]. Moreover, `PRISM` offers a large selection of benchmarks [KNP12]; some of them are used throughout this thesis.

The generation of counterexamples for probabilistic systems is the main focus of this thesis. Consider the DTMC which is depicted below. With probability 0.05, the successor of the initial state s_1 is s_2, and with probability 0.45 the successor is s_3. With probability 0.5, the successor is s_1 itself. The states s_2 and s_3 have self-loops with probability 1 which means that once an execution enters one of those states it will stay there *almost surely*. For this DTMC, the overall probability to reach the critical state s_2 is 0.1. The underlying computations are discussed later. Intuitively, the probability of *all paths* leading to s_2 has to be considered.



Now consider the quantitative property “The probability of reaching the critical state is less than or equal to 0.05”. This property is *violated*, as the actual probability is equal to 0.1. Forming a counterexample is not as trivial as for the preceding Kripke structure. The path leading from s_1 to s_2 has probability 0.05. This path itself does not violate the property. However, the path looping one time on state s_1 and then taking the transition to s_2 has probability $0.5 \cdot 0.05 = 0.025$. The two paths together have a probability mass of 0.075 which is larger than the allowed probability of 0.05. The set of these two paths serves as a counterexample to the property.

The first publications on the generation of counterexamples for DTMCs were [AHL05, AL06, HK07a, DHK08, ADvR08, HKD09, AL10]. The related work is discussed in the corresponding chapter. Let us, however, give a short intuition on the approach that was made in [HK07a, HKD09]: As mentioned before, a counterexample is a set of paths whose joint probability mass exceeds a certain probability threshold. These paths are *evidences* for a certain property, e. g., for reaching a target state. As only paths describing stochastically independent events are considered;

their probabilities can just be added when computing the probability of a whole set of evidences. In [HK07a, HKD09], graph algorithms are utilized to compute a *minimal* counterexample. In particular, the probabilities are altered such that the *most probable* paths now correspond to the *shortest* paths when adding the altered values. Then, a *k-shortest path search* [Epp98] yields the k most probable paths leading to a target state. As soon as the probability mass of the thereby computed paths is large enough, the search terminates yielding a counterexample that is minimal in terms of the number of paths. Note that the value of k is determined on-the-fly such that the search continues until enough probability mass is accumulated.

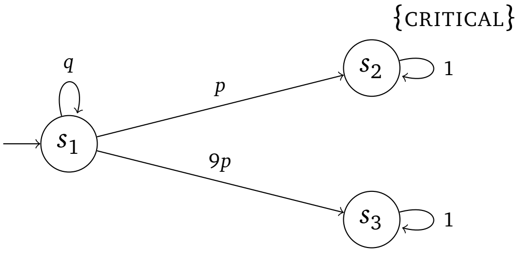
Consider again the DTMC depicted above and the property "The probability of reaching a critical state is less than 0.1". The first three paths leading from s_1 to s_2 in descending order of their probabilities are:

$\pi_1 = s_1, s_2$	probability: 0.05
$\pi_2 = s_1, s_1, s_2$	probability: 0.025
$\pi_3 = s_1, s_1, s_1, s_2$	probability: 0.00125

The probability mass of these three paths is 0.07625, which is still too small to induce a counterexample. In fact, to reach the bound of 0.1, *all* paths leading from s_1 to s_2 need to be considered, which are infinitely many. The shortest path algorithm as described above would therefore not terminate, while errors might occur as the probabilities of the paths become infinitesimally small. This simple example illustrates one major drawback of counterexamples that are represented as sets of paths: Many paths are similar except for their number of loop-iterations, which calls for a special treatment of loops.

Although the generation of counterexamples covers the most important part of this thesis, we have a short excursion to another research direction. Consider systems where no certain probabilities are fixed. At an early design stage, this is a realistic assumption. Determining appropriate probabilities is an own research field called *fitting* [SR13, TBT06]: In model-based performance analysis, probability distributions are generated from experimental measurements.

Therefore, probabilities are handled as *parameters* yielding *parametric* discrete-time Markov chains (*PDTMCs*). Transition probabilities are then interpreted as functions in the system's parameters. Using these functions, one can, e. g., find appropriate values of the parameters such that certain properties are satisfied, or analyze the sensitivity of reachability probabilities under small changes in the parameters. Standard methods for DTMC model checking such as solving a linear equation system are not feasible for PDTMCs, since the resulting equation system would be non-linear. Consider a modification of the DTMC shown above, where the probability to reach the critical state s_2 is now described by a parameter $p \in [0, 1]$. Going from s_1 to the uncritical state s_3 has probability $9p$ indicating that the probability of going there should be much higher than going to s_2, which happens with probability p. The self-loop on s_1 is described by $q \in [0, 1]$ which again has implicit dependencies to p, if well-defined probability distributions are considered.



Instantiating p with 0.05 yields that the probability of going from s_1 to s_3 is 0.45 as in the DTMC above. Furthermore, as outgoing probabilities for each state have to sum up to 1 we know that $p + 9p + q = 1$ yielding $q = 0.5$. What we already see in this simple example is that there are certain additional constraints for parameter evaluations that need to be taken care of. These complications demand dedicated model checking procedures, the only one publicly available so far being implemented in PARAM [HHWZ10].

1.1 Contributions and structure of this thesis

In this section, we describe our approaches that are presented in this thesis and point to the corresponding chapters. Apart from the contributions, we present detailed foundations needed throughout this work in Chapter 2. The related work that has been done before or during the period of this thesis is listed on an intuitive level in Chapter 3. After the theoretical approaches we describe our implementations and tools followed by an experimental evaluation in Chapter 8. The thesis concludes with a short summary and an outlook to future work in Chapter 9.

SCC-based model checking As we have indicated before, benchmarks with a complex loop structure pose a difficult problem for probabilistic counterexample generation. Many paths have to be collected that are similar to each other except for the number of iterations of the same loops. In order to exploit this fact, we developed an abstraction scheme for DTMCs based on the *strongly connected components* (*SCCs*) of the underlying directed graph of an input DTMC. First, the SCCs are determined. By heuristically ignoring those states of an SCC to which there is a transition from outside the SCC, it is decomposed into further SCCs. In the context of the original graph we call these SCCs *sub-SCCs*. This process is iterated until only single states remain. By a subsequent bottom-up computation, the probability of reaching states outside each sub-SCC from its formerly ignored states is computed, while the sub-SCCs are replaced by abstract states. In the end, this gives the probability of reaching the set of target states from the unique initial state. This method was published in [11].

An extension of this work was applied to PDTMCs yielding rational functions that describe reachability probabilities for each sub-SCC. As this induces very large functions even in the early computation steps, they need to be cancelled in each step. This involves the costly computation of the greatest common divisor (GCD) for two polynomials and is therefore not feasible for large benchmarks. However, we were able to partly overcome this problem by developing a new way to compute the GCD on *factorizations* of the given polynomials. These various methods which were published in [2], improving the computation times in comparison with another approach [HHWZ10] by several orders of magnitude for most of the benchmarks.

The approaches are explained in detail in Chapter 4. Summarized, the main contributions are:

1. A novel *model checking method* for DTMCs that—even in its prototype implementation—is competitive to the well-established tools MRMC [KZH$^+$11] and PRISM [KNP11] on many benchmarks having up to one million states.

2. A *hierarchical abstraction scheme* for DTMCs with respect to their SCC structure. As one of the major benefits, this offers the possibility to search for counterexamples on an abstract graph while each of the abstract states can be refined upon request.

3. A new *model checking method for PDTMCs* that is faster than the only other available tool, PARAM [HHWZ10], by orders of magnitude on many common benchmarks.

Counterexample generation based on path searching algorithms Based on the SCC abstraction described above, we developed a new method for generating counterexamples on possibly abstract input DTMCs that violate a probabilistic reachability property. We shaped the notion of a *critical subsystem* of a DTMC which is a sub-graph of the original system where the reachability property is also violated. This highlights the critical parts of the system. Moreover, this subsystem can be seen as a symbolic representation of a set of paths leading from the initial state to one of the target states inside the subsystem; this set forms a counterexample.

The search is done in a *hierarchical* manner by offering the possibility of starting the search on simple abstract graphs and concretizing important parts in order to explore the graph in more detail. The first proposed search algorithm, which we call *global search*, enumerates paths leading from initial states to target states of the system in descending order of their probabilities, as proposed in [HKD09]. Using these paths, a subsystem is incrementally built until the needed probability mass is reached, i. e., until the subsystem becomes *critical*. The second search algorithm, called *local search* or *fragment search*, incrementally extends a subsystem by finding most probable path fragments that connect paths which were considered before.

In comparison to [HKD09], these algorithms lead to improvements by several orders of magnitude in the number of paths that are needed to form counterexamples and for most benchmarks also in terms of computation time. These methods were published in [10]; an extended version is accessible as a technical report [16].

The implementation is publicly available as part of our open-source tool COMICS [6]. The tool offers both a graphical user interface and a command-line version for benchmarking. The GUI depicts the whole hierarchical refinement process which can be completely controlled by the user. Alternatively, many heuristics are offered to automate the process. The only other publicly available tool is DiPro [ALFLS11] which was outperformed on most benchmarks having up to a few million states.

These methods for explicit representations of DTMCs are presented in Chapter 5, Sections 5.2-5.5. The main contributions are:

1. A new *hierarchical scheme for finding counterexamples on DTMCs*. This allows to search for counterexamples on very large input graphs, as the abstract systems are very small and simply structured. As abstract counterexamples can be concretized, no information is lost, however, not the entire system has to be explored.

2. An *improvement* in terms of running time, the number of explicitly listed paths, and the possible size of input instances by orders of magnitude using the proposed search algorithms in comparison to previous approaches.

3. The publicly available *open-source tool* COMICS, which is easy to use and comes with both a command-line version for benchmarking and connecting it with other tools and a GUI where the user can create specialized benchmarks and explore the hierarchical counterexamples in detail.

We now report on the adaptions of the aforementioned methods to symbolic data structures. A very successful model checking method is *bounded model checking* [CBRZ01]. This technique was adapted to counterexample generation for DTMCs in [WBB09] using BDDs and MTBDDs as representations for the input systems. Proceeding from this *symbolic* representation, modern SAT solvers are used to list the paths that form a counterexample. Thereby, paths leading from the initial state to a target state are encoded by a formula in a way that a satisfying solution of the formula corresponds to such a path.

As the bounded model checking approach itself finds arbitrary paths leading from an initial state to one of the target states, the probabilities are ignored. It may therefore be the case that many paths of low probabilities are found while it would lead to an earlier termination to find the paths in descending order of their probabilities. While this *informed* search is not directly possible for SAT solvers, we tackled this problem by using *SAT modulo Theories* (*SMT*) solving instead. This gave us the opportunity to compute a path's probability not by an external callback but actually to encode the probability computation into the SMT formula and pose restrictions on the minimal probability of a path: A formula is satisfied if and only if the corresponding path has at least a certain probability. This new way of bounded model checking leads to fewer search iterations in most of the cases. These approaches are part of the publications [25, 26] and are not contributions of this thesis.

Subsequently, we adapted our previous explicit approaches—namely the counterexample representation as *critical subsystems* and the *fragment search* approach—to the setting of bounded model checking. In combination we were able to achieve remarkable improvements to the applicability of bounded model checking for many test cases as reported in [7, 3].

In order to prefer more probable paths during the search process, we guide the SAT solver during its variable assignments. Considering partial assignments of variables that encode states of the input system, one can determine which possible paths can be encoded by the remaining, still unassigned, variables. We developed a heuristic which forces the SAT solver to choose assignments that induce more probable path fragments.

Even though these bounded model checking approaches improved the previous works, they were in general still not applicable to very large DTMCs. We therefore saw the need to investigate possibilities of working directly on the symbolic representations instead of letting a SAT solver enumerate paths. By that time, the only available fully symbolic method for the generation of counterexamples was an adaption of the approach given in [HKD09] to symbolic data structures [GSS10].

As this approach was only a sort of proof-of-concept to show that one can compute the k-shortest paths directly on an MTBDD which was not applicable to large benchmarks, we strived to develop new methods. The first result was presented in [7] and is basically an extension of the search algorithms presented in [10]. Using this method symbolically, we were able to generate counterexamples for systems with up to 10^8 states within reasonable time and with low memory consumption. A comprehensive article further improving the methods such that systems with up to 10^{15} states could be handled, is published in [3].

These symbolic approaches are presented in Chapter 5, Sections 5.6 and 5.7. Altogether the achievements are:

1. An *adaption* of the usage of critical subsystems and a path-fragment search approach to *bounded model checking*.

2. A new heuristics to *guide* a SAT solver to assign states such that more probable paths are preferred.

3. New methods that work directly on the *symbolic* representation of a DTMC thus enabling the generation of counterexamples for systems consisting of billions of states.

Counterexamples as minimal critical subsystems In [HKD09], the notion of *minimal counterexamples* in terms of the number of paths was shaped. These might still be very large or even infinite. According to our counterexample representation as critical subsystems, we wanted to explore the possibilities of computing a *minimal* subgraph of an input DTMC that still violates a certain property. Here, minimality can be defined both in terms of the number of states and in the number of transitions. In any case, the size of the representation is always bounded by the size of the original system. As a first approach we used (non-optimizing) SMT solvers combined with a binary search over the number of involved states to compute such *minimal critical subsystems*. During the experimental evaluation, it turned out that using *mixed integer linear programming* (*MILP*) [Sch86] was more efficient by orders of magnitude than SMT for this setting. Together with a number of optimizations, we were able to solve this minimization problem for many benchmarks with up to roughly ten thousand states. Thereby, we could not only compute very small counterexamples but we could also measure the quality of our previous heuristic approaches as described before. Although larger benchmarks could in general not be solved to optimality within a predefined time limit, all state-of-the-art MILP solvers like `SCIP` [Ach09], `CPLEX` [cpl12], or `GUROBI` [Gur13] are not only able to return the best solution found until the time limit but also to give a lower bound on the value of the optimal solution. This allows to judge the quality of an (possibly not yet optimal) intermediate solution and enables the applicability to larger benchmarks.

Moreover, we extended this approach to MDPs, where the problem of finding a minimal critical subsystem has been proven to be NP-hard [CV10]. We published our results for DTMCs and MDPs in [9].

Subsequently, we developed the first directly usable method to compute counterexamples for arbitrary *ω-regular properties* and DTMCs, which was published in [8]. A comprehensive article containing a method to compute counterexamples for ω-regular properties of *MDPs* was published in [4]; the correctness and completeness of all MILP and SMT encodings is also included. A corresponding technical report is available [15].

We present these approaches in Chapter 6 where the methods are extended to probabilistic automata (PAs). All in all we developed:

1. A method to compute *minimal* critical subsystems for reachability properties using SMT and MILP solving.

2. The first method to compute counterexamples for arbitrary ω-regular properties both for *DTMCs* and *PAs*.

High-level counterexamples All approaches summarized above use a state-based representation of counterexamples. A very common way to model probabilistic systems like DTMCs, MDPs or PAs is to use an extension [HSM97] of Dijkstra's *guarded command language* [Dij75]. Using this language, single *modules* describing the behavior of system components are specified. These modules can be accumulated to build the entire system, usually a PA, by means of *parallel composition*. This is the modeling formalism used for `PRISM` [KNP11], adapted from a stochastic version of Alur and Henzinger's reactive modules [AH99].

It seems a natural idea to provide counterexamples not at the state space level of the system but at the actual modeling level. On the one hand, this is due to the fact that a system designer likes to be pointed to errors at this level. On the other hand, while the high-level descriptions in the PRISM language are often very compact, they might result in PAs with millions of states where even minimal counterexamples or subsystems are incomprehensible.

The idea of such *critical model descriptions* is to automatically compute fragments of the original description modules that are relevant for the violation of a property. In detail, we determine—

a preferably small—set of guarded commands that together induce a critical subsystem when composed. As a consequence, in order to correct the system description, at least one of the returned guarded commands has to be changed.

Our methodology is as follows: First, the composition of the modules induces a full PA model at the state space level. During the composition, each resulting transition is labeled with unique identifiers of its constituting guarded command(s), i. e., the commands that actually induce this transition. Then, model checking is applied to determine whether a property is violated. The problem of computing a *minimal critical command set* is then reduced to computing a *minimal critical labeling* of the PA. Utilizing this flexible problem formulation, we present several alternatives of human-readable counterexamples. Furthermore, we show that this problem is NP-hard, which justifies the usage of an MILP solver. Besides the theoretical principles of our technique, we illustrate its practical feasibility by showing the results of applying a prototypical implementation to various examples from the `PRISM` benchmark suite. This approach was first presented in [5]. An extended version was published in [1]. This version is also accessible as a technical report [13].

Finally, the high-level approach is explained in Chapter 7. The contributions are:

1. A new approach to constructing counterexamples based on the *high-level* modeling language for probabilistic systems.

2. A flexible formulation in terms of a *minimal critical labeling*, enabling the possibility to create human-readable counterexamples.

1.2 Relevant publications

In this section we list the articles that contain contributions presented in this thesis. We distinguish peer-reviewed publications such as conference proceedings or journals and technical reports. Afterwards, we give a short note on how the author has contributed to the contents. The section concludes with publications of the author that are not content of this thesis.

1.2.1 Peer-reviewed publications

[1] Ralf Wimmer, Nils Jansen, Erika Abraham, and Joost-Pieter Katoen. High-level counterexamples for probabilistic automata. *Logical Methods in Computer Science*, 11(1:15), 2015.

[2] Nils Jansen, Florian Corzilius, Matthias Volk, Ralf Wimmer, Erika Ábrahám, Joost-Pieter Katoen, and Bernd Becker. Accelerating parametric probabilistic verification. In *Proc. of QEST*, volume 8657 of *LNCS*, pages 404–420. Springer, 2014.

[3] Nils Jansen, Ralf Wimmer, Erika Ábrahám, Barna Zajzon, Joost-Pieter Katoen, Bernd Becker, and Johann Schuster. Symbolic counterexample generation for large discrete-time Markov chains. *Science of Computer Programming*, 91(A):90–114, 2014.

[4] Ralf Wimmer, Nils Jansen, Erika Ábrahám, Joost-Pieter Katoen, and Bernd Becker. Minimal counterexamples for linear-time probabilistic verification. *Theoretical Computer Science*, 549:61–100, 2014.

[5] Ralf Wimmer, Nils Jansen, Andreas Vorpahl, Erika Abraham, Joost-Pieter Katoen, and Bernd Becker. High-level counterexamples for probabilistic automata. In *Proc. of QEST*, volume 8054 of *LNCS*, pages 18–33. Springer, 2013.

[6] Nils Jansen, Erika Ábrahám, Matthias Volk, Ralf Wimmer, Joost-Pieter Katoen, and Bernd Becker. The COMICS tool – Computing minimal counterexamples for DTMCs. In *Proc. of ATVA*, volume 7561 of *LNCS*, pages 349–353. Springer, 2012.

[7] Nils Jansen, Erika Ábrahám, Barna Zajzon, Ralf Wimmer, Johann Schuster, Joost-Pieter Katoen, and Bernd Becker. Symbolic counterexample generation for discrete-time Markov chains. In *Proc. of FACS*, volume 7684 of *LNCS*, pages 134–151. Springer, 2012.

[8] Ralf Wimmer, Nils Jansen, Erika Ábrahám, Joost-Pieter Katoen, and Bernd Becker. Minimal critical subsystems as counterexamples for ω-regular DTMC properties. In *Proc. of MBMV*, pages 169–180. Verlag Dr. Kovač, 2012.

[9] Ralf Wimmer, Nils Jansen, Erika Ábrahám, Joost-Pieter Katoen, and Bernd Becker. Minimal critical subsystems for discrete-time Markov models. In *Proc. of TACAS*, volume 7214 of *LNCS*, pages 299–314. Springer, 2012.

[10] Nils Jansen, Erika Ábrahám, Jens Katelaan, Ralf Wimmer, Joost-Pieter Katoen, and Bernd Becker. Hierarchical counterexamples for discrete-time Markov chains. In *Proc. of ATVA*, volume 6996 of *LNCS*, pages 443–452. Springer, 2011.

[11] Erika Ábrahám, Nils Jansen, Ralf Wimmer, Joost-Pieter Katoen, and Bernd Becker. DTMC model checking by SCC reduction. In *Proc. of QEST*, pages 37–46. IEEE Computer Society, 2010.

1.2.2 Technical reports

[12] Nils Jansen, Florian Corzilius, Matthias Volk, Ralf Wimmer, Erika Ábrahám, Joost-Pieter Katoen, and Bernd Becker. Accelerating parametric probabilistic verification. CoRR, 2013.

[13] Ralf Wimmer, Nils Jansen, Andreas Vorpahl, Erika Ábrahám, Joost-Pieter Katoen, and Bernd Becker. High-level counterexamples for probabilistic automata. Technical report, University of Freiburg, 2013.

[14] Nils Jansen, Erika Ábrahám, Maik Scheffler, Matthias Volk, Andreas Vorpahl, Ralf Wimmer, Joost-Pieter Katoen, and Bernd Becker. The COMICS tool - Computing minimal counterexamples for discrete-time Markov chains. CoRR, 2012.

[15] Ralf Wimmer, Nils Jansen, Erika Ábrahám, Joost-Pieter Katoen, and Bernd Becker. Minimal counterexamples for refuting ω-regular properties of Markov decision processes. Reports of SFB/TR 14 AVACS, 2012.

[16] Nils Jansen, Erika Ábrahám, Jens Katelaan, Ralf Wimmer, Joost-Pieter Katoen, and Bernd Becker. Hierarchical counterexamples for discrete-time Markov chains. Technical report, RWTH Aachen University, 2011.

1.2.3 A note on contributions by the author

All publications listed above were the result of the work of many people, as can be seen by the numerous authors. Most of the results were jointly achieved in many discussions via phoning, texting and of course many meetings in Aachen, Freiburg, Dagstuhl, or at conferences.

However, for this thesis it is necessary to somehow measure my contributions. I will therefore now chronologically, i. e., according to the publication date, explain what I explicitly contributed. In general, I co-wrote all publications and technical reports. I will therefore only talk about the development of the results.

To begin with, in [11] the basic idea was developed by me together with Erika Ábrahám in various initial discussions while getting familiar with the topic of probabilistic systems. The implementation was mainly done by me.

The results of [10, 16] again were developed together by Erika Ábrahám and me, while the implementation was done by a student under my supervision. The tool presented in [6, 14] was developed by me with great help of a number of students which are mentioned in the acknowledgements.

Extending the former approaches on counterexample generation to symbolic data structures was inspired by a previous work of Ralf Wimmer and initiated by me, which led to [7]. The implementation was done by a student and me. I was the main developer of the further improvements which led to the extensions in [3].

The initial idea of [9] came up at a discussion in Dagstuhl and was initially carried out by Ralf Wimmer. I developed several improvements to get scalable methods and worked with Ralf Wimmer and Erika Ábrahám on most of the proofs. The key idea for some of the extensions in [4, 15] was given by Joost-Pieter Katoen while Ralf Wimmer and I initiated the further development. Most of the details were worked out in several discussions in Aachen and Dagstuhl. The implementations for [4, 15] were done by Ralf Wimmer and me together with a student.

Ralf Wimmer had the idea for [5, 13] while he and I together worked on all the details and again did the implementation together with a student.

The part of the results of [2, 12] that is presented in this thesis were mainly developed by me while the implementation was done by a student under the supervision of Florian Corzilius and me.

1.3 Further publications

Additionally, we list the publications of the author that are not contained in this thesis.

[17] Christian Dehnert, Sebastian Junges, Nils Jansen, Florian Corzilius, Matthias Volk, Harold Bruintjes, Joost-Pieter Katoen, and Erika Abraham. Prophesy: A probabilistic parameter synthesis tool. In *Proc. of CAV*, 2015. To appear.

[18] Friedrich Gretz, Nils Jansen, Benjamin Lucien Kaminski, Joost-Pieter Katoen, Annabelle McIver, and Federico Olmedo. Conditioning in probabilistic programming. In *Proc. of MFPS*, LNCS. Springer, 2015. To appear.

[19] Shashank Pathak, Erika Ábrahám, Nils Jansen, Armando Tacchella, and Joost-Pieter Katoen. A greedy approach for the efficient repair of stochastic models. In *Proc. of NFM*, volume 9058 of *LNCS*, pages 295–309. Springer, 2015.

[20] Tim Quatmann, Nils Jansen, Christian Dehnert, Ralf Wimmer, Erika Abraham, Joost-Pieter Katoen, and Bernd Becker. Counterexamples for expected rewards. In *Proc. of FM*, volume 9109 of *LNCS*, pages 435–452. Springer, 2015.

[21] Erika Ábrahám, Bernd Becker, Christian Dehnert, Nils Jansen, Joost-Pieter Katoen, and Ralf Wimmer. Counterexample generation for discrete-time Markov models: An introductory survey. In *Proc. of SFM*, volume 8483 of *LNCS*, pages 65–121. Springer, 2014.

[22] Christian Dehnert, Nils Jansen, Ralf Wimmer, Erika Ábrahám, and Joost-Pieter Katoen. Fast debugging of PRISM models. In *Proc. of ATVA*, volume 8837 of *LNCS*, pages 146–162. Springer, 2014.

[23] Daniel Neider and Nils Jansen. Regular model checking using solver technologies and automata learning. In *Proc. of NFM*, volume 7871 of *LNCS*, pages 16–31. Springer, 2013.

[24] Erika Ábrahám, Nadine Bergner, Philipp Brauner, Florian Corzilius, Nils Jansen, Thiemo Leonhardt, Ulrich Loup, Johanna Nellen, and Ulrik Schroeder. On collaboratively conveying computer science to pupils. In *Proc. of KOLI*, pages 132–137. ACM, 2011.

[25] Bettina Braitling, Ralf Wimmer, Bernd Becker, Nils Jansen, and Erika Ábrahám. Counterexample generation for Markov chains using SMT-based bounded model checking. In *Proc. of FMOODS/FORTE*, volume 6722 of *LNCS*, pages 75–89. Springer, 2011.

[26] Bettina Braitling, Ralf Wimmer, Bernd Becker, Nils Jansen, and Erika Ábrahám. SMT-based counterexample generation for Markov chains. In *Proc. of MBMV*, pages 19–28. Offis Oldenburg, 2011.

[27] Erika Ábrahám, Philipp Brauner, Nils Jansen, Thiemo Leonhardt, Ulrich Loup, and Ulrik Schroeder. Podcastproduktion als kollaborativer Zugang zur theoretischen Informatik. In *Proc. of DeLFI*, volume 169 of *LNI*, pages 239–251. Gesellschaft für Informatik, 2010.

1.4 How to read this thesis

In order to ease the reading of this work, we begin every chapter with a short paragraph containing a *Summary* of what is presented in this chapter. This is followed by a paragraph called *Background*, which contains the preliminaries that are relevant for this chapter together with a link to the according definitions or sections in the chapter about *Foundations*.

1.4.1 Algorithms

All algorithms are given in pseudo code and more or less mathematically formal, depending on the problem at hand. For every algorithm we list all used parameters, variables and methods afterwards. This is followed by an explanation on how the algorithm proceeds. A small toy example can be seen below:

Algorithm 1

```
Integer ExampleAlgo(Integer $x$)
begin
    Integer $y := 0$
    $y := x$
    return testMethod($y$)
end
```

Parameters

y is an integer parameter.

Return value

The result of `testMethod(y)` is returned.

Variables

x is an integer variable. If variables are not initialized, they are considered to have the value $\emptyset$.

Methods

testMethod does something with integer parameter y and returns an integer value.

Procedure

The algorithm proceeds as follows: First, the variable y is initialized to 0. Then, it is set to the value of the parameter x. Finally, the result of **testMethod** called with parameter y is returned.

1.4.2 Problem encodings

In some parts of this thesis, problems are handled by utilizing solving techniques. Therefore, we have to encode a solution to a problem by means of constraints that have to be satisfied. As these constraints are not very intuitive at some parts, we present all encodings in the following form.

Intuition Here we give the intuition of the problem that is to be encoded. In our toy example we want to ensure that all used variables sum up to a value that is less than or equal to 1.

Variables

$x \in [0,1] \subseteq \mathbb{R}$ is a real-valued variable between 0 and 1

$y \in \{0,1\}$ is an integer variable that can take the values 0 or 1

Constraints

$$x + y \leq 1 \tag{1.1}$$

Explanation The sum of x and y is forced to take a value less than or equal to 1 by Constraint 1.1. This implies that if the integer variable y is assigned 1, x has to be assigned 0. If y is assigned 0, x can be assigned any possible value out of $\mathbb{R}$ between 0 and 1.

Formula size The formula size is constant.

CHAPTER 2

Foundations

In this chapter we lay the foundations needed for the results presented in this thesis. After shortly fixing some basic notations we formalize the probabilistic models that are subject of our methods. We present the different specification languages as well as corresponding verification techniques for the introduced models. We also discuss the concept of symbolic representations of state spaces, which is used for large systems.

We introduce some basic solving techniques which are used throughout the thesis as blackbox algorithms. The chapter concludes with an introduction to probabilistic counterexamples.

2.1 Basic notations and definitions

Numbers, intervals and matrices We denote the set of real numbers by $\mathbb{R}$, the rational numbers by $\mathbb{Q}$, the natural numbers including 0 by $\mathbb{N}$, and the integer numbers by $\mathbb{Z}$. By $\mathbb{R}_{>0}$ we denote the positive real numbers and by $\mathbb{R}_{\geq 0}$ the nonnegative real numbers. The same holds for $\mathbb{Q}_{>0}, \mathbb{Q}_{\geq 0}, \mathbb{N}_{>0}, \mathbb{Z}_{>0}$, and $\mathbb{Z}_{\geq 0}$, respectively.

Let $[0,1] \subseteq \mathbb{R}$ denote the *closed interval* of all real numbers between 0 and 1, including the bounds; $(0,1] \subseteq \mathbb{R}$ denotes the *open interval* of all real numbers between 0 and 1 excluding 0.

$A \in \mathbb{R}^{m \times n}$ denotes a matrix A with m rows and n columns with values out of $\mathbb{R}$. The same holds for $\mathbb{N}^{m \times n}$, $\mathbb{Q}^{m \times n}$ and $\mathbb{Z}^{m \times n}$, respectively.

Sets Let X, Y denote arbitrary sets. If $X \cap Y = \emptyset$ holds, we write $X \uplus Y$ for the *disjoint union* of the sets X and Y. As usual, we write $X \subseteq Y$ if X is *subset* of Y. If additionally $Y \setminus X \neq \emptyset$ holds, we write $X \subset Y$.

By $2^X = \{X' \mid X' \subseteq X\}$ we denote the *power set* of X. By $(X_i)_{i \geq 0}$ we denote an arbitrary *family* of sets X_i. We call a set $P \subseteq 2^X \setminus \emptyset$ a *partition* of X if

$$\bigcup_{P' \in P} P' = X \qquad \text{and} \qquad \forall P', P'' \in P \colon P' \neq P'' \Rightarrow P' \cap P'' = \emptyset$$

holds. The elements of the partition P are called *blocks*. By X^ω we denote the set of all infinite sequences of elements from X: $x_1, x_2, \ldots$ for $x_i \in X, i > 0$. X^n denotes the set of all finite sequences $x_1, \ldots, x_n$ of X for $x_i \in X, 1 \leq i \leq n$.

Distributions Let X be a finite or countably infinite set. A *probability distribution* over X is a function $\mu \colon X \to [0,1] \subseteq \mathbb{R}$[1] with $\sum_{x \in X} \mu(x) = \mu(X) = 1$. A *sub-stochastic distribution* over X is a function $\mu \colon X \to [0,1] \subseteq \mathbb{R}$ such that $\sum_{x \in X} \mu(x) = \mu(X) \leq 1$. We denote the set of all (sub-stochastic) distributions on X by $\mathit{Distr}(X)$ or $\mathit{subDistr}(X)$, respectively. The set $\mathit{supp}(\mu) = \{x \in X \mid \mu(x) > 0\}$ is called the *support* of μ for a (sub-stochastic) distribution μ. We write $\mu' \subseteq \mu$ for $\mu, \mu' \in \mathit{subDistr}(X)$, if for all $x \in X$ it holds that $\mu'(x) > 0$ implies $\mu'(x) = \mu(x)$ and $\mu'(x) = 0$ for $\mu(x) = 0$. If for $\mu \in \mathit{Distr}(X)$ it holds that $\mu(x) = 1$ for one $x \in X$ and $\mu(y) = 0$ for all $y \in X \setminus \{x\}$, μ is called a *Dirac distribution*.

Polynomials For an arbitrary function $f : X \to Y$ we denote the *domain of f* by $\mathrm{dom}(f) = X$. Let V denote a set of variables. The domain of a variable $x \in V$ is denoted by $\mathrm{dom}(x)$. The domain of the whole set of variables is denoted by $\mathrm{dom}(V) = \bigcup_{x \in V} \mathrm{dom}(x)$.

Definition 1 (Polynomial, monomial) *Let $V = \{x_1, \ldots, x_n\}$ be a set of* variables *with $\mathrm{dom}(V) = \mathbb{R}$. A* monomial *$m$ over V is the product of a coefficient $a \in \mathbb{Z}$ and variables from V:*

$$m := a \cdot x_1^{e_1} \cdot \ldots \cdot x_n^{e_n}$$

with $e_i \in \mathbb{N}$ for all $1 \leq i \leq n$. The set of all monomials over V is denoted by Mon_V. A polynomial *g over V is a sum of monomials over V:*

$$g := m_1 + \ldots + m_l$$

with $m_i \in Mon_V, 1 \leq i \leq l$. The set of all polynomials over V is denoted by Pol_V.

We also need the notion of a *rational function*, which is the fraction $\frac{g_1}{g_2}$ of two polynomials. Thereby, this function is clearly not defined at the roots of the denominator. We additionally restrict the denominator to be unequal to 0. By $g_2 \neq 0$ we state that the polynomial g_2 cannot be simplified to 0 which is necessary to define the division by g_2. Moreover, the rational function $\frac{g_1}{g_2}$ is not defined at the roots of g_2. In order to have valid evaluations of a rational function for our algorithms, we later restrict these evaluations accordingly.

[1] For the algorithms in this thesis, probabilities are from $\mathbb{Q}$.

Definition 2 (Rational function) *Let $V = \{x_1, \ldots, x_n\}$ be a set of* variables *with* $\mathrm{dom}(V) = \mathbb{R}$. *A* rational function *f over V is given by $f := \frac{g_1}{g_2}$ with $g_1, g_2 \in Pol_V$ and $g_2 \neq 0$. The set of all rational functions over V is denoted by Rat_V.*

2.2 Probabilistic models

We introduce the different probabilistic models which are used throughout this thesis. All our models are based on *discrete time*, i. e., instantaneous transitions discretely model the advance of time by one time unit. Possible interpretations are that scenarios like the ticks of clock are modeled or that for the probabilistic information the amount of time that is used is irrelevant. *Continuous time models* like continuous time Markov chains are out of the scope of this thesis. For an overview of different models, both discrete-time and continuous-time, we refer to [Fel68, How79].

2.2.1 Discrete-time Markov chains

Intuitively, Markov Chains are transition systems where at each state there is a probabilistic choice for the successor state. If the time model is discretized such that we have instantaneous transitions, we speak of *discrete-time Markov chains*. The probability of going from a state to another one depends only the current state, i. e., this probability is *history independent* of the path taken so far. This property is called the *Markov property*. This is due to the fact, that a DTMC is actually a *stochastic process* whose random variables are all *geometrically distributed*. For details we refer to [BK08, Chapter 10].

Let in the following *AP* be a finite set of *atomic propositions*.

Definition 3 (DTMC) *A* discrete-time Markov chain (DTMC) *is a tuple $\mathcal{D} = (S, I, P, L)$ with:*

- *S a countable set of states*
- *$I \in subDistr(S)$ an* initial distribution
- *$P \colon S \to subDistr(S)$ a* transition probability matrix
- *$L \colon S \to 2^{AP}$ a* labeling function.

Remark 1 (Finite state space) *Note that although the set of states is allowed to be countably infinite, for our algorithms we assume* finite *state spaces.*

The probability of going to a state $s' \in S$ from a state $s \in S$ in one step is given by the corresponding entry $P(s,s')$ in the transition probability matrix. The probability to reach a set $S' \subseteq S$ of states from s in one step is given by $P(s,S') = \sum_{s' \in S'} P(s,s')$. Analogously, $P(S',s) =$

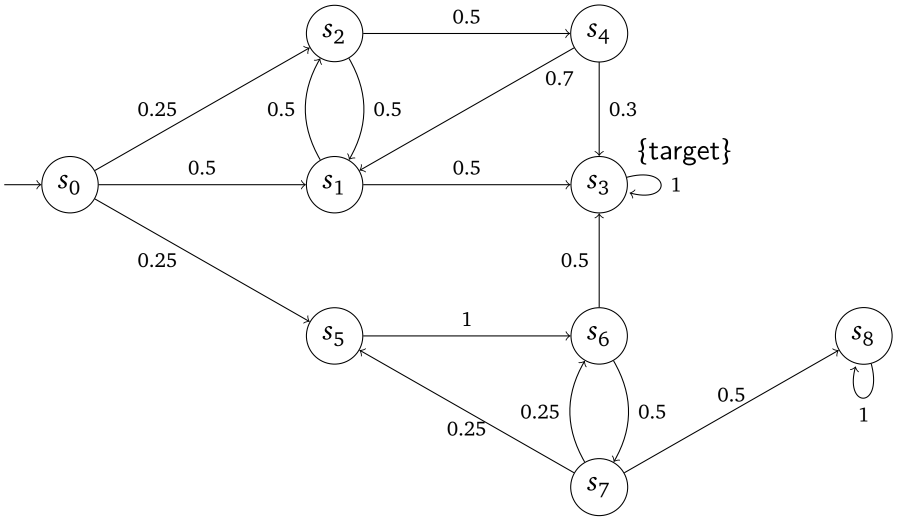


Figure 2.1: Example DTMC $\mathcal{D}$

$\sum_{s' \in S'} P(s', s)$ holds. The initial distribution determines the probability of starting at certain states. We denote the set of states where this probability is not 0, the *initial states*, by

$$Init_{\mathcal{D}} = \{s \in S \mid I(s) > 0\} .$$

Example 1 *Consider a DTMC $\mathcal{D} = (S, I, P, L)$ with $S = \{s_0, \ldots, s_8\}$, $I(s_0) = 1$, $L(s_3) = \{\text{target}\}$ and $L(s) = \emptyset$ for all $s \neq s_3$. The transition probability matrix is given by:*

$$P = \begin{pmatrix} 0 & 0.5 & 0.25 & 0 & 0 & 0.25 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0.5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 & 0.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.7 & 0 & 0.3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0 & 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.25 & 0.25 & 0 & 0.5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

The DTMC $\mathcal{D}$ is depicted in Figure 2.1.

Every state has (possibly empty) sets of predecessors and successors:

Definition 4 (Predecessors, successors) *For a DTMC $\mathscr{D} = (S, I, P, L)$, the* successors *of a state $s \in S$ are given by*

$$succ_{\mathscr{D}}(s) = \{s' \in S \,|\, P(s,s') > 0\} \ .$$

The predecessors *of $s \in S$ are given by*

$$pred_{\mathscr{D}}(s) = \{s' \in S \,|\, P(s',s) > 0\} \ .$$

A state $s \in S$ with $succ_{\mathscr{D}}(s) = \{s\}$ is called an *absorbing state*.

Remark 2 (Sub-stochastic distribution) *Our definition of a DTMC allows sub-stochastic distributions for the initial distribution I and for the transition probability matrix P. Usually, the sum of probabilities is required to be exactly 1. This can be obtained by defining for a DTMC $\mathscr{D}$ a new DTMC $\mathscr{D}' = (S', I', P', L')$ where all "missing" probabilities are directed to a new sink state $s_\perp$:*

- $S' = S \uplus \{s_\perp\}$
- $I'(s) = \begin{cases} I(s) & \textit{if } s \in S \\ 0 & \textit{otherwise} \end{cases}$
- $P'(s,s') = \begin{cases} P(s,s') & \textit{if } s,s' \in S \\ 1 & \textit{if } s = s' = s_\perp \\ 1 - \sum_{s' \in S} P(s,s') & \textit{if } s \in S, s' = s_\perp \\ 0 & \textit{if } s = s_\perp, s' \in S \end{cases}$
- $L'(s) = \begin{cases} L(s) & \textit{if } s \in S \\ \emptyset & \textit{if } s = s_\perp \ . \end{cases}$

We often directly or indirectly refer to the graph which is induced by a DTMC $\mathscr{D}$:

Definition 5 (Graph of a DTMC) *For a DTMC $\mathscr{D} = (S, I, P, L)$, the* underlying directed graph *of $\mathscr{D}$ is given by $\mathscr{G}_{\mathscr{D}} = (S, E_{\mathscr{D}})$ with the* set of edges

$$E_{\mathscr{D}} = \{(s,s') \in S \times S \mid s' \in succ_{\mathscr{D}}(s)\}.$$

We call the edges of $\mathscr{G}_{\mathscr{D}}$ also *transitions*. A transition $(s,s') \in E_{\mathscr{D}}$ has a *source state* s and a *destination state* s'. We denote the number of states by $|S| = n_{\mathscr{D}}$ and the number of transitions by $|E_{\mathscr{D}}| = m_{\mathscr{D}}$.

Remark 3 (Explicit graph representation) *If the matrix is explicitly stored in data-structures like vectors or 2-dimensional matrices, we speak of an* explicit graph representation. *This representation is used, for instance, by the probabilistic model checker* `MRMC` *[KZH$^+$11]. This is the counterpart of* symbolic graph representations *which are discussed in Section 2.4.*

Sparse matrix representation As indicated by the previous example, typically the transition probability matrices of DTMCs contain many entries having the value 0, i. e., the matrices are very *sparse*. For many operations it is beneficial to use so-called *sparse matrix representations*. Three vectors *row*, *col* and *val* are used to store A. The vector *val* stores all non-zero entries in a certain order. Vector *col* stores the indices of the columns of the values in *val*. *row* stores pointers to the other vectors such that the ith element of *row* points to the first entry of row i in the original matrix A represented by *col* and *val*. This compact representation is beneficial for numerical operations like matrix-vector multiplications. For more details, we refer to [Par02, Ste94].

If only a part of the states of a DTMC is to be considered, we use the notion of a *subsystem*.

Definition 6 (Subsystem of a DTMC) *For a DTMC $\mathcal{D} = (S, I, P, L)$, a* subsystem *of $\mathcal{D}$ is a DTMC $\mathcal{D}' = (S', I', P', L')$ with:*

- $S' \subseteq S$
- $I'(s) = I(s)$ *for all* $s \in S'$
- $P'(s, s') = P(s, s')$ *for all* $s, s' \in S'$
- $L'(s) = L(s)$ *for all* $s \in S'$.

We write $\mathcal{D}' \sqsubseteq \mathcal{D}$.

Paths of DTMCs Assume in the following a DTMC $\mathcal{D} = (S, I, P, L)$. We fix some notations regarding the paths that go through a DTMC.

Definition 7 (Path of an DTMC) *A* path π *of a DTMC $\mathcal{D} = (S, I, P, L)$ is a finite or infinite sequence $\pi = s_0, s_1, \ldots$ of states such that $s_i \in S$ and $P(s_i, s_{i+1}) > 0$ for all $i \geq 0$.*

If π is a *finite path* $\pi = s_0, s_1, \ldots, s_n$ we define the *length* of π by $|\pi| = n$. The length of an infinite path π is $|\pi| = \infty$. The first state π of a path is denoted by *first*$(\pi) = s_0$; the last state of a finite path π is denoted by *last*$(\pi) = s_n$. By π^i we denote the prefix of π of length i. Let $\pi(i)$ denote the ith state of π. By $s_0, \ldots, s_j, (s_{j+1}, \ldots, s_k)^*, \ldots$ for $0 \leq j < k \leq |\pi|$ and $s_j = s_k$ we denote a set of paths that are equal except for the number of iterations of the loop $s_j, \ldots, s_k$.

Example 2 (Loops) *Consider the DTMC $\mathcal{D}$ from Example 1. The path $\pi_1 = s_0, s_1, s_2, s_1, s_3$ contains a loop $\pi_l = s_1, s_2, s_1$. The set of all paths differing only in the number of iterations of this loop is denoted by $\Pi = s_0, s_1, (s_2, s_1)^*, s_3$. For instance, $\pi_2 = s_0, s_1, s_2, s_1, s_2, s_1, s_3 \in \Pi$. Moreover, the path $\pi_3 = s_0, s_1, s_3$ not traversing that loop is also part of Π.*

By $\mathit{Paths}^{\mathcal{D}}$ we denote the set of all paths of $\mathcal{D}$ and by $\mathit{Paths}^{\mathcal{D}}(s)$ the set of all paths of $\mathcal{D}$ that start in state $s \in S$. Analogously, we define the sets of finite paths $\mathit{Paths}^{\mathcal{D}}_{\mathit{fin}}$ and infinite paths $\mathit{Paths}^{\mathcal{D}}_{\mathit{inf}}$ as well as $\mathit{Paths}^{\mathcal{D}}_{\mathit{fin}}(s)$ and $\mathit{Paths}^{\mathcal{D}}_{\mathit{inf}}(s)$, respectively. For states s and s' we denote by $\mathit{Paths}^{\mathcal{D}}_{\mathit{fin}}(s, s')$ the set of finite paths that start in s' and end in s''. For sets of states $S' \subseteq S$ and $S'' \subseteq S$ we denote by $\mathit{Paths}^{\mathcal{D}}_{\mathit{fin}}(S', S'')$ the set of finite paths that start in a state $s \in S'$ and end in a state $s'' \in S''$, formally $\mathit{Paths}^{\mathcal{D}}_{\mathit{fin}}(S', S'') = \bigcup_{s' \in S'} \bigcup_{s'' \in S''} \mathit{Paths}^{\mathcal{D}}_{\mathit{fin}}(s', s'')$. Sometimes we omit the subscript $\mathcal{D}$ if it is clear from the context.

The *concatenation* of a finite path $\pi_1 \in \mathit{Paths}^{\mathcal{D}}_{\mathit{fin}}$ and a finite or infinite path $\pi_2 \in \mathit{Paths}^{\mathcal{D}}$ with $\mathit{last}(\pi_1) = \mathit{first}(\pi_2)$ is denoted by $\pi = \pi_1\pi_2 = \pi_1(0), \ldots, \pi_1(|\pi_1 - 1|), \pi_2(0), \pi_2(1), \ldots$ The set of all *prefixes* of $\pi \in \mathit{Paths}^{\mathcal{D}}$ is $\mathit{pref}_{\mathcal{D}}(\pi) = \{\pi^i \mid 0 \leq i \leq |\pi|\}$. If $|\pi_1| < |\pi|$ and $\pi_1 \in \mathit{pref}_{\mathcal{D}}(\pi)$ hold, π_1 is called a *real prefix of* π.

We call a set R of paths *prefix-free*, if for all $\pi_1, \pi_2 \in R \subseteq \mathit{Paths}^{\mathcal{D}}$ it holds that either $\pi_1 = \pi_2$ or $\pi_1 \notin \mathit{pref}_{\mathcal{D}}(\pi_2)$. By $\mathit{Paths}^{\mathcal{D}}_{k} \subseteq \mathit{Paths}^{\mathcal{D}}_{\mathit{fin}}$ we denote all finite paths of length $k \in \mathbb{N}$.

Definition 8 (Trace) *The* trace *of a finite or infinite path $\pi = s_0, s_1, \ldots$ of a DTMC $\mathcal{D} = (S, I, P, L)$ is given by the sequence $\mathit{trace}(\pi) = L(s_0), L(s_1) \ldots$*

We sometimes use state names as atomic propositions, i. e., $s \in L(s)$ for $s \in S$.

Probability measure on DTMCs In order to define a probability measure for sets of infinite paths, we first have to recall the general notions of *σ-algebras* and associated *probability measures*, which together form a *probability space*.

Definition 9 (σ-algebra) *Let X be a set. A set $\Sigma \subseteq 2^X$ is called a σ-*algebra *over X if:*

1. *$X \in \Sigma$ (underlying set contained)*
2. *$Y \in \Sigma \Rightarrow (X \setminus Y) \in \Sigma$ (closed under complement)*
3. *$Y_1, Y_2, \ldots \in \Sigma \Rightarrow (\bigcup_{i \geq 1} Y_i) \in \Sigma$ (closed under countable union).*

From Conditions 1 and 2 it follows that $\emptyset \in \Sigma$. In order to assign probabilities to the elements of a σ-algebra, a *probability measure* is associated.

Definition 10 (Probability measure) *Let Σ be a σ-algebra over a set X. A function $\mu\colon \Sigma \to [0, 1] \subseteq \mathbb{R}$ is called a* probability measure *if:*

- *$\mu(\emptyset) = 0$ (empty set is assigned zero)*

- *$\forall Y \in \Sigma \colon \mu(Y) = 1 - \mu(X \setminus Y)$ (closed under complement)*
- *For all families $(Y_i)_{i \geq 1}$ of pairwise disjoint sets $Y_i \in \Sigma$ it holds that*

$$\mu(\bigcup_{i\geq 1} Y_i) = \sum_{i \geq 1} \mu(Y_i) \text{ (countable additivity).}$$

These elements form a probability space.

Definition 11 (Probability space) *Let Σ be a σ-algebra over a set X and $\mu \colon \Sigma \to [0,1] \subseteq \mathbb{R}$ an associated probability measure. The triple (X, Σ, μ) is called a* probability space.

In the literature, X is often referred to as the *sample space* and the elements of the σ-algebra are called *events*. For more details we refer to standard books about measure theory, such as [BW07, BT02].

We now associate a probability space to a DTMC $\mathcal{D}$. The set $Paths_{inf}$ of infinite paths of $\mathcal{D}$ forms the sample space. In order to define a suitable σ-algebra over $Paths_{inf}$ we define so-called *cylinder sets* of finite paths.

Definition 12 (Cylinder set) *The* cylinder set *of a finite path $\pi = s_0, s_1, \ldots, s_n \in Paths^{\mathcal{D}}_{fin}(s_0)$ is the set $cyl(\pi) = \{\pi' \in Paths^{\mathcal{D}}_{inf}(s_0) \mid \pi \in pref_{\mathcal{D}}(\pi')\}$ of all infinite extensions of π.*

For a DTMC $\mathcal{D}$ and a state $s \in S$, a *probability space* $(\Omega, \mathcal{F}, Pr^{\mathcal{D}}_s)$ can be defined as follows: The *sample space* $\Omega = Paths^{\mathcal{D}}_{inf}(s)$ is the set of all infinite paths starting in s. The *events* $\mathcal{F} \subseteq 2^{\Omega}$ are given by the smallest σ-algebra that contains the cylinder sets of all finite paths in $Paths^{\mathcal{D}}_{fin}(s)$, i. e., it is the closure of the cylinder sets under complement and countable union containing Ω. For details we refer to [BK08, Definition 10.10]. The probability of a finite path $\pi = s_0, s_1, \ldots, s_n$ is given by the product of all transition probabilities:

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

For paths π of length $|\pi| = 1$ we set $\mathbf{P}^{\mathcal{D}}(\pi) = 1$. Two finite paths present two *stochastically independent* events, if no path is a prefix of the other one, i. e., their cylinder sets are disjoint. If we are interested in the probability of finite paths that start in a specific state, we have

$$\mathbf{P}^{\mathcal{D}}_s(\pi) = \begin{cases} \mathbf{P}^{\mathcal{D}}(\pi) & \text{if } s_0 = s \\ 0 & \text{otherwise} \end{cases}$$

The probability measure is extended to sets of paths: For a prefix-free set of finite paths $R \subseteq \mathit{Paths}^{\mathcal{D}}_{\mathit{fin}}$ we have

$$\mathbf{P}^{\mathcal{D}}(R) = \sum_{\pi \in R} \mathbf{P}^{\mathcal{D}}(\pi).$$

For the *concatenation of two paths* $\pi = \pi_1\pi_2 \in \mathit{Paths}^{\mathcal{D}}_{\mathit{fin}}$ with $\pi_1, \pi_2 \in \mathit{Paths}^{\mathcal{D}}_{\mathit{fin}}$ and $\mathit{first}(\pi_2) = \mathit{last}(\pi_1) = s$ it holds that $\mathbf{P}^{\mathcal{D}}(\pi_1) \cdot \mathbf{P}^{\mathcal{D}}(\pi_2) = \mathbf{P}^{\mathcal{D}}(\pi)$.

The probability of cylinder sets is defined using the measure for finite paths:

$$\mathit{Pr}^{\mathcal{D}}_{s_0}\big(\mathit{cyl}(s_0, s_1, \ldots, s_n)\big) = \mathbf{P}^{\mathcal{D}}(s_0, s_1, \ldots, s_n)$$

This measure can uniquely be extended to the whole σ-algebra associated with $\mathcal{D}$, yielding the probability measure $\mathit{Pr}^{\mathcal{D}}_{s_0} \colon \mathscr{F} \to [0,1] \subseteq \mathbb{R}$ [Nor97]. A set Π of paths is called *measurable* iff $\Pi \in \mathscr{F}$.

Remark 4 (Measure for finite paths) *In the following we sometimes let finite paths denote their cylinder sets.*

Remark 5 (Initial distribution vs. initial state) *Instead of defining an initial distribution I for a DTMC $\mathcal{D}$, we often use a single* initial state *s_I. This is no restriction, as every DTMC having an arbitrary initial distribution can be transformed to one having one unique initial state. This transformation does not affect the satisfaction of the properties we investigate. Let $\mathcal{D} = (S, I, P, L)$ be a DTMC. We define $\mathcal{D}' = (S', I', P', L')$ with a fresh unique initial state $s_I \notin S$:*

- $S' = S \uplus \{s_I\}$
- $I'(s) = \begin{cases} 1 & \text{if } s = s_I \\ 0 & \text{if } s \in S \end{cases}$
- $P'(s,s') = \begin{cases} I(s') & \text{if } s = s_I, s' \in S \\ P(s,s') & \text{if } s, s' \in S \\ 0 & \text{otherwise} \end{cases}$
- $L'(s) = \begin{cases} \emptyset & \text{if } s = s_I \\ L(s) & \text{if } s \in S\ . \end{cases}$

We also write $\mathcal{D}' = (S', s_I, P', L')$.

Sets of states Throughout this thesis, we often have to argue about certain sets of states. For a DTMC $\mathcal{D} = (S, I, P, L)$ and a subset of states $K \subseteq S$ we define the set of *input states* of K in $\mathcal{D}$ by $\mathit{Inp}_{\mathcal{D}}(K) = \{s \in K \mid I(s) > 0 \lor \exists s' \in S \setminus K.\ P(s', s) > 0\}$, i. e., the states inside K that have an incoming transition from outside K. Analogously, we define the set of *output states* of K in $\mathscr{D}$ by $Out_{\mathscr{D}}(K) = \{s \in S \setminus K \mid \exists s' \in K.\ P(s', s) > 0\}$, i. e., the states outside K that have an incoming transition from a state inside K. The set of *inner states* of K in $\mathscr{D}$ is given by $K \setminus Inp_{\mathscr{D}}(K)$. We sometimes omit the subscript $\mathscr{D}$ if it is clear from the context. A set of states K in $\mathscr{D}$ is called *absorbing* in $\mathscr{D}$ if the probability to reach a state of $Out_{\mathscr{D}}(K)$ is less than 1, i. e., if $Out_{\mathscr{D}}(K)$ is not reached with probability 1.

Special sets of states are *strongly connected components*.

Definition 13 (Strongly connected component) *Let $\mathscr{D} = (S, I, P, L)$ be a DTMC and $S' \subseteq S$ a subset of states with $S' \neq \emptyset$. S' is called a* strongly connected sub-component (SCS) of *$\mathscr{D}$, if for all $s, s' \in S'$ there is a path $s_0, s_1, \ldots, s_n \in Paths^{\mathscr{D}}_{fin}(s, s')$ with $s_i \in S'$ for all $i = 0, \ldots, n$.*

A strongly connected subgraph of $\mathscr{D}$ is a called a strongly connected component (SCC) *if it is maximal, i. e., for all strongly connected subgraphs $S'' \subseteq S$ we have that $S' \not\subset S''$.*

S' is a bottom SCC (BSCC) *if it is an SCC and for all $s \in S'$ we have that $\sum_{s' \in S'} P(s, s') = 1$.*

Intuitively, from every state of an SCS S' every other state in S' is reachable inside S'. The SCC structure of a directed graph can be determined by Tarjan's algorithm in linear time [Tar72].

As the set of output states of a BSCC is empty, the probability to visit each state in a BSCC infinitely often is 1. Moreover, if we assume only *stochastic distributions*, the probability to finally reach a set of absorbing states, i. e., the set of all BSCCs of a DTMC, is 1. This implies that the probability to finally *leave* a non absorbing SCC is also 1. For details about these properties, we refer to [BK08, Chapter 10.1.2].

2.2.2 Probabilistic automata and Markov decision processes

In order to allow both probabilistic and nondeterministic behavior, *probabilistic automata* and *Markov decision processes* extend DTMCs by nondeterministic choices for probability distributions over successor states.

Definition 14 (PA) *A* probabilistic automaton (PA) *is a tuple $\mathscr{M} = (S, I, Act, \mathscr{P}, L)$ with:*

- *S a countable set of states*
- *$I \in subDistr(S)$ an* initial distribution
- *Act a finite set of actions*
- *$\mathscr{P} : S \to 2^{Act \times subDistr(S)}$ a* probabilistic transition relation *where $\mathscr{P}(s)$ is finite for all $s \in S$*
- *$L : S \to 2^{AP}$ a labeling function.*

If every action $\alpha \in Act$ occurs only once at each state of $\mathscr{M}$, the model is called a Markov decision process.

Definition 15 (MDP) *A discrete-time Markov decision process (MDP) is a probabilistic automaton $\mathscr{M} = (S, I, Act, \mathscr{P}, L)$ where it holds for all $s \in S$ and all $\alpha \in Act$ that*

$$|\{\mu \in subDistr(S) \mid (\alpha, \mu) \in \mathscr{P}(s)\}| \leq 1 \ .$$

For MDPs, the probabilistic transition relation can also be written as a three-dimensional transition probability matrix of the form $\mathscr{P} : S \times Act \to subDistr(S)$. If possible, we often use this notation.

Remark 6 (Finite state space) *As for DTMCs, we assume PAs and MDPs to have a finite state space, see Remark 1.*

We define sets of predecessors and successors for a state, for a pair of state and action, and for a state and a pair of action and distribution.

Definition 16 *For a PA $\mathscr{M} = (S, I, Act, \mathscr{P}, L)$, the* predecessors *of $s \in S$ for $\alpha \in Act$ and $\mu \in subDistr(S)$ are given by:*

$$\begin{aligned} pred_{\mathscr{M}}(s, \alpha, \mu) &= \{s' \in S \mid (\alpha, \mu) \in \mathscr{P}(s') \wedge s \in supp(\mu)\} \\ pred_{\mathscr{M}}(s, \alpha) &= \{s' \in S \mid \exists \mu \in subDistr(S).\, s' \in pred(s, \alpha, \mu)\} \\ pred_{\mathscr{M}}(s) &= \bigcup_{\alpha \in Act} pred_{\mathscr{M}}(s, \alpha). \end{aligned}$$

The successors *of $s \in S$ for $\alpha \in Act$ and $\mu \in subDistr(S)$ are given by:*

$$\begin{aligned} succ_{\mathscr{M}}(s, \alpha, \mu) &= \{s' \in S \mid (\alpha, \mu) \in \mathscr{P}(s) \wedge s' \in supp(\mu)\} \\ succ_{\mathscr{M}}(s, \alpha) &= \{s' \in S \mid \exists \mu \in subDistr(S).\, s' \in succ(s, \alpha, \mu)\} \\ succ_{\mathscr{M}}(s) &= \bigcup_{\alpha \in Act} succ_{\mathscr{M}}(s, \alpha). \end{aligned}$$

We sometimes abbreviate pairs of action and distribution by $(\alpha, \mu) = \eta \in \mathscr{P}(s)$ for a state $s \in S$. In the following, we overload some notations we originally defined for (sub-)distributions such that they are also used for pairs of an action $\alpha \in Act$ and a (sub-)distribution $\mu \in subDistr(s)$. For instance, the support of a distribution μ is also defined if a specific action is given: $supp(\eta) = supp(\mu)$ for $\eta \in Act \times \{\mu\}$. We also call $(s, \eta, s') \in S \times \mathscr{P}(s) \times S$ a *branch of s*.

For a state $s \in S$, a successor state is determined as follows: First, $\eta \in \mathscr{P}(s)$ is chosen *nondeterministically*. Then, $s' \in supp(\eta)$ is determined *probabilistically* according to the distribution in η. This yields the transitions (s, η, s') to be taken from s. This process can be repeated infinitely often starting with an initial state s_I such that $I(s_I) > 0$.

We fix the set of actions that are *enabled* in state $s \in S$ of $\mathscr{M}$ by

$$Act_s^{\mathscr{M}} = \{\alpha \in Act \mid \exists \mu \in subDistr(S).\, (\alpha, \mu) \in \mathscr{P}(s)\}$$

In order to avoid deadlock states in the corresponding PA with *stochastic distributions*, we assume for a every PA $\mathscr{M}$ that $|Act_s^{\mathscr{M}}| > 0$ for all $s \in S$. If $|Act_s^{\mathscr{M}}| = 1$ for all $s \in S$ holds for an MDP $\mathscr{M}$, it behaves like a DTMC where the actions are just omitted.

Remark 7 *The Markov property holds also for PAs. After the nondeterministic choice has been made in state s, the probability to move to another state depends only on the chosen probability distribution and not on the history of already visited states.*

Example 3 *Consider a PA* $\mathscr{M} = (S, I, Act, \mathscr{P}, L)$ *with* $S = \{s_0, \ldots, s_8\}$, $I(s_0) = 1$ *and* $I(s) = 0$ *for all* $s \neq s_0$, $Act = \{\alpha, \beta\}$, $L(s_3) = \{\text{target}\}$ *and* $L(s) = \emptyset$ *for all* $s \neq s_3$. *This PA is an extension of the DTMC* $\mathscr{D}$ *from Example 1. In addition, there are nondeterministic choices for states* s_1 *and* s_2 *with* $\mathscr{P}(s_1) = \{(\alpha, \mu_1), (\beta, \mu_2)\}$ *and* $\mathscr{P}(s_2) = \{(\alpha, \mu_3), (\beta, \mu_4)\}$ *with*

$$\mu_1(s_2) = 0.5 \qquad \mu_1(s_3) = 0.5 \qquad \mu_2(s_2) = 1$$
$$\mu_3(s_1) = 0.5 \qquad \mu_3(s_4) = 0.5 \qquad \mu_4(s_1) = 1$$

For all other states s_i, $i \in \{0, 3, 4, 5, 6, 7, 8\}$, *it holds that* $Act_{s_i}^{\mathscr{M}} = \{\alpha\}$, *i. e., there is no nondeterminism involved. The PA* $\mathscr{M}$ *is graphically depicted in Figure 2.2. If there is only one action available in a state, we omit the action. Otherwise, there is a thick edge leading to the action from where on the probabilistic choice is depicted. Note that this PA is an MDP as for all states each action occurs at most once. We therefore also omit the distributions.*

In the following we often need the notion of a subsystem of a PA. The definition is analogous to DTMCs:

Definition 17 (Subsystem of a PA) *For a PA* $\mathscr{M} = (S, I, Act, \mathscr{P}, L)$, *a* subsystem of $\mathscr{M}$ *is a PA* $\mathscr{M}' = (S', I', Act, \mathscr{P}', L')$ *with*

- $S' \subseteq S$
- $I'(s) = I(s)$ *for all* $s \in S'$
- $\mathscr{P}'(s) = \{(\alpha, \mu) \in Act_s^{\mathscr{M}} \times subDistr(S') \mid \exists \mu' \in subDistr(S). (\alpha, \mu') \in \mathscr{P}(s) \wedge \mu \subseteq \mu' \wedge supp(\mu) = supp(\mu') \cap S'\}$ *for all* $s \in S'$
- $L'(s) = L(s)$ *for all* $s \in S'$.

By using the notion $\mu \subseteq \mu'$ for $(\alpha, \mu') \in \mathscr{P}(s)$, see the paragraph about distributions in Section 2.1, we ensure that $\mu(s) = \mu'(s)$ for all $s \in S'$ inside the subsystem.

Paths of PAs Paths are defined analogously to DTMCs. Assume in the following a PA $\mathscr{M} = (S, I, Act, \mathscr{P}, L)$.

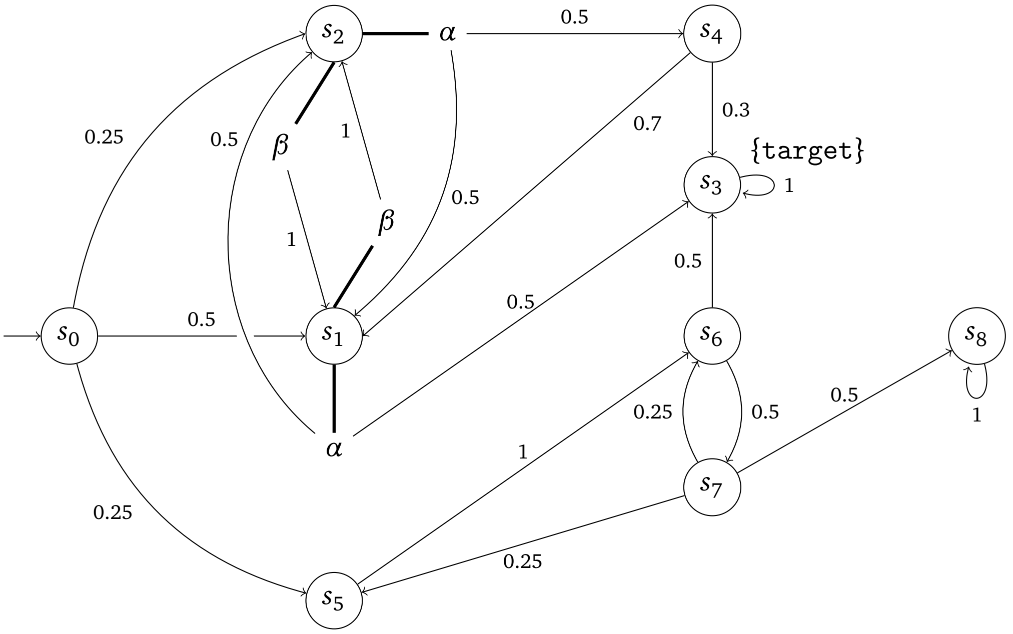


Figure 2.2: Example PA $\mathscr{M}$

Definition 18 (Path of a PA) *A path π of a PA $\mathscr{M}$ is a sequence $\pi = s_0\alpha_0\mu_0, s_1\alpha_1\mu_1, \ldots$ of states, actions, and distributions such that $s_{i+1} \in succ_{\mathscr{M}}(s_i, \alpha_i, \mu_i)$ for all $i \geq 0$.*

For a PA $\mathscr{M}$, the notations $Paths^{\mathscr{M}}_{fin}$, $Paths^{\mathscr{M}}_{fin}(s)$, $Paths^{\mathscr{M}}_{inf}$, $Paths^{\mathscr{M}}_{inf}(s)$ for $s \in S$, and $trace(\pi)$ for a path π of $\mathscr{M}$ are analogous to the definition for DTMCs, see the corresponding paragraph in Section 2.2.1. The same holds for notations concerning the concatenation or the first or last state of a path of a PA.

Remark 8 (Initial distribution vs. initial state for PAs) *As for DTMCs, we often refer to a PA $\mathscr{M} = (S, s_I, Act, \mathscr{P}, L)$ with a single initial state where $I(s_I) = 1$ holds. For arbitrary PAs, a similar construction as for DTMCs leads to PAs with a single initial state, see Remark 5. All our concepts are equally adaptable for PAs with single or multiple initial states.*

Probability measures on PAs For DTMCs, probabilities are defined on measurable sets of paths. In case of PAs, the nondeterminism has to be resolved to define a similar measure. This is done by so-called *schedulers*, often also referred to as *adversaries* or *strategies*.

Definition 19 (Scheduler) *A* scheduler *for a PA $\mathscr{M} = (S, I, Act, \mathscr{P}, L)$ is a function*

$$\sigma : Paths^{\mathscr{M}}_{fin}(s_I) \rightarrow Distr(Act \times subDistr(S))$$

for every $s_I \in S$ with $I(s_I) > 0$ such that $supp(\sigma(\pi)) \subseteq \mathcal{P}(last(\pi))$ for each $\pi \in Paths^{\mathcal{M}}_{fin}(s_I)$. We denote the set of schedulers on $\mathcal{M}$ by $Sched^{\mathcal{M}}$.

A scheduler resolves the nondeterministic choices depending on the finite path which led to the current state by assigning probabilities to the nondeterministic choices available in the last state of a finite path. It therefore transforms the nondeterministic model into a fully probabilistic one.

The resulting PA is deterministic with respect to the choice of actions, which semantically yields a DTMC whose states are the finite paths of the PA that start in the initial states.

Definition 20 (DTMC induced by a scheduler) *For a PA $\mathcal{M} = (S, I, Act, \mathcal{P}, L)$ and a scheduler σ for $\mathcal{M}$, the* DTMC induced by $\mathcal{M}$ and σ *is given by $\mathcal{M}_\sigma = \mathcal{D}_\sigma = (S_\sigma, I_\sigma, P_\sigma, L_\sigma)$ with*

- $S_\sigma = \bigcup_{s \in supp(I)} Paths^{\mathcal{M}}_{fin}(s)$
- $I_\sigma = I$
- $P_\sigma(\pi, \pi') = \begin{cases} \sigma(\pi)((\alpha, \mu)) \cdot \mu(s), & \text{if } \pi' = \pi(\alpha, \mu)s \\ 0, & \text{otherwise} \end{cases}$
- $L_\sigma(\pi) = L(last(\pi))$ *for all* $\pi \in S_\sigma$.

The probability measure for a PA can now be defined on this induced DTMC, which is in general countably infinite. For details we refer, e. g., to [dA97, BK08].

Remark 9 (Symbol for induced DTMC) *In general, for all MDPs and PAs we use the symbol $\mathcal{M}$ while we use $\mathcal{D}$ for DTMCs. However, to keep the reference to the PA we use both $\mathcal{M}_\sigma$ and $\mathcal{D}_\sigma$ for the DTMC that is induced by a PA or an MDP $\mathcal{M}$ and a scheduler σ.*

In the following, we do not need schedulers in their full generality. In many cases, a simple subclass of schedulers suffices, namely *memoryless deterministic schedulers*. The unique action-distribution pair assigned to a finite path by such a scheduler depends only on the last state of the path and is assigned via a *Dirac* distribution. Note that for finite PAs this yields a finite DTMC.

Definition 21 (Memoryless deterministic scheduler) *For a PA $\mathcal{M} = (S, I, Act, \mathcal{P}, L)$, a scheduler σ for $\mathcal{M}$ is* memoryless *iff for all $s \in supp(I)$ and $\pi, \pi' \in Paths^{\mathcal{M}}_{fin}(s)$ with $last(\pi) = last(\pi')$ we have that $\sigma(\pi) = \sigma(\pi')$.*

A scheduler σ for $\mathcal{M}$ is deterministic *iff for all $s \in supp(I)$, $\pi \in Paths^{\mathcal{M}}_{fin}(s)$ and $(\alpha, \mu) \in Act \times subDistr(S)$ we have that $\sigma(\pi)((\alpha, \mu)) \in \{0, 1\}$.*

Remark 10 (DTMC induced by a memoryless deterministic scheduler) *We can regard memoryless deterministic schedulers as functions $\sigma\colon S \to Act \times subDistr(S)$. The induced DTMC $\mathcal{M}_\sigma =$*

$\mathcal{D}_\sigma = (S_\sigma, I_\sigma, P_\sigma, L_\sigma)$ is given by $S_\sigma = S$, $I_\sigma = I$, $P_\sigma(s,s') = \mu(s')$ for $\sigma(s) = (\alpha, \mu) \in \mathcal{P}(s)$ and $L_\sigma = L$.

Example 4 (Scheduler) *Consider the PA $\mathcal{M}$ from Example 3. We define a memoryless deterministic scheduler σ with $\sigma(s) = \alpha$ for all $s \in S$. The induced DTMC $\mathcal{D}_\sigma$ equals the DTMC $\mathcal{D}$ from Example 1.*

2.2.3 Parametric Markov chains

To allow for DTMCs where not all probabilities are fixed numbers but *parameters* with unknown values, we follow the way of [HHZ10, Daw04] and use rational functions, see Definition 2, for defining probabilities.

Definition 22 (PDTMC) *A* parametric discrete-time Markov chain (PDTMC) *is a tuple $\mathcal{R} = (S, V, I, \mathfrak{P}, L)$ with:*

- *S a countable set of states*
- *$V = \{x_1, \ldots, x_n\}$ a finite set of parameters*
- *$I : S \to Rat_V$ an initial distribution*
- *$\mathfrak{P} : S \times S \to Rat_V$*
- *$L : S \to 2^{AP}$ a* labeling function.

Intuitively, for states $s, s' \in S$ with $\mathfrak{P}(s,s') = f$, the rational function $f \in Rat_V$ describes the probability of going from s to s'. Instantiating all parameters from V with respect to certain constraints yields a concrete probability. This is formalized later tailored to our algorithms.

The *underlying graph $\mathcal{G}_\mathcal{R} = (S, E_\mathcal{R})$ of a PDTMC $\mathcal{R}$* is defined analogously to DTMCs, see Definition 5, while the set of transitions $E_\mathcal{R}$ is defined for rational functions:

$$E_\mathcal{R} = \{(s,s') \in S \times S \mid \mathfrak{P}(s,s') \neq 0\}$$

Example 5 (PDTMC) *Consider the PDTMC in Figure 2.3 which is a parametrized version of the DTMC in Example 1. Parameters are p and q, leaving transitions probabilities outgoing from states s_0 and s_6 unspecified. Instantiating p by 0.25 and q by 0.25 yields the same DTMC as in Example 1.*

2.3 Specifications for probabilistic models

We now introduce the specifications we use throughout this thesis to define properties. We start with basic reachability properties followed by the probabilistic extension of computation tree logic, namely PCTL. We conclude with ω-regular properties.

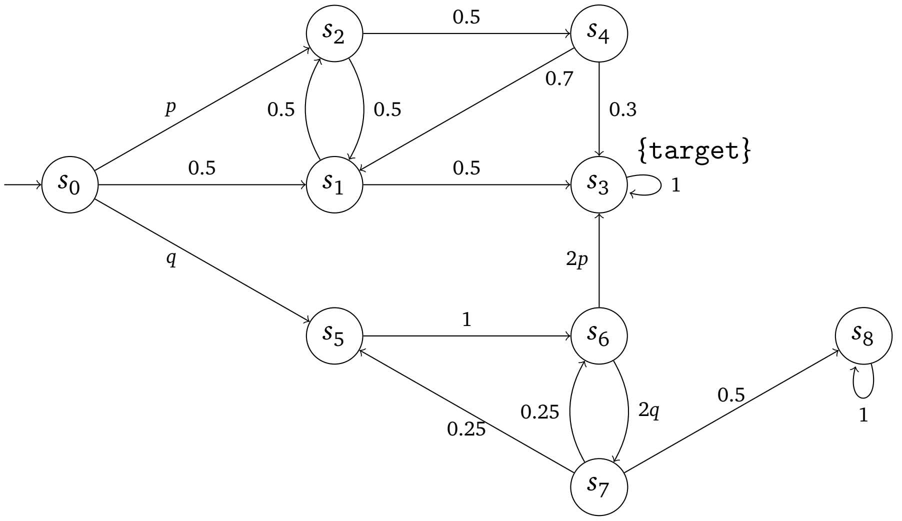


Figure 2.3: Example PDTMC with parameters p and q

2.3.1 Reachability properties

Many important problems when analyzing probabilistic models as presented in Section 2.2 can be reduced to so-called *reachability properties*. We are interested in a quantitative analysis such as:

> "*What is the probability to reach a certain set of states T starting in state s?*"

Such a set of *target states T* might, e. g., represent a *bad event* in the model at hand and should only occur with a certain small probability. Formally, we formulate properties like $\mathbb{P}_{\bowtie\lambda}(\lozenge \mathsf{target})$ for $\bowtie \in \{<, \leq, =, >, \geq\}$, $\lambda \in [0,1] \subseteq \mathbb{R}$, $\mathsf{target} \in AP$ and $T = \{s \in S \mid \mathsf{target} \in L(s)\}$. Note that in fact λ is from $[0,1] \cap \mathbb{Q}$ as no real-valued probability bound can be used in practice. $\mathbb{P}$ measures the probability for the property and is formalized later. For example, $\mathbb{P}_{\leq 0.1}(\lozenge \mathsf{target})$ means that the probability of reaching a state labeled with target has to be less than or equal to 0.1. If the probability is higher at state s, this property evaluates to `false` for this state. We also directly write $\mathbb{P}_{\bowtie\lambda}(\lozenge T)$ for these properties, thereby implying that the set T consists exactly of the states that are labeled with target. Moreover, we restrict the properties to the form $\mathbb{P}_{\leq\lambda}(\lozenge T)$ and sometimes $\mathbb{P}_{<\lambda}(\lozenge T)$. The cases $\geq$ and $>$ can be reduced to $\leq$ and $<$ using negation and the complement probabilities, e. g., $\mathbb{P}_{>\lambda}(\lozenge T)$ is equivalent to $\mathbb{P}_{\leq 1-\lambda}(\lozenge \neg T)$.

In the following we explain how such properties can be verified for DTMCs and PAs. For PDTMCs, this is still ongoing research. We present some details later in this thesis.

2.3.1.1 Model checking reachability properties on DTMCs

Assume in the following a DTMC $\mathcal{D} = (S, s_I, P, L)$ with a single initial state and a set of target states $T = \{t \in S \mid \mathsf{target} \in L(t)\}$ uniquely labeled by $\mathsf{target} \in AP$. To measure the probability of finally reaching a state from T when starting at a specific state $s \in S$, we consider all infinite paths in $\mathit{Paths}^{\mathcal{D}}_{\mathit{inf}}(s)$ that contain a state from T. Formally, the set of paths that contribute to the probability of reaching T from a state s is given by

$$\lozenge T(s) = \{\pi \in \mathit{Paths}^{\mathcal{D}}_{\mathit{inf}}(s) \mid \exists i.\, \mathsf{target} \in L(\pi(i))\}$$

where we overload $\lozenge T$ to both denote a set of paths and a reachability property. We sometimes also omit s, if it is clear from the context, e. g., if s is the initial state of a DTMC.

The set $\lozenge T(s)$ it measurable, as it corresponds to the union of all cylinder sets of finite paths from $\mathit{Paths}^{\mathcal{D}}_{\mathit{fin}}(s,T)$:

$$\lozenge T(s) = \bigcup_{\pi \in \mathit{Paths}^{\mathcal{D}}_{\mathit{fin}}(s,T)} \mathit{cyl}(\pi)$$

In order to compute probabilities, we first observe that for $\pi, \pi' \in \mathit{Paths}^{\mathcal{D}}_{\mathit{fin}}(s,T)$ with $\pi \in \mathit{pref}_{\mathcal{D}}(\pi')$ it holds that $\mathit{cyl}(\pi') \subseteq \mathit{cyl}(\pi)$. However, if π and π' are not prefixes of each other then $\mathit{cyl}(\pi) \cap \mathit{cyl}(\pi') = \emptyset$. Thus we can restrict the considered paths to the ones that end when first visiting a target state:

$$\lozenge T_{\mathrm{fin}}(s) = \{\pi \in \mathit{Paths}^{\mathcal{D}}_{\mathit{fin}}(s,T) \,\Big|\, \forall 0 \le i < |\pi|.\, \pi(i) \notin T\}$$

As no path is a prefix of another one in this set, the probability of this set can be computed by the sum of the path probabilities using the measure for finite paths, see Section 2.2.1:

$$\mathit{Pr}^{\mathcal{D}}_{s}(\lozenge T(s)) = \sum_{\pi \in \lozenge T_{\mathit{fin}}(s)} \mathbf{P}(\pi)$$

Having a measurable set, we recall the property at hand: $\varphi = \mathbb{P}_{\bowtie\lambda}(\lozenge T)$ with $\bowtie \in \{<, \le, >, \ge\}$, $\lambda \in [0,1] \subseteq \mathbb{R}$ and $T \subseteq S$. We use the notation $\mathcal{D} \models \mathbb{P}_{\le\lambda}(\lozenge T)$ to express that $\mathit{Pr}^{\mathcal{D}}_{s_I}(\lozenge T(s_I))$ is less than or equal to the bound $\lambda \in [0,1] \subseteq \mathbb{R}$.

Intuitively, the probability of all paths starting in the initial state s_I has to be inside the interval defined by $\bowtie\lambda$. Prior to checking $\mathcal{D} \models \mathbb{P}_{\le\lambda}(\lozenge T)$, the states that cannot reach a target state can be safely removed from the DTMC $\mathcal{D}$.

Definition 23 (Relevant states of a DTMC) *Given a DTMC $\mathcal{D} = (S, s_I, P, L)$ and a set of target states T, a state $s \in S$ is called* relevant for $\mathcal{D}$ and T *if*

$$\mathit{Paths}^{\mathcal{D}}_{\mathit{fin}}(s,T) \neq \emptyset.$$

By $S^{\mathcal{D}}_{\mathit{rel}(T)}$ we denote the set of relevant states for $\mathcal{D}$ and T. States from $S \setminus S^{\mathcal{D}}_{\mathit{rel}(T)}$ are called irrelevant *for $\mathcal{D}$ and T.*

The set $S^{\mathcal{D}}_{\mathrm{rel}(T)}$ can be determined in linear time by a backward reachability analysis from the target states, if we assume the state space to be finite.

The property $\mathbb{P}_{\leq\lambda}(\lozenge T)$ for DTMC $\mathscr{D}$ is model checked by computing $Pr_s^{\mathscr{D}}(\lozenge T)$ for all states $s \in S$ and comparing this probability for the initial state with the bound λ. The probabilities $p_s = Pr_s^{\mathscr{D}}(\lozenge T)$ are obtained as the unique solution of the following linear equation system [BK08, p. 760]:

$$p_s = \begin{cases} 1 & \text{if } s \in T \\ 0 & \text{if } s \notin S^{\mathscr{D}}_{\mathrm{rel}(T)} \\ \sum_{s' \in S} P(s,s') \cdot p_{s'} & \text{otherwise.} \end{cases}$$

For some applications it might be necessary to remove all irrelevant states together with their incident transitions. In this case, $p_s > 0$ holds for all remaining states.

2.3.1.2 Model checking reachability properties on PAs

For PAs, computing reachability probabilities gives rise to the question how the nondeterminism should be resolved. For our setting, we require that a reachability property has to hold *for all possible schedulers*.

Formally, for a formula $\varphi = \mathbb{P}_{\leq\lambda}(\lozenge T)$ and a PA $\mathscr{M}$ we have $\mathscr{M} \models \varphi$ iff for all schedulers $\sigma \in Sched^{\mathscr{M}}$ it holds that $\mathscr{M}_\sigma \models \varphi$, i. e., in the induced DTMC $\mathscr{M}_\sigma$ the property is satisfied.

As a first step for verifying reachability properties on PAs, we identify again the set of relevant states. For the complementary set of irrelevant states the reachability probability is 0.

Definition 24 (Relevant states of PAs) *Let $\mathscr{M} = (S, s_I, Act, \mathscr{P}, L)$ be a PA and $T \subseteq S$ a set of target states. Then*

$$S^{\mathscr{M}}_{rel(T)} = \{s \in S \mid \exists \sigma \in Sched_{\mathscr{M}}.Pr_s^{\mathscr{M}_\sigma}(\lozenge T(s)) > 0\}$$

is the set of relevant states *for T. If $s \notin S^{\mathscr{M}}_{rel(T)}$ then s is called* irrelevant *for T.*

Intuitively, the irrelevant states for T are the ones for which no scheduler exists such that a state from T is reachable. These states can be computed in linear time by a backward reachability analysis on the PA $\mathscr{M}$ with a finite state space [BK08, Algorithm 46]. The removal of irrelevant states does not affect the reachability probabilities. To check whether $\mathscr{M}_\sigma \models \mathbb{P}_{\leq\lambda}(\lozenge T)$ holds for all schedulers σ of the PA $\mathscr{M}$, it suffices to consider a memoryless deterministic scheduler σ^* which maximizes the reachability probability for $\lozenge T$ and check whether $Pr_{s_I}^{\mathscr{M}_{\sigma^*}}(\lozenge T) \leq \lambda$ [BK08, Lemma 10.102], i. e., if the probability bound is exceeded in the DTMC that is induced by σ^*. We call this scheduler σ^* a *maximizing scheduler* for $\mathscr{M}$ and T. The maximal probabilities

$p_s = Pr_s^{\mathcal{M}_{\sigma^*}}(\Diamond T)$ for each $s \in S$ can be characterized by the following equation system:

$$p_s = \begin{cases} 1 & \text{if } s \in T \\ 0 & \text{if } s \notin S^{\mathcal{M}}_{\mathrm{rel}(T)} \\ \max\{\sum\limits_{s' \in S} \mu(s') \cdot p_{s'} \mid \exists \alpha.\, (\mu, \alpha) \in \mathscr{P}(s)\} & \text{otherwise.} \end{cases}$$

This equation system can for instance be solved by using value iteration or policy iteration. It can also be transformed into a linear optimization problem that yields the maximal reachability probability together with an optimal scheduler [BK08, Theorem 10.105]. In this case the probability of irrelevant states needs to be explicitly set to 0.

2.3.2 Probabilistic computation tree logic

Although we mainly use reachability properties throughout this thesis, for the sake of completeness we introduce the *probabilistic computation tree logic* (PCTL) [HJ94] which extends the classical *computation tree logic* (CTL) by the quantitative operator $\mathbb{P}$. As CTL, also PCTL defines state and path formulas. Intuitively, a state formula φ is interpreted on the states of a DTMC and a path formula ψ is interpreted over infinite paths of a DTMC.

Definition 25 (Syntax of PCTL) *Let AP be a set of atomic propositions.* PCTL formulae *over AP are built according to the following context-free grammar, starting with symbol φ:*

$$\begin{aligned} \varphi &::= \mathtt{true} \mid a \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathbb{P}_J(\psi) \\ \psi &::= \bigcirc\varphi \mid \varphi\,\mathscr{U}^{\leq n}\varphi \mid \varphi\,\mathscr{U}\,\varphi \end{aligned}$$

for $J \subseteq [0,1] \subseteq \mathbb{R}$ with rational bounds and $n \in \mathbb{N}$. The formula φ is called a PCTL state formula *and ψ is called a PCTL* path formula.

We can abbreviate $\mathbb{P}_{[0,\lambda]}(\varphi) = \mathbb{P}_{\leq\lambda}(\varphi)$ and $\mathbb{P}_{]\lambda,1]}(\varphi) = \mathbb{P}_{>\lambda}(\varphi)$ for $\lambda \in [0,1] \subseteq \mathbb{R}$. This is done analogously for $<$ and $\geq$. As syntactic sugar for PCTL path formulae φ we derive the "finally"-operator $\Diamond$ as $\mathbb{P}_{\lhd\lambda}(\Diamond\varphi) = \mathbb{P}_{\lhd\lambda}(\text{true}\,\mathscr{U}\,\varphi)$ and the "globally"-operator $\Box$ as $\mathbb{P}_{\lhd\lambda}(\Box\varphi) = \mathbb{P}_{\rhd 1-\lambda}(\text{true}\,\mathscr{U}\,\neg\varphi)$ where $\rhd$ is $>$, $\geq$, $\leq$, $<$ if $\lhd$ is $<$, $\leq$, $\geq$, $>$, respectively.

We now shortly explain the semantics of PCTL formulae for a DTMC $\mathscr{D}$. For details we refer to [BK08, Chapter 10]. Consider in the following a DTMC $\mathscr{D} = (S, s_I, P, L)$. Formally, the satisfaction of a PCTL path formula for an *infinite* path $\pi \in Paths^{\mathscr{D}}_{inf}$ is defined as follows:

$$\begin{aligned} \mathscr{D}, \pi \models \bigcirc\varphi \quad &\Longleftrightarrow \quad \mathscr{D}, \pi(1) \models \varphi \\ \mathscr{D}, \pi \models \varphi_1 \mathscr{U} \varphi_2 \quad &\Longleftrightarrow \quad \exists i \geq 0.\, (\mathscr{D}, \pi(i) \models \varphi_2 \wedge \forall 0 \leq k < i.\, \mathscr{D}, \pi(i) \models \varphi_1) \\ \mathscr{D}, \pi \models \varphi_1 \mathscr{U}^{\leq n} \varphi_2 \quad &\Longleftrightarrow \quad \exists i \geq 0.\, (i \leq n \wedge \mathscr{D}, \pi(i) \models \varphi_2 \wedge \forall 0 \leq k < i.\, \mathscr{D}, \pi(i) \models \varphi_1) \end{aligned}$$

Consider a finite path $\pi' \in \mathit{Paths}^{\mathscr{D}}_{\mathit{fin}}$. If for all infinite paths π that are contained in its cylinder, i. e., $\pi \in \mathit{cyl}(\pi')$, and for a PCTL path formula ψ it holds that $\pi \models \psi$, we also write $\pi' \models \psi$. Thereby, PCTL path formulae can also be satisfied by finite paths.

We define a state s of the DTMC $\mathscr{D}$ to satisfy a PCTL state formula φ, written $\mathscr{D}, s \models \varphi$, recursively as follows:

$$\begin{array}{lcl}
\mathscr{D}, s \models \text{true} & & \text{always,} \\
\mathscr{D}, s \models a & \iff & a \in L(s) \\
\mathscr{D}, s \models \neg\varphi & \iff & \mathscr{D}, s \not\models \varphi \\
\mathscr{D}, s \models (\varphi_1 \wedge \varphi_2) & \iff & \mathscr{D}, s \models \varphi_1 \text{ and } \mathscr{D}, s \models \varphi_2 \\
\mathscr{D}, s \models \mathbb{P}_J(\psi) & \iff & \Pr^{\mathscr{D}}\big(\{\pi \in \mathit{Paths}^{\mathscr{D}}_{\mathit{inf}}(s) \mid \mathscr{D}, \pi \models \psi\}\big) \in J.
\end{array}$$

A DTMC $\mathscr{D}$ satisfies a PCTL state formula φ, written $\mathscr{D} \models \varphi$, if its initial state s_I does, i. e., if $\mathscr{D}, s_I \models \varphi$.

For PAs, the semantics is again defined for *all* possible schedulers, i. e., a PCTL formula φ holds for a PA $\mathscr{M}$ iff it holds for the DTMCs $\mathscr{M}_\sigma$ with respect to all schedulers σ for $\mathscr{M}$.

2.3.3 ω-regular properties

In this section we introduce *ω-regular properties* for DTMCs and PAs and the corresponding model checking procedures. We follow the standard automata-theoretic approach, as described, e. g., in [dA97, Var99, CSS03, BK08]. As a formalism for describing these properties, we introduce deterministic Rabin automata. We expect the reader to be familiar with standard notions of automata theory, for an introduction to this topic we refer to the standard textbook of Hopcroft and Ullman [HJ79].

Remark 11 (Rabin automata vs. Büchi automata) *The question might arise why as formalism for ω-regular properties deterministic Rabin automata are chosen instead of non-deterministic Büchi automata (NBAs), since a DRA describing the same language as an NBA can be exponentially larger. The problem lies in the nondeterminism of the NBA. Building a product automaton would not result in a model like a DTMC or a PA as two different kinds of nondeterminism would have to be considered: On the one hand—in case of PAs—the choice of the probability distribution and on the other hand the nondeterministic choice of the symbol to read for the NBA which would result in a nondeterministic choice of atomic propositions.*

Let us first define arbitrary *linear-time properties*.

Definition 26 (Linear time property) *Assume a set of atomic propositions AP. A* linear time property *over AP is a set $\mathscr{L}$ of traces $\gamma_0\gamma_1\gamma_2\ldots$ with $\gamma_i \subseteq AP$ for $i \in \mathbb{N}$.*

Intuitively, a linear time property is a set of infinite traces. In this thesis we deal with a certain class, namely the *ω-regular properties*. Note that the reachability properties as introduced in Section 2.3.1 build a simple subclass of ω-regular properties.

Definition 27 (Deterministic Rabin automaton) *A* deterministic Rabin automaton (DRA) *is a tuple $\mathscr{A} = (Q, q_I, \Sigma, \delta, F)$ with*

- *Q a finite nonempty set of states*
- *$q_I \in Q$ an initial state*
- *Σ a finite input alphabet*
- *$\delta : Q \times \Sigma \to Q$ a transition function*
- *$F \subseteq 2^Q \times 2^Q$ an acceptance condition.*

A *run* r of $\mathscr{A}$ is a state sequence $q_0, q_1, q_2, \ldots \in Q^\omega$ with $q_0 = q_I$ such that for all $i \geq 0$ there is a $\gamma_i \in \Sigma$ with $q_{i+1} = \delta(q_i, \gamma_i)$. We say that r is the run of $\mathscr{A}$ on the infinite word $w = \gamma_0\gamma_1 \ldots$ over Σ. Note that for every infinite word $w \in \Sigma^\omega$ the run of $\mathscr{A}$ on w is unique as this is a *deterministic* automaton. By $\inf(r)$ we denote the set of all states which appear infinitely often in the run r. Given the acceptance condition $F = \big\{(R_i, A_i) \,|\, i = 1, \ldots, n\big\}$, a run r is *accepting* if there exists an $i \in \{1, \ldots, n\}$ with $\inf(r) \cap R_i = \emptyset$ and $\inf(r) \cap A_i \neq \emptyset$. Intuitively this means that at least some state from A_i has to be visited infinitely often while the corresponding set R_i is visited only finitely often by an accepting run of $\mathscr{A}$.

We call the set of all infinite words over Σ that have an accepting run on $\mathscr{A}$ the *language of* $\mathscr{A}$, denoted by $\mathscr{L}(\mathscr{A})$. The class of ω-regular properties is defined by using this notion:

Definition 28 (ω-Regular property, Safra [Saf89]) *A linear-time property $\mathscr{L}$ is* ω-regular *if and only if there is a DRA $\mathscr{A}$ with $\mathscr{L} = \mathscr{L}(\mathscr{A})$.*

Assume a set *AP* of atomic propositions, a DRA $\mathscr{A}$ with alphabet 2^{AP} and the ω-regular property $\mathscr{L} = \mathscr{L}(\mathscr{A})$. Intuitively, a path π of a DTMC $\mathscr{D}$ satisfies $\mathscr{L}$ if the run of $\mathscr{A}$ on *trace*(π) is accepting. Formally, the following set of paths of $\mathscr{D}$ that start in s satisfies $\mathscr{L}$:

$$\mathscr{L}_s^{\mathscr{D}} = \{\pi \in Paths_{inf}^{\mathscr{D}}(s) \mid trace(\pi) \in \mathscr{L}\}.$$

For each ω-regular property $\mathscr{L}$ and DTMC $\mathscr{D}$, this set of paths is measurable in the probability space defined in Section 2.2.2, see [Var85]. As a consequence, we measure the probability $Pr_{s_I}^{\mathscr{D}}(\mathscr{L}_{s_I}^{\mathscr{D}})$, i. e., the probability of all paths that start in the initial state and whose traces satisfy the ω-regular property, against the bound λ:

$$\mathscr{D} \models \mathbb{P}_{\leq\lambda}(\mathscr{L}) \quad \Leftrightarrow \quad Pr_{s_I}^{\mathscr{D}}(\mathscr{L}_{s_I}^{\mathscr{D}}) \leq \lambda$$

2.3.3.1 Model checking ω-regular properties on DTMCs

Let in the following $\mathscr{D} = (S, s_I, P, L)$ be a DTMC. We consider an ω-regular property $\mathscr{L}$ and assume that a DRA $\mathscr{A} = (Q, q_I, \Sigma, \delta, F)$ with $\Sigma = 2^{AP}$ and $\mathscr{L} = \mathscr{L}(\mathscr{A})$ is given.

To compute the probability of $\mathscr{L}$ for a DTMC $\mathscr{D}$ we build the product automaton of the DTMC $\mathscr{D}$ and the DRA $\mathscr{A}$. Within this product automaton, which is again a DTMC, this problem reduces to computing reachability probabilities as introduced in Section 2.3.1.

Definition 29 (Product automaton) *Let $\mathscr{D} = (S, s_I, P, L)$ be a DTMC and let $\mathscr{A} = (Q, q_I, 2^{AP}, \delta, F)$ be a DRA with $F = \{(R_i, A_i) \,\big|\, i = 1, \ldots, n\}$. The* product automaton *of $\mathscr{D}$* and *$\mathscr{A}$ is a DTMC $\mathscr{D} \otimes \mathscr{A} = (S \times Q, (s,q)_I, P', L')$ over the set AP' of atomic propositions such that:*

- $(s,q)_I = \big(s_I, \delta(q_I, L(s_I))\big)$
- $P'\big((s,q),(s',q')\big) = \begin{cases} P(s,s') & \text{if } q' = \delta\big(q, L(s')\big) \\ 0 & \text{otherwise} \end{cases}$
- $AP' = \{R_i, A_i \mid i = 1, \ldots, n\}$
- $A_i \in L'(s,q)$ *iff* $q \in A_i$, *and* $R_i \in L'(s,q)$ *iff* $q \in R_i$, *for* $i = 1, \ldots, n$

Intuitively, in the DTMC resulting from this product construction, states whose DRA component is from A_i or R_i are labeled with this label enabling to measure the probability of reaching such states.

The next step is to determine the BSCCs of the product automaton. According to the labels of these absorbing states sets, we define a sort of target states.

Definition 30 (Accepting BSCC) *Let $\mathscr{D} = (S, s_I, P, L)$ be a DTMC, $\mathscr{A} = (Q, q_I, 2^{AP}, \delta, F)$ be a DRA and $\mathscr{D} \otimes \mathscr{A} = (S \times Q, (s,q)_I, P', L')$ be the product automaton of $\mathscr{D}$ and $\mathscr{A}$. A BSCC $B \subseteq S \times Q$ of $\mathscr{D} \otimes \mathscr{A}$ is called* accepting *if there is $(R_i, A_i) \in F$ such that $A_i \in L'(s,q)$ for some $(s,q) \in B$ and $R_i \notin L'(s',q')$ for all $(s',q') \in B$. The set of* all accepting BSSCs *for $\mathscr{D}$ and $\mathscr{A}$ is denoted by $\mathscr{B}$.*

Intuitively, for an accepting BSCC at least one state has to be labeled with an A_i while no state must be labeled with R_i. For convenience, we introduce the proposition *accept* and extend the labeling by $accept \in L'(s,q)$ iff (s,q) is a state in an accepting BSCC of $\mathscr{D} \otimes \mathscr{A}$. Then the following theorem holds:

Theorem 1 ([dA97]) *Let $\mathscr{D}$ be a DTMC, $\mathscr{L}$ an ω-regular property, and $\mathscr{A}$ a DRA with $\mathscr{L} = \mathscr{L}(\mathscr{A})$. Then:*

$$Pr^{\mathscr{D}}_{s_I}(\mathscr{L}^{\mathscr{D}}_{s_I}) = Pr^{\mathscr{D}\otimes\mathscr{A}}_{(s,q)_I}(\lozenge accept)$$

Using this theorem, the computation of the probability for an ω-regular property is reduced to computing the probability of reaching an accepting BSCC in the corresponding product automaton. This gives us a model checking procedure.

2.3.3.2 Model checking ω-regular properties on PAs

As for other properties, a PA $\mathcal{M}$ satisfies the property $\mathbb{P}_{\leq\lambda}(\mathcal{L})$ iff the property is satisfied for all schedulers, i. e., if $\mathcal{M}_\sigma \models \mathbb{P}_{\leq\lambda}(\mathcal{L})$ for all $\sigma \in Sched_{\mathcal{M}}$.

Analogously to DTMCs, checking the property $\mathcal{L}$ for a PA $\mathcal{M}$ can be carried out by building the product automaton of the PA $\mathcal{M}$ with the DRA $\mathcal{A}$ and computing reachability probabilities in the resulting PA.

Definition 31 (Product automaton) *Let $\mathcal{M} = (S, s_I, Act, \mathcal{P}, L)$ be a PA and $\mathcal{A} = (Q, q_I, 2^{AP}, \delta, F)$ a DRA with $F = \{(R_i, A_i) \mid i = 1, \ldots, n\}$. The* product automaton *of $\mathcal{M}$* and *$\mathcal{A}$ is a PA $\mathcal{M} \otimes \mathcal{A} = (S \times Q, (s,q)_I, Act, \mathcal{P}', L')$ over the set AP' of atomic propositions such that*

- $(s,q)_I = \big(s_I, \delta(q_I, L(s_I))\big)$,
- $\mathcal{P}'\big((s,q)\big) = \{(\alpha,\mu) \in Act \times subDistr(S \times Q) \mid \exists(\alpha,\mu') \in \mathcal{P}(s)$ *with* $\mu(s', \delta(q, L(s'))) = \mu'(s)$ *for all* $s' \in S\}$
- $AP' = \{R_i, A_i \mid i = 1, \ldots, n\}$, *and*
- $A_i \in L'(s,q)$ *iff* $q \in A_i$, *and* $R_i \in L'(s,q)$ *iff* $q \in R_i$, *for* $i = 1, \ldots, n$.

Again, we need to consider strongly connected components inside the product where the limit behavior with respect to the acceptance condition is mirrored. However, as the product here is a PA, an SCC depends on the choice of scheduler, i. e., a scheduler has to induce an SCC in the induced DTMC. We introduce the notion of so-called *end components*.

Definition 32 (Sub-PA) *Let $\mathcal{M} = (S, s_I, Act, \mathcal{P}, L)$ be a PA. A* sub-PA *of $\mathcal{M}$ is a pair $\mathcal{E} = (S', A)$ with a non-empty set of states $S' \subseteq S$ and a function $A \colon S' \to 2^{Act \times subDistr(S)} \setminus \emptyset$ such that $succ_{\mathcal{M}}(s, \alpha, \mu) \subseteq S'$ holds for all $s \in S'$ and $(\alpha, \mu) \in A(s) \subseteq \mathcal{P}(s)$.*

A sub-PA is a subset of states such that for the pairs of actions and distributions at state s given by $A(s)$ all successors are again within this subset, i. e., it is closed under nondeterministic choices according to A.

Definition 33 (Accepting end component) *Given a PA $\mathcal{M} = (S, I, Act, \mathcal{P}, L)$, a sub-PA $\mathcal{E} = (S', A)$ is an* end component *of $\mathcal{M}$ if the directed graph $G = (S', V)$ with $V = \big\{(s, s') \in S' \times S' \mid \exists(\alpha,\mu) \in A(s). s' \in succ_{\mathcal{M}}(s, \alpha, \mu)\big\}$ is strongly connected and $\mu \in Distr(S')$ for all $s \in S'$ and $\alpha \in A(s)$.*

Let $\mathcal{A} = (Q, q_I, 2^{AP}, \delta, F)$ be a DRA with $F = \{(R_i, A_i) \mid i = 1, \ldots, n\}$ and $\mathcal{E} = (S', A)$ an end component of the product automaton $\mathcal{M} \otimes \mathcal{A} = (S \times Q, (s,q)_I, Act, \mathcal{P}', L')$. $\mathcal{E}$ is accepting *if there is an $i \in \{1, \ldots, n\}$ such that for all $(s,q) \in S'$ it holds that $R_i \notin L'(s,q)$ and $\exists(s,q) \in \mathcal{E}. A_i \in L'(s,q)$.*

Intuitively, $\mathcal{E}$ is an end component if there is a memoryless scheduler σ such that $\mathcal{E}$ is a BSCC of the induced DTMC. An end component is accepting if there is a pair $(R_i, A_i) \in F$ such that the label A_i occurs in the end component while R_i does not. Note that by explicitly requiring that the distributions are not sub-stochastic, it is ensured that the probability of staying in these BSCCs is always 1. Furthermore, an end component (S', A') is called *maximal*, if there is no other end component (S'', A'') such that $(S', A') \neq (S'', A'')$ with $S' \subseteq S''$ and $A'(s) \subseteq A''(s)$ for all $s \in S$.

To determine whether $\mathbb{P}_{\leq\lambda}(\mathscr{L})$ is satisfied by $\mathscr{M}$, it suffices to compute a maximizing scheduler σ^* as described in Section 2.3.1.2 and consider the induced DTMC $\mathscr{M}_{\sigma^*}$:

$$Pr_{s_I}^{\mathscr{M}_{\sigma^*}}(\mathscr{L}_{s_I}^{\mathscr{M}_{\sigma^*}}) = \max_{\sigma \in Sched_{\mathscr{M}}} Pr_{s_I}^{\mathscr{M}_\sigma}(\mathscr{L}_{s_I}^{\mathscr{M}_\sigma}) \leq \lambda$$

We extend the labeling of $\mathscr{M} \otimes \mathscr{A}$ such that *accept* $\in L'(s, q)$ iff (s, q) belongs to an accepting end component. The model checking is now again reduced to compute a reachability probability with respect to accepting end components:

Theorem 2 ([dA97]) *Let $\mathscr{M}$ be a PA, $\mathscr{L}$ an ω-regular property and $\mathscr{A}$ a DRA with $\mathscr{L}(\mathscr{A}) = \mathscr{L}$. Then:*

$$Pr_{s_I}^{\mathscr{M}_{\sigma^*}}(\mathscr{L}_{s_I}^{\mathscr{M}_{\sigma^*}}) = Pr_{(s,q)_I}^{\mathscr{M}_{\sigma^*} \otimes \mathscr{A}}(\Diamond accept) .$$

To determine the relevant states of $\mathscr{M} \otimes \mathscr{A}$, we compute its maximal end components. This can be done efficiently [CH11]. States from which a maximal end component containing a state in $\bigcup_{i=1}^{n} A_i$ is reachable under at least one scheduler, are relevant. Strictly speaking, this condition is not sufficient since end components additionally have to satisfy a condition on the R_i-states to be accepting. However, exactly identifying the relevant states would require to determine all end components, which is in general computationally infeasible. Therefore we always use an over-approximation of the relevant states where no state from R_i is reachable.

2.4 Symbolic graph representations

In this section we give a short introduction on how we represent discrete-time Markov chains by means of symbolic data structures. The key idea behind these *implicit* representations is that we have a function and a variable encoding of states and transitions such that the function evaluates to `true` for an assignment of variables if and only if this assignment corresponds to a state or transition encoding. We also use symbolic representations to encode whole state and transition sets, e. g., paths of a graph. Symbolic representations are in practice often smaller by orders of magnitude than explicit ones and allow to reduce not only the memory consumption but also the computational costs for operations on the data structures.

we introduce *binary decision diagrams* (BDDs) [Bry86] and *multi-terminal binary decision diagrams* (MTBDDs) [FMY97]. (MT)BDDs have been applied very successfully for the verification of digital circuits [BCM$^+$92]. In the past, approaches have been extended to the verification of probabilistic systems [KNP02, HKN$^+$03, Par02]. Symbolic graph representations have the drawback that for some systems the representation is large (e. g., for multiplier circuits [Bry91]), and that their size can strongly depend on the ordering of the variables. An optimal ordering, however, is hard to find [BW96], but good heuristics are available [Rud93, Par02].

2.4.1 Ordered binary decision diagrams

Definition 34 (BDD and MTBDD) *Let Var be a set of Boolean variables. A* binary decision diagram *(BDD) over Var is a rooted, acyclic, directed graph* $B = (V, n_{\text{root}}, E)$ *with*

- V *a finite set of nodes*
- $n_{\text{root}} \in V$ *the root node*
- $E \subseteq V \times V$ *the set of edges*

Each node is either an inner node *with two outgoing edges or a* leaf node *with no outgoing edges, denoted by* $V_{inner} \subseteq V$ *or* $V_{leaf} \subseteq V$*, respectively. Let the two functions* $label_{inner}\colon V_{inner} \to Var$ *and* $label_{leaf}\colon V_{leaf} \to \{0,1\}$ *define a* BDD-labeling*. The two* successor *nodes of inner nodes* $n \in V$ *are denoted by* $hi(n)$ *and* $lo(n)$*.*

A multi-terminal binary decision diagram *(MTBDD) is defined similar to an BDD with the exception that the MTBDD labeling for leaf nodes is given by* $label_{leaf}\colon V_{leaf} \to \mathbb{R}$*.*

An (MT)BDD is called ordered *if there is a linear order* $< \subseteq Var \times Var$ *on the set of variables such that for all inner nodes n either* $hi(n)$ *is a leaf node or* $label_{inner}(n) < label_{inner}\big(hi(n)\big)$*, and the same for* $lo(n)$*. The relation* $<$ *is called a* variable ordering*. Ordered (MT)BDDs are denoted by* $B = (V, n_{\text{root}}, E, <)$*.*

In the following, we assume all (MT)BDDs to be ordered. Intuitively, BDDs represent Boolean formulae. As all inner nodes are labeled by one variable from *Var*, a path from the root to a leaf induces a partial assignment of the variables. The assignment is satisfying for the represented Boolean formula, iff the path ends in a leaf labeled by 1.

More formally, let B be a BDD over *Var* and $\mathcal{V}(Var) = \big\{\nu : Var \to \{0,1\}\big\}$ the set of all variable valuations. Each $\nu \in \mathcal{V}(Var)$ induces a path in B from the root to a leaf: If an inner node $n \in V$ is labeled by $label_{inner}(n) = \sigma \in Var$ and we have the assignment $\nu(\sigma) = 1$, the path takes the edge $hi(n)$ and vice versa for $\nu(\sigma) = 0$ and $lo(n)$.

A BDD B represents a function $f_B\colon \mathcal{V}(Var) \to \{0,1\}$ assigning to each assignment $\nu \in \mathcal{V}(Var)$ the label of the leaf node reached in B by the path induced by ν. Analogously, each MTBDD B represents a function $f_B\colon \mathcal{V}(Var) \to \mathbb{R}$.

An (MT)BDD is *reduced* if all functions rooted at different nodes are different. For a fixed variable ordering, reduced (MT)BDDs are canonical data structures for representing functions $f : \mathcal{V}(Var) \to \{0,1\}$ resp. $f : \mathcal{V}(Var) \to \mathbb{R}$ [Bry86]. In the following we assume all (MT)BDDs to be reduced and ordered with respect to a fixed variable order.

By Var' we denote the variable set *Var* with each variable $x \in Var$ renamed to some $x' \in Var'$

such that $\mathit{Var} \cap \mathit{Var}' = \emptyset$ and $x_1 \neq x_2 \Rightarrow x_1' \neq x_2'$. Our algorithms use standard BDD operations like ITE (if-then-else) to implement the union $B_1 \cup B_2$, the intersection $B_1 \cap B_2$, variable renaming $B[x \to x']$, and existential quantification $\exists x.B$ for $x \in \mathit{Var}$, $x' \in \mathit{Var}'$. For MTBDDs additionally APPLY and ABSTRACT are used to perform numerical operations. For details on these operations we refer to [Sch12].

2.4.2 Symbolic representations of DTMCs

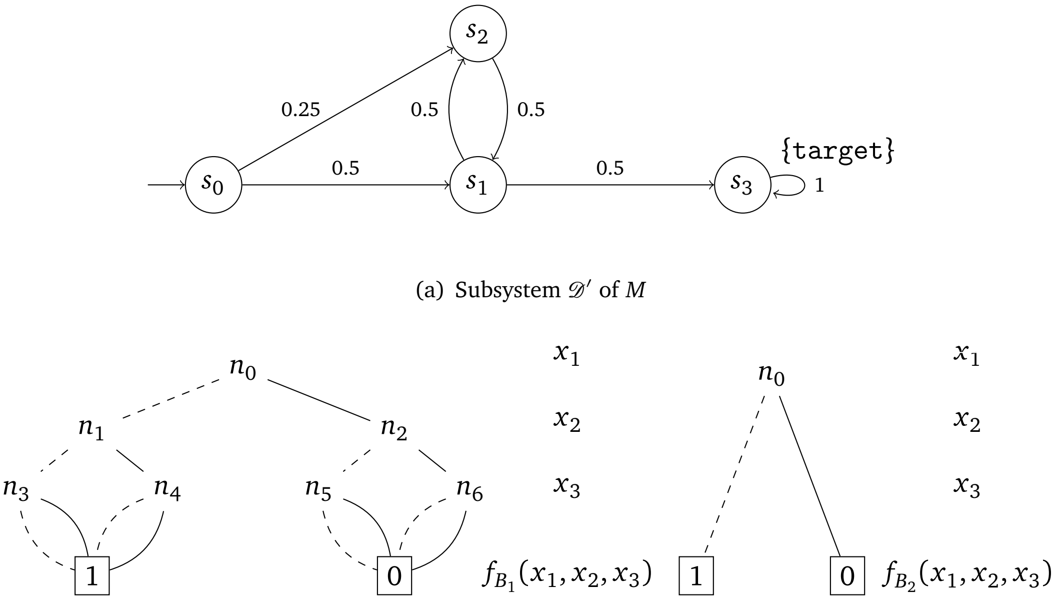
As shortly explained in Remark 3 on Page 22, DTMCs can easily be specified by explicit matrix representations. However, for larger system consisting of millions of states, a *symbolic* representation of the state space as well as the transitions probability matrix are established, e. g., in the probabilistic model checker `PRISM` [KNP11].

BDDs and MTBDDs can be used to represent DTMCs symbolically as follows: Let $\mathcal{D} = (S, s_I, P, L)$ be a DTMC and *Var* a set of Boolean variables such that for each $s \in S$ there is a unique binary encoding $\nu_s \colon \mathit{Var} \to \{0,1\}$ with $\nu_s \neq \nu_{s'}$ for all $s, s' \in S$ with $s \neq s'$. For $s, s' \in S$ we also define $\nu_{s,s'} \colon \mathit{Var} \mathbin{\dot\cup} \mathit{Var}' \to \mathbb{R}$ with $\nu_{s,s'}(x) = \nu_s(x)$ and $\nu_{s,s'}(x') = \nu_{s'}(x)$ for all $x \in \mathit{Var}$, $x' \in \mathit{Var}'$. A target state set $T \subseteq S$ is represented by a BDD $\widehat{T}$ over *Var* such that $\widehat{T}(\nu_s) = 1$ iff $s \in T$. Similarly, we have a BDD $\widehat{I}$ for the initial state such that $\widehat{I}(\nu_s) = 1$ iff $s = s_I$. The probability matrix $P \colon S \times S \to [0,1] \subseteq \mathbb{R}$ is represented by an MTBDD $\widehat{P}$ over $\mathit{Var} \mathbin{\dot\cup} \mathit{Var}'$ such that $\widehat{P}(\nu_{s,s'}) = P(s,s')$ for all $s, s' \in S$. For an MTBDD B over *Var* we use B_{bool} to denote the BDD over *Var* with $B_{\text{bool}}(\nu) = 1$ iff $B(\nu) > 0$ for all valuations ν.

The transition matrices of practically relevant systems are usually sparse and well-structured with relatively few different probabilities; therefore the symbolic MTBDD representation is in many cases more compact by several orders of magnitude than explicit representations.

Remark 12 (Probabilistic model checking on symbolic graph representations) *For symbolically represented DTMCs, we only need model checking of reachability properties. This is done by solving a linear equation system as described in Section 2.3.1 based on the symbolic representation of the transition probability matrix. For details we refer to [BCH$^+$97, Par02].*

Example 6 *Consider the DTMC $\mathcal{D}$ from Figure 2.1. To illustrate the symbolic representation of this DTMC, we ignore the absorbing non-target state s_8. We use a variable set $\mathit{Var} = \{x_1, x_2, x_3\}$ with the ordering $x_1 < x_2 < x_3$. Note that if we included state s_8, we would need an additional variable. To represent transitions, we extend the set of variables by a copy of itself: $\mathit{Var} \mathbin{\dot\cup} \mathit{Var}' = \{x_1, x_2, x_3, x_1', x_2', x_3'\}$. A possible unique encoding of the states as well as the transitions is given by the following assignments:*



(a) Subsystem $\mathscr{D}'$ of M

(b) BDD B_1 representing the state space of DTMC $\mathscr{D}'$

(c) BDD B_2, reduced version of B_1

Figure 2.4: This figure shows equivalent BDD representations of the state space of the subsystem $\mathscr{D}'$ of $\mathscr{D}$, depicted in Figure 2.1.

	x_1	x_2	x_3
s_0	0	0	0
s_1	0	0	1
s_2	0	1	0
s_3	0	1	1
s_4	1	1	1
s_5	1	1	0
s_6	1	0	0
s_7	1	0	1

	x_1	x_2	x_3	x'_1	x'_2	x'_3
$s_0 \to s_1$	0	0	0	0	0	1
$s_0 \to s_2$	0	0	0	0	1	0
$s_0 \to s_5$	0	0	0	1	1	0
$s_1 \to s_2$	0	0	1	0	1	0
$s_1 \to s_3$	0	0	1	0	1	1
$s_2 \to s_1$	0	1	0	0	0	1
$s_2 \to s_4$	0	1	0	1	1	1
$s_3 \to s_3$	0	1	1	0	1	1
$s_4 \to s_1$	1	1	1	0	0	1
$s_4 \to s_3$	1	1	1	0	1	1
$s_5 \to s_6$	1	1	0	1	0	0
$s_6 \to s_3$	1	0	0	0	1	1
$s_6 \to s_7$	1	0	0	1	0	1
$s_7 \to s_5$	1	0	1	1	1	0
$s_7 \to s_6$	1	0	1	1	0	0

Based on the above encoding, the BDD B_1 in Figure 2.4(b) represents the state space of the subsystem $\mathscr{D}' \sqsubseteq \mathscr{D}$ from Figure 2.4(a). Node n_0 is labeled by the variable x_1, n_1 and n_2 are labeled

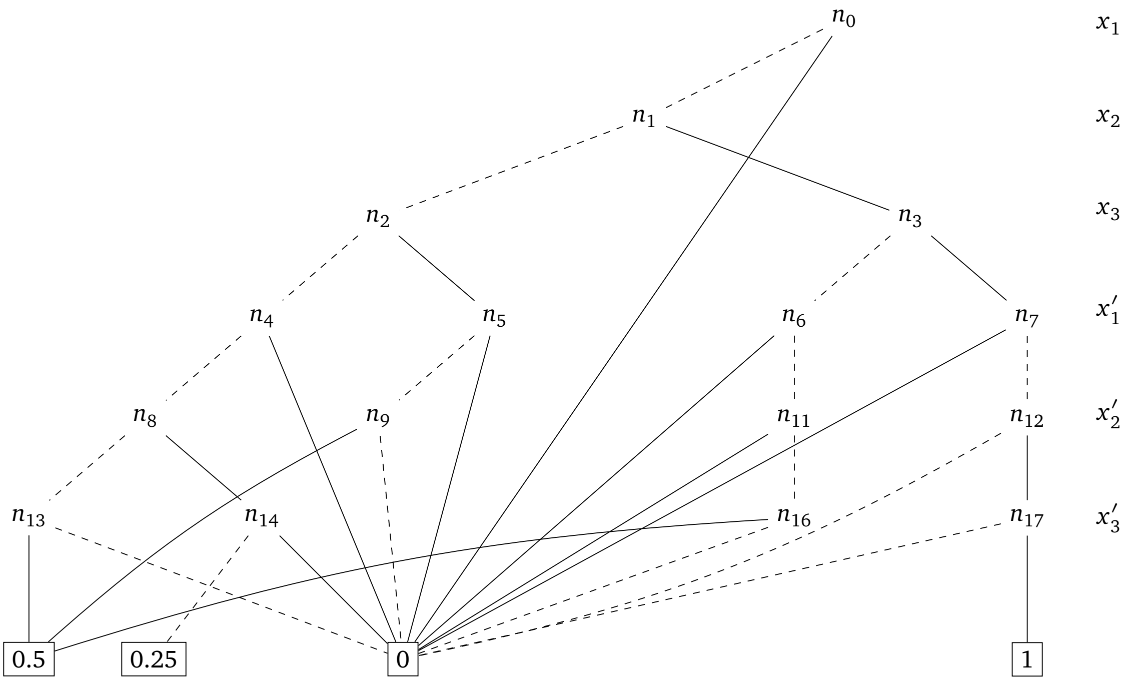


Figure 2.5: MTBDD B_3 representing the transition matrix of the DTMC $\mathscr{D}'$.

by x_2, and n_3, n_4, n_5, n_6 by x_3. Thus each level corresponds to the choice of the value for exactly one variable. The leaves labeled by 0 and 1 indicate whether the function $f_{B_1}(x_1, x_2, x_3)$ evaluates to 0 or 1. Dashed edges indicate $lo(n)$, i. e., that the variable at whose level the edge starts, is set to 0, solid edges depict $hi(n)$, i. e., that it is set to 1.

Consider the path $n_0, n_1, n_3, 1$ which results from choosing the low successor for each inner node. This path is induced by the assignment ν_{s_0} with $\nu_{s_0}(x_1) = \nu_{s_0}(x_2) = \nu_{s_0}(x_3) = 0$ and has the evaluation $f_{B_1}(0,0,0) = 1$. Thus, state s_0 is part of the set encoded by this BDD. Consider furthermore the path $n_0, n_2, n_6, 0$ which results from taking the high successor for all inner nodes, i. e., $f_{B_1}(1,1,1) = 0$. As this corresponds to state s_4 and evaluates to 0, the state s_4 is not included in this set.

The BDD B_2 in Figure 2.4(c) encodes the same state set as B_1 but it is reduced. Since in B_1 the choice of assignment for variable x_1 already determines the evaluation of the whole function, all intermediate nodes after n_0 can be eliminated.

Finally, the transition probability matrix of the DTMC $\mathscr{D}'$ can be encoded by the MTBDD B_3 in Figure 2.5. For each $s, s' \in S$, the path induced by the assignment $\nu_{s,s'}$ leads to a leaf that is labeled by the probability $P(s, s')$ to move from s to s' in $\mathscr{D}'$. For example, the path $n_0, n_1, n_2, n_4, n_8, n_{13}, 0.5$ is induced by the assignment ν_{s_0,s_1}, which corresponds to the transition between the states s_0 and s_1 with probability 0.5. This MTBDD is already reduced. Note that in our implementation we use an interleaved variable ordering for the transition MTBDD, i. e., the levels would be in the order $x_1, x'_1, x_2, x'_2, x_3, x'_3$. We refrained from this ordering as a transition is easier to read with a non-interleaved ordering.

2.5 Probabilistic counterexamples

In this section we give a short introduction to counterexamples for probabilistic reachability properties. The concepts presented here are basically taken from Han, Katoen and Damman in [HKD09] as this work can be considered as the basic foundation for this thesis.

Intuitively, a counterexample for a DTMC $\mathcal{D} = (S, s_I, P, L)$, a set of target states $T \subseteq S$, and an upper probability bound $\lambda \in [0,1] \subseteq \mathbb{Q}$ is a set of paths starting in s_I and ending in a target state out of T such that the combined probability mass of these paths exceeds the bound. For a PA $\mathcal{M}$, a counterexample specifies a deterministic memoryless scheduler σ and a counterexample for the induced DTMC $\mathcal{M}_\sigma$. Formally, we first fix the set of paths that can be part of a counterexample, so-called *evidences* for the violation of a PCTL state formula of the form $\mathbb{P}_{\leq\lambda}(\lozenge T)$.

Definition 35 (Evidence and counterexample) *Given a DTMC $\mathcal{D} = (S, s_I, P, L)$ and a PCTL state formula $\psi = \mathbb{P}_{\leq\lambda}(\lozenge T)$ such that $\mathcal{D} \not\models \psi$. A path $\pi \in Paths^{\mathcal{D}}_{fin}(s_I, T)$, i. e., $\pi \models \lozenge T$, is called an* evidence for the violation of the PCTL path formula $\lozenge T$ in $\mathcal{D}$. *Path π is called a* shortest evidence *iff there exists no other evidence π' which is a real prefix of π.*

A counterexample for $\mathcal{D}$ and ψ *is a set of shortest evidences $C_\psi \subseteq Paths^{\mathcal{D}}_{fin}(s_I, T)$ such that $C_\psi \not\models \psi$.*

An evidence is a finite path contributing to the violation of a property with respect to the probability. A counterexample is a set of finite paths—shortest evidences—whose cumulated probability mass leads to violation of the PCTL property. For the reachability property $\psi = \mathbb{P}_{\leq\lambda}(\lozenge T)$, a counterexample is a set $C_\psi \subseteq Paths^{\mathcal{D}}_{fin}(s_I, T)$ such that $Pr^{\mathcal{D}}_{s_I}(C_\psi) = \sum_{\pi \in C_\psi} \mathbf{P}^{\mathcal{D}}(\pi) > \lambda$. Note that as the shortest evidences are all stochastically independent paths, the probability computation consists just of computing the sum over all paths. Analogously, if we have a strict bound on the probability, i. e., $\psi = \mathbb{P}_{<\lambda}(\lozenge T)$, it suffices to actually reach the probability bound for the counterexample $C_\psi : Pr^{\mathcal{D}}_{s_I}(C_\psi) = \sum_{\pi \in C_\psi} \mathbf{P}^{\mathcal{D}}(\pi) \geq \lambda$

Counterexamples for reachability properties are the main focus of this thesis. The reduction of Until-formulae can be done as follows.

Reduction of until-formulae to reachability for DTMCs Both model checking and counterexample generation for PCTL properties like $\psi = \mathbb{P}_{\leq\lambda}(a\,\mathcal{U}\,b)$ can be reduced to the handling of mere reachability properties. Consider the following simple transformation of a DTMC $\mathcal{D} = (S, s_I, P, L)$ where ψ is violated:

- For all $s \in S$ with $b \in L(s)$ set $P(s,s') = 0$ for all $s' \in S \setminus \{s\}$ and $P(s,s) = 1$.
- For all $s \in S$ with $a, b \notin L(s)$ set $P(s,s') = 0$ for all $s' \in S \setminus \{s\}$ and $P(s,s) = 1$.

Basically, all states that are labeled with b as well as all states that are neither labeled with a or with b are made absorbing. Now, the only non-absorbing states are those labeled with a. Thereby, every path that reaches a b-labelled state satisfies $a\,\mathcal{U}\,b$. By checking the probability of

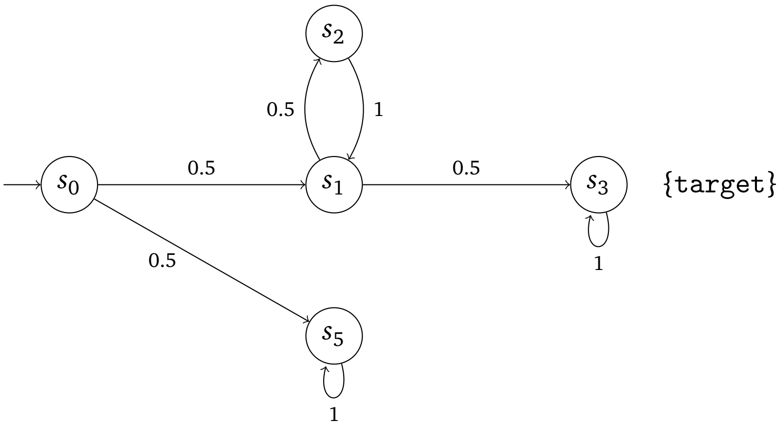


Figure 2.6: DTMC $\mathcal{D}_{inf}$ inducing an infinite set of paths as counterexample for property $\mathbb{P}_{<0.5}(\lozenge \mathrm{target})$

reaching b-states we actually compute the probability of the path formula $a\,\mathcal{U}\,b$ in the original DTMC.

Lower bounds on the probability for DTMCs If reachability properties together with a lower bound on the probability occur, the computation of counterexamples can be reduced to the case of upper bounds by considering sets of paths that don't reach the target states T but end in bottom SCCs without states from T. Formally, we fix $B \subseteq S \setminus T$ as the set of these bottom SCCs. Then we have $\psi = \mathbb{P}_{\geq\lambda}(\lozenge T)$ and a counterexample $C_\psi \subseteq Paths^{\mathcal{D}}_{fin}(s_I, B)$ such that $\mathbf{P}^{\mathcal{D}}(C_\psi) > (1-\lambda)$. The intuition is that if the probability of not satisfying the reachability property is higher than $1-\lambda$ then the lower bound λ on satisfying the reachability property cannot not be reached.

We refer again to [HKD09] for further details on counterexamples for, e. g., step bounded reachability.

ω-regular properties for DTMCs We already discussed that model checking ω-regular properties on DTMCs can be reduced to computing reachability probabilities, see Section 2.3.3.1. The same holds for computing counterexamples of ω-regular properties.

Definition 36 (Counterexample for ω-regular properties) *Let $\mathcal{D} = (S, s_I, P, L)$ be a DTMC, $\mathcal{L}$ be an ω-regular property and $\mathcal{A} = (Q, q_I, \Sigma, \delta, F)$ be a DRA with $\mathcal{L}(\mathcal{A}) = \mathcal{L}$. Let $\mathcal{D} \otimes \mathcal{A} = (S \times Q, (s,q)_I, Act, P', L')$ be the product of $\mathcal{D}$ and $\mathcal{A}$. Let $\mathcal{B} = \{B_1, \ldots, B_n\} \subseteq 2^{S\times Q}$ be the accepting BSCCs of $\mathcal{D} \otimes \mathcal{A}$ and $\lambda \in [0,1] \subseteq \mathbb{Q}$ be an upper probability bound. Let $T_{\mathcal{B}} = \{s \in S \times Q \mid \exists B \in \mathcal{B} . s \in B\}$.*

A counterexample for $\mathcal{D}$ and $\mathcal{L}$ *is defined as a counterexample for $\mathcal{D}$ and the PCTL state formula $\mathbb{P}_{\leq\lambda}(\lozenge T_{\mathcal{B}})$.*

A counterexample for an ω-regular property is thereby simply a counterexample for a reachability property where the states of the accepting BSCCs from $\mathcal{B}$ serve as target states.

To give a short intuition on the problems of path-based counterexamples, we discuss a short example. The number of paths that are needed to form a counterexample might be very large. In case of strict probability bounds it can even be infinite [HKD09]. This is illustrated by the following example.

Example 7 (Infinite number of paths) *Consider the DTMC $\mathcal{D}_{inf}$ depicted in Figure 2.6, which is a slightly modified version of the DTMC $\mathcal{D}$ of Figure 2.1 on Page 20, and the set of target states $T = \{s_3\}$. We are interested in the reachability property $\mathbb{P}_{<0.5}\lozenge(T)$ with a strict probability bound of 0.5. The probability of reaching the only target state s_3 is 0.5 here, so the property is violated. In order for a set of paths to form a counterexample it would need to have the exact reachability probability 0.5. The set of all paths leading to s_3 from the initial state is given by:*

$$C = \{\pi_i \in Paths_{fin}^{\mathcal{D}_{inf}} \mid \pi = s_0 s_1 (s_2 s_1)^i s_3, i \in \mathbb{N}\} = Paths_{fin}^{\mathcal{D}_{inf}}(s_0, s_3)$$

The probability of this set of paths can be computed as follows:

$$\begin{aligned} Pr_{s_I}^{\mathcal{D}_{inf}}(C) &= \sum_{i\in\mathbb{N}} 0.5 \cdot (0.5 \cdot 1)^i \cdot 0.5 \\ &= 0.5 \cdot \sum_{i\in\mathbb{N}} (0.5)^i \cdot 0.5 \\ &= 0.5 \cdot \frac{1}{1-0.5} \cdot 0.5 \quad \textit{(using the geometric series)} \\ &= 0.5 \end{aligned}$$

So, this infinite set of paths has exactly the probability $Pr_{s_I}^{\mathcal{D}_{inf}}(C) = 0.5$ and forms the only possible counterexample.

Although this example might seem unrealistic, there are actually many examples and practical case studies where the number of needed paths is exponentially larger then the number of states of the DTMC under consideration [HKD09].

What remains is to give a formal definition of counterexamples for PAs. As mentioned in the beginning, the crucial point here is to compute a scheduler that induces a DTMC where the actual counterexample is formed. Note that it might not be beneficial to just compute the maximizing scheduler as explained in Section 2.3.1.2.

Definition 37 (Counterexample for PAs) *Given a PA $\mathcal{M} = (S, I, Act, \mathcal{P}, L)$ and a PCTL formula ψ such that $\mathcal{M} \not\models \psi$, a* counterexample *for $\mathcal{M}$ and ψ is a pair (σ, C_ψ) such that σ is a scheduler for $\mathcal{M}$ and C_ψ is a counterexample for ψ and the DTMC $\mathcal{M}_\sigma$.*

2.6 Solving technologies

We introduce three different concepts of solvers which are all used later on. We use classical SAT solvers, their extension by “theories”, SAT modulo theories (SMT), and mixed integer linear programming (MILP).

2.6.1 SAT solving

We present a short summary of SAT solving. We first need the syntax of *quantifier-free Boolean formulae*.

Definition 38 (Boolean formula) *Assume a set $Var = \{x_1, \dots, x_n\}$ of Boolean variables. A* quantifier-free Boolean formula *is given by the following grammar:*

$$\varphi ::= x_i \mid \neg\varphi \mid (\varphi \wedge \varphi)$$

with $x_i \in Var$.

The set of all Boolean formulae over *Var* is denoted by *Bool*(*Var*). We use syntactic sugar like $\vee, \rightarrow, \leftrightarrow$. As for evaluations for BDDs, see Section 2.4.1, let $\mathscr{V}(Var) = \{\nu : Var \rightarrow \{0,1\}\}$ be the set of all variable valuations. An evaluation—also referred to as *assignment*—$\nu \in \mathscr{V}(Var)$ for *Var* assigns to every variable from *Var* a Boolean value. Using the Boolean connectives, this can be used to define a valuation for Boolean formulae. We overload the evaluation function by $\nu : Bool(Var) \rightarrow \{0,1\}$. A formula $\varphi \in Bool(Var)$ is called *satisfiable* if and only if there is an evaluation such that $\nu(\varphi) = 1$. The problem of checking the satisfiability of Boolean formulae is called the *satisfiability problem*. Tools deciding whether a Boolean formula is satisfiable are called *SAT solvers*. Though the satisfiability problem is known to be NP-hard [Coo71], in the past there have been great advances developing SAT solvers such that practical examples with many thousands of variables can be handled. Basic work was done by Davis, Putnam, Logemann and Loveland by developing a method called DPLL algorithm [DP60, DLL62].

For instance, a famous open-source SAT solver is `MiniSAT` [ES03]; a modern extension is `Glucoser` [AS09]. For further introduction to SAT solving we refer to [BHvMW09].

2.6.2 SMT solving

SMT refers to SAT-modulo-theories [dMB11], which is a generalization of the classical satisfiability problem (SAT). An SMT formula allows for atoms of a given theory as atomic proposition; here we use linear real arithmetic as theory.

Definition 39 (SMT formula) *Assume a set $Var = \{x_1, \dots, x_n\}$ of real-valued variables. A* quantifier-free linear real-arithmetic formula *φ is given by the following grammar:*

$$t ::= a \mid a \cdot x_i$$

$$
\begin{aligned}
p &::= t \mid p + p \\
c &::= p = p \mid p < p \\
\varphi &::= c \mid \neg\varphi \mid \varphi \wedge \varphi
\end{aligned}
$$

where $a \in \mathbb{Z}$.

SMT problems can be solved by the combination of a DPLL-procedure (as used for deciding SAT problems) and a theory solver that is able to decide the satisfiability of conjunctions of theory atoms. For a description of such a combined algorithm see, e. g., [DdM06]. Popular SMT solvers are, e. g., `Z3` [dMB08] or `MathSAT` [BCF$^+$08].

2.6.3 MILP solving

A *mixed integer linear program* optimizes a linear objective function under a condition specified by a conjunction of linear inequalities. A subset of the variables in the inequalities is restricted to take only integer values, which makes solving MILPs NP-hard [GJ79, Problem MP1].

Definition 40 (Mixed integer linear program) *Let $A \in \mathbb{Q}^{m\times n}$, $B \in \mathbb{Q}^{m\times k}$, $b \in \mathbb{Q}^m$, $c \in \mathbb{Q}^n$, and $d \in \mathbb{Q}^k$. A* mixed integer linear program *(MILP) consists in computing* $\min c^T x + d^T y$ *such that $Ax + By \leq b$ and $x \in \mathbb{R}^n$, $y \in \mathbb{Z}^k$.*

MILPs are typically solved by a combination of a branch-and-bound algorithm with the generation of so-called cutting planes. These algorithms heavily rely on the fact that relaxations of MILPs which result by removing the integrality constraints can be solved efficiently. MILPs are widely used in operations research, hardware-software co-design, and numerous other applications. Efficient open source as well as commercial implementations are available like `Gurobi` [Gur13], `Scip` [Ach09] or `Cplex` [cpl12] by IBM. We refer to, e. g., [Sch86] for more information on solving MILPs.

CHAPTER 3

Related work

In this chapter we discuss other works on counterexample generation for DTMCs and MDPs or PAs, respectively. We explain the intuition of all available approaches that—to the best of our knowledge—have been made so far. Differences and relations to our own work are discussed at the end of each corresponding chapter. Roughly, we have two categories: *path-based counterexamples* and *subsystem-based counterexamples*. For a detailed overview with numerous examples we refer to [21].

3.1 Path-based counterexamples

Counterexamples based on paths of a system are the classic notion as in Definition 35 on Page 45. We summarize all approaches that represent counterexamples in this way.

3.1.1 Minimal and smallest counterexamples

In [HK07a, HKD09], Han and Katoen shaped the notions of *minimal* and *smallest counterexamples*. A minimal counterexample for a reachability property and a DTMC contains the minimal number of evidences that is needed to form a counterexample. In general, a minimal counterexample is not unique. To require a stronger condition, a smallest counterexample is a minimal counterexample with maximal probability. Again, this stronger condition does not necessarily induce a unique counterexample.

As a way to compute these counterexamples, the authors of [HK07a, HKD09] use a k-shortest path algorithm for weighted directed graphs. They show that the k most probable paths in a DTMC (forming a smallest counterexample) correspond to the k shortest paths in a related weighted digraph. This digraph is constructed from the DTMC by considering the negative logarithms of the transition-probabilities as weights of the edges. In order to compute these k shortest paths, the authors choose the *recursive enumeration algorithm* (REA) [JM99] by Jiménez and Marzal, where the number k of paths is not determined beforehand but on-the-fly by an external condition. In this case that means that the search terminates once a counterexample has been found, i. e., that the paths have enough cumulated probability mass. This avoids fixing some (arbitrary) k in advance, and allows for finding the smallest k yielding a counterexample. Besides the mentioned publications, for more detailed information we refer to the dissertation of Tingting Han [Han09].

As an alternative, Aljazzar and Leue proposed a K^*-algorithm [AL11] for finding the k shortest paths. In contrast to other algorithms like the aforementioned REA or Eppstein's algorithm [Epp98], the state space of the graph at hand does not have to be generated to its full extend. Starting from an initial state, the graph is expanded *on-the-fly* which is often beneficial for large state spaces.

Another possibility to handle large state spaces is to use symbolic representations of DTMCs, see Section 2.4.2. A first approach was made by Günther, Schuster and Siegle [GSS10], who proposed a BDD-based algorithm for computing the k most probable paths of a DTMC. They use an adaption of Dijkstra's shortest path algorithm [Dij59], called *flooding Dijkstra*, to determine the most probable path. In order to get the k-shortest paths, they transform the DTMC in each step such that the most probable path of the transformed system corresponds to the second-most-probable path in the original DTMC. This involves copying the DTMC and redirecting transitions from the original system to the copy. In the end, this yields a symbolic representation of a *minimal* counterexample. Details are explained in Section 5.7 as this is also relevant to our approaches. Note that the underlying BDD grows exponentially when applying these transformations for all needed paths.

3.1.2 Heuristic approaches

Instead of striving to compute minimal or even smallest counterexamples, it might be reasonable to use heuristics and compute counterexamples more efficiently or for larger systems.

The only heuristic approach generating path-based counterexamples we are aware of is an adaption of *bounded model checking* (*BMC*) [BCC$^+$03] by Wimmer *et al.* in [WBB09, 26, 25]. A SAT solver is used to generate evidences until enough probability mass is accumulated. The basic idea of BMC is to formulate the existence of an evidence of a certain length as a satisfiability problem. In [WBB09], purely propositional formulas are used which does not allow to take the actual probability of an evidence into account; in [26, 25] this was extended to SMT formulas over linear real arithmetic, which allows to enforce a lower bound on the probability of evidences.

In both cases, the starting point is a symbolic representation of the DTMC, see Section 2.4.2. From the BDDs and MTBDDs, a SAT formula is generated where a satisfying variable assignment corresponds to a path of the DTMC starting in an initial state and ending at a target state. Starting with a small path length n, all paths of this length are enumerated. If afterwards the set of paths does not form a counterexample, n is incremented and again paths are collected. This approach is explained in more detail in Section 5.6.

As an optimization, loops occurring on the found paths can be identified. The probability of further paths containing states that occur on an already identified loop is then computed considering arbitrary unrollings of that loop. Every found loop and path is excluded from the further iterations.

3.1.3 Compact representations of counterexamples

As the number of paths needed to form a counterexample can be very high, several approaches were made to obtain compact representations of a path-based counterexample. All of them exploit the fact that many paths in a counterexample differ only in the number of loop iterations.

First, Han, Katoen and Damman [HKD09, DHK08] proposed the representation of counterexamples as *regular expressions*: Formally, a DTMC is transformed into a deterministic finite automaton (DFA) whose alphabet consists of pairs of states and probabilities and whose graph structure is equal to the one of the DTMC. The transitions of the automaton are then labeled by pairs of states and probabilities such that a path of the DTMC corresponds to a run of the automaton. The run corresponding to the most probable path of the DTMC is considered. Applying the state elimination algorithm to obtain a regular expression from an automaton to this path yields a compact representation of all paths that use loops occurring on this path. By defining a measure for regular expressions, their probability mass is computed. If this is not enough, another iteration with the next most probable path is started.

A different compaction of counterexamples based on the strongly connected components (SCCs) of a DTMC is described by Andrés, D'Argenio and van Rossum in [ADvR08]. First, all SCCs are determined. Then, direct edges leading from the input states of the SCCs to their output states replace the inner states. These edges carry the whole probability mass of all paths walking through the SCCs and leaving them. This is computed via model checking for DTMCs. Counterexamples are then computed on this abstract, acyclic DTMC.

3.2 Subsystem-based counterexamples

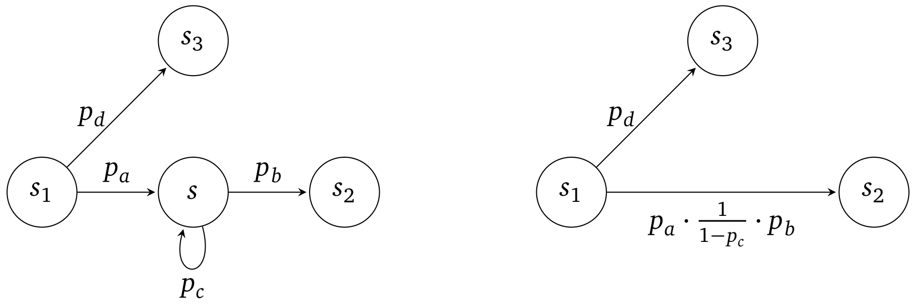
Apart from what we are going to present in this thesis, the only other approach generating critical subsystem as representations for counterexamples is [AL06, AL10] by Aljazzar and Leue. They exploit a *best-first* search for various search algorithms. The main advantage is that this can be pursued in an on-the-fly manner, avoiding an a priori generation of the state space. Using the simulation engine of `PRISM` [KNP11], a successor relation on states delivers for one state exactly the explicit representation of its successors. Starting from an initial state of the system, the system is thereby successively extended along most probable local paths. Additionally, a heuristic function enables the user to use specific knowledge about benchmarks to prefer or penalize certain states of the system.

As for many cases it is beneficial to represent a counterexample by a subsystem instead of a set of paths, an *extended best-first search (XBF)* is used, where for each node not only the predecessor inducing the optimal path is stored but also the other connections. Thereby, a whole system instead of a single path is obtained. The approaches are implemented in the tool `DiPro` [ALFLS11], which was the first publicly available tool supporting counterexample generation for probabilistic systems.

This approach was transferred to MDPs (and therefore also PAs) in [AL09] presenting the first approach to counterexamples for MDPs. The idea is to compute a *maximizing* scheduler and compute a counterexample on the induced DTMC, see Section 2.2.2. The drawback is that again the whole state space has to be generated and model checking has to be applied beforehand. Therefore, another method was proposed which allows to not only compute the paths but also the scheduler on the fly. The problem when applying on-the-fly algorithms like K^* to a PA is that the generated paths might not be compatible to the same scheduler. Therefore all paths are kept and clustered according to the scheduler choice made in each state using dedicated data structures. Using this, a (not necessarily memoryless) scheduler is determined which is compatible to the set of paths that was computed.

3.3 Parametric systems

The only methods that were developed for the verification of parametric DTMCs are—to the best of our knowledge—presented in [Daw04, HHZ10]. In [Daw04], it was proposed both for DTMCs and PDTMCs to utilize the computation of regular expressions for deterministic finite automata by *state elimination*. A state of the (P)DTMC is chosen for elimination. All transitions having this state as source or destination state are then replaced by transitions that “bypass” the former state. This is illustrated by the toy example below, where state s is to be eliminated.



The shortest path from s_1 to s_2 has probability $p_a \cdot p_b$, which is therefore the first summand of the probability of going from s_1 to s_2. The probability of the self-loop on s has to be taken into account for all possible paths leading from s_1 to s_2 over s. We get an infinite sum of paths by the unrolling of this loop. This leads to the geometric series and to the probability

$p_a \cdot \frac{1}{1-p_c} \cdot p_b$. The parametric probability is represented by a *rational function*, see Definition 2 on Page 19. An instantiation of the parameters yields a concrete probability. For instance, let $p_a = 0.5, p_d = 0.5, p_c = 0.9$ and $p_b = 0.1$. For this "concrete" DTMC we have the probability $0.5 \cdot \frac{1}{1-0.9} \cdot 0.1 = 0.5$ of going from s_1 to s_2.

This technique was optimized and implemented in [HHZ09]. The first publicly available tool for the verification of PDTMCs was PARAM [HHWZ10]. A conclusive work with several extensions is available in [HHZ10].

CHAPTER 4

SCC-based model checking

Summary In this first main chapter we present a general scheme to abstract DTMCs with respect to their strongly connected components. As we saw before, the size of counterexamples strongly relies on the loop structure of the system: A large number of paths might have to be collected that are similar except for the number of iterations of the same loops. Following a partial order on states given by the loop structure of the SCCs, we determine *reachability probabilities* by a bottom-up computation. This forms a *model checking algorithm*, published in [11]. Subsequently, we describe how these approaches can be extended to *parametric Markov chains* (PDTMCs). This was published in [2] together with a novel factorization technique for polynomials which is out of the scope of this thesis.

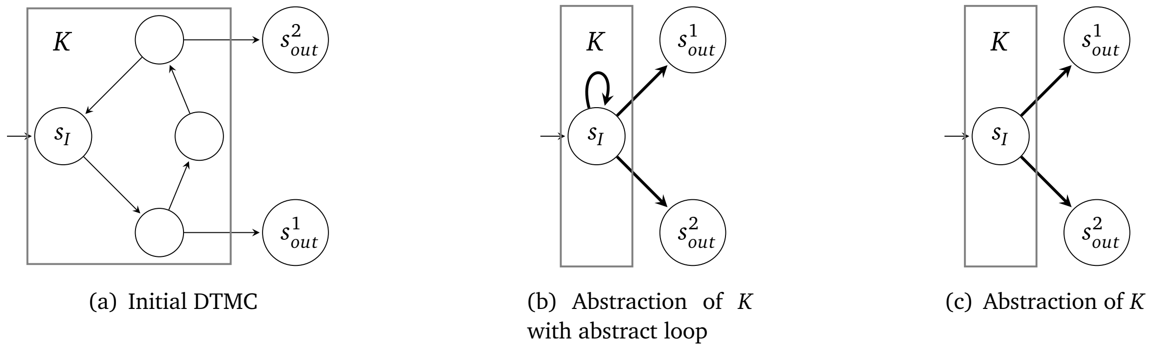
Background This chapter does not need extensive background. It suffices to be familiar with some properties of DTMCs, in particular the model checking of reachability properties, see Sections 2.2.1 and 2.3.1.1. we make use of Tarjan's Algorithm [Tar72] to compute the maximal strongly connected components (SCCs) of a directed graph, see Definition 13. For the foundations of PDTMCs we refer to Section 2.2.3.

4.1 SCC-based abstraction

Let us first lie the theoretical background we need for a definition of our abstraction scheme. In the following, we assume a DTMC $\mathcal{D} = (S, I, P, L)$ with only stochastic distributions and a set of absorbing target states $T \subseteq S$. Our goal is to compute for each initial state $s_I \in \mathit{Init}_{\mathcal{D}}$ and each target state $t \in T$ the probability of reaching t from s_I.

Remark 13 (Stochastic distributions) *In this chapter we assume that both the initial distribution is stochastic and the transition probabilities sum up to* 1 *for each state, i. e., for all* $s \in S$ *it holds that* $\sum_{s' \in S} P(s,s') = 1$*, thereby also inducing stochastic distributions. This is necessary as we make use of the fact that the probability of reaching a bottom SCC is always* 1*, see Section 2.2.1.*

The basic concept of our model checking approach is to replace a non-absorbing subset of states $K \subseteq S$ and their transitions inside a DTMC $\mathscr{D}$ by transitions directly leading from the input states $Inp(K)$ of K to the output states $Out(K)$ of K. These transitions carry the total probabilities of all paths visiting only states in K. This concept is illustrated in Figure 4.1: In Figure 4.1(a), an arbitrary, non-absorbing set of states K has one input state s_I and two output states s^1_{out}, s^2_{out}. The abstraction in Figure 4.1(c) hides every state of K except for s_I; all transitions are directly leading to the output states. Figure 4.1(b) can be ignored for the moment.



(a) Initial DTMC

(b) Abstraction of K with abstract loop

(c) Abstraction of K

Figure 4.1: Concept of DTMC abstraction

As we need a probability measure for arbitrary subsets of states, we first define sub-DTMCs that are induced by such sets of states.

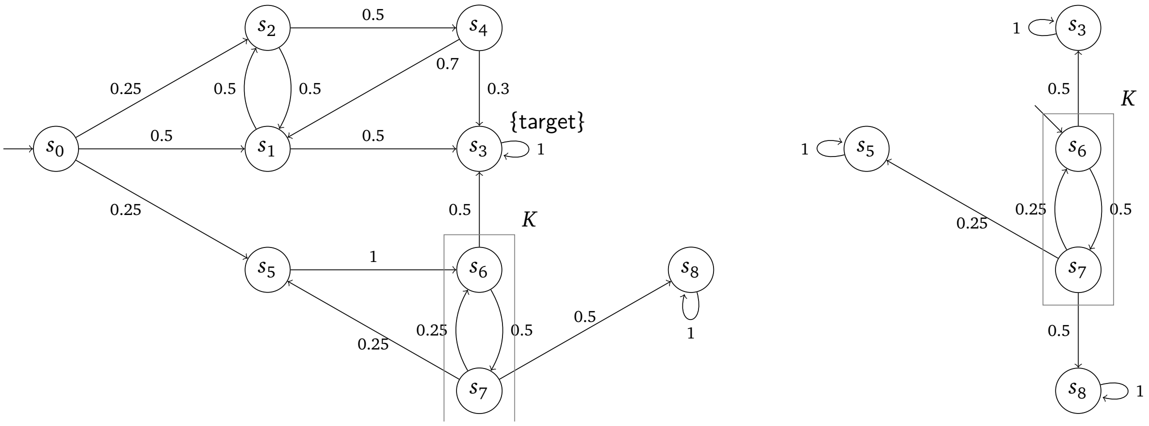
Definition 41 (Induced DTMC) *Given a DTMC* $\mathscr{D} = (S, I, P, L)$ *and a non-absorbing subset* $K \subseteq S$ *of states, the* induced DTMC *over* K and $\mathscr{D}$ *is given by* $\mathscr{D}^K = (S^K, I^K, P^K, L^K)$ *with:*

- $S^K = K \cup Out(K)$
- $\forall s \in S^K.\, I^K(s) \neq 0 \iff s \in Inp(K)$
- $P^K(s,s') = \begin{cases} P(s,s') & \text{if } s \in K \wedge s' \in S^K \\ 1 & \text{if } s = s' \in Out(K) \\ 0 & \text{otherwise} \end{cases}$
- $\forall s \in S^K.\, L^K(s) = L(s)$.

Intuitively, for inner states of K all incoming and outgoing transitions are preserved while the output states are made absorbing.

Remark 14 (Initial distribution for induced DTMCs) *We allow an arbitrary input distribution I with the only constraint that $I(s) \neq 0$ iff s is an input state of K, i. e., these states form the support of I.*

Example 8 *Consider the DTMC $\mathcal{D}$ of Figure 2.1 on Page 20, again depicted in Figure 4.2(a) and a subset of states $K = \{s_6, s_7\}$. The induced DTMC $\mathcal{D}^K = (S^K, I^K, P^K, L^K)$ over K and $\mathcal{D}$ is shown in Figure 4.2(b) with output states $Out(K) = \{s_3, s_5, s_8\}$ and input state $Inp(K) = \{s_6\}$.*



(a) DTMC $\mathcal{D}$ and set $K = \{s_6, s_7\}$

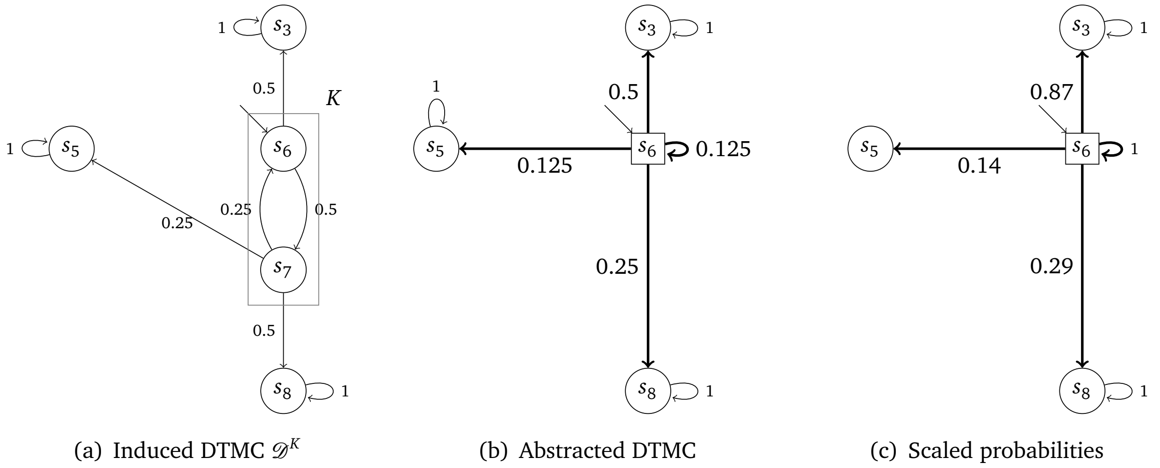
(b) Induced DTMC $\mathcal{D}^K$

Figure 4.2: Induced DTMC $\mathcal{D}^K$ for $K = \{s_6, s_7\}$

For our abstraction we take into account all finite paths that do not intermediately return to the initial state. Consider now Figure 4.1(b), were abstract transitions lead to the output states together with a self-loop on the initial state. The outgoing transitions abstract' all paths that do not visit the input state again, while the self-loop describes all paths that return to the input state. These paths build the set of all paths that add to the probability of finally reaching one of the output states. Note that inside a non-absorbing set of states, the probability of reaching the set of all output states is 1. This is due to the fact that the probability to finally reach a BSCC, i. e., an absorbing set of states, is 1, see [BK08, Theorem 10.27].

Figure 4.1(c) shows the final abstraction where the probability of the self-loop is taken into account for determining the transition probabilities of the outgoing transitions.

Formally, we now define the probability of all finite paths that start in a state s and finally reach a state s' without returning to s beforehand. Note that this includes the case where $s = s'$.



(a) Induced DTMC $\mathscr{D}^K$ (b) Abstracted DTMC (c) Scaled probabilities

Figure 4.3: DTMC Abstraction

Definition 42 *Given a DTMC $\mathscr{D} = (S, I, P, L)$, a non-absorbing state $s \in S$, and a state $s' \in S$, the* path abstraction *of s and s' is given by:*

$$p^{\mathscr{D}}_{abs}(s, s') = \mathbf{P}^{\mathscr{D}}\big(\{\pi = s_0, \ldots, s_n \in Paths^{\mathscr{D}}_{fin}(s, s') \,|\, s_i \neq s \wedge s_i \neq s', 0 < i < n\}\big).$$

Using this we are now ready to define the abstraction of a DTMC $\mathscr{D}$ with respect to initial states and target states. The probabilities are the total reachability probabilities between these states. Let us first consider an example.

Example 9 *Consider again the DTMC $\mathscr{D}^K = (S^K, I^K, P^K, L^K)$ in Figure 4.2(b) which is again depicted in Figure 4.3(a) and let the set of target states $T^K = \{s_3, s_5, s_8\}$ correspond to the absorbing states of $\mathscr{D}^K$. The abstract DTMC $\mathscr{D}^K_{abs} = (S^K_{abs}, I^K_{abs}, P^K_{abs}, L^K_{abs})$ has the state space $S^K_{abs} = \{s_3, s_5, s_6, s_8\}$ and edges from s_6 to all other states. The first abstraction step according to the path abstraction $p^{\mathscr{D}^K}_{abs}$ as in Definition 42 is depicted in Figure 4.3(b). The abstract edges, drawn boldface, include a self-loop on the input state s_6 with probability $p^{\mathscr{D}^K}_{abs}(s_6, s_6)$.*

The probabilities of all finite paths that either leave K without visiting state s_6 again or starting and ending in s_6 are:

$$\begin{aligned} p_{s_6,s_3} &= p^{\mathscr{D}^K}_{abs}(s_6, s_3) = 0.5 & \qquad p_{s_6,s_5} &= p^{\mathscr{D}^K}_{abs}(s_6, s_5) = 0.125 \\ p_{s_6,s_6} &= p^{\mathscr{D}^K}_{abs}(s_6, s_6) = 0.125 & \qquad p_{s_6,s_8} &= p^{\mathscr{D}^K}_{abs}(s_6, s_8) = 0.25 \end{aligned}$$

Note that p_{s_6,s_6} is given implicitly by $1 - p_{s_6,s_6} = p_{s_6,s_3} + p_{s_6,s_5} + p_{s_6,s_8}$. The total probability of reaching the output states is given by paths which first use the loop on s_6 arbitrarily many times (including zero times) and then take a transition to an output state. For example, using the geometric *series, the probability of all paths leading from s_6 to s_3 is given by*

$$\sum_{i=0}^{\infty}(p_{s_6,s_6})^i \cdot p_{s_6,s_3} \;=\; \frac{1}{1-p_{s_6,s_6}} \cdot p_{s_6,s_3} \approx 0.57$$

As the probability of finally reaching the set of absorbing states in $\mathscr{D}^K$ is 1, we can directly scale the probabilities of the outgoing edges such that their sum is equal to 1. This is achieved by dividing each outgoing probability by the sum of all outgoing probabilities, $p_{out} = 0.5 + 0.125 + 0.25 = 0.875$. The abstract and scaled DTMC is depicted in Figure 4.3(c) with the probabilities given by

$$\begin{aligned} \hat{p}_{s_6,s_3} &= 0.5\ /p_{out} \approx 0.57 & \qquad \hat{p}_{s_6,s_5} &= 0.125\ /p_{out} \approx 0.14 \\ \hat{p}_{s_6,s_8} &= 0.25\ /p_{out} \approx 0.29 \end{aligned}$$

We now define the final abstraction formally.

Definition 43 (Abstract DTMC) *For a DTMC $\mathscr{D} = (S, I, P, L)$ with all BSCCs being single absorbing states from $T \subseteq S$, the* abstract DTMC *$\mathscr{D}_{abs} = (S_{abs}, I_{abs}, P_{abs}, L_{abs})$ is given by:*

- $S_{abs} = \{s \in S \mid I(s) \neq 0 \vee s \in T\}$
- $\forall s \in S_{abs}.\, I_{abs}(s) = I(s)$
- $P_{abs}(s,s') = \begin{cases} \dfrac{p^{\mathscr{D}}_{abs}(s,s')}{\sum_{s'' \in T} p^{\mathscr{D}}_{abs}(s,s'')} & \text{if } I(s) > 0 \wedge s' \in T \\ 1 & \text{if } s = s' \in T \\ 0 & \text{otherwise.} \end{cases}$

To summarize, we are able to replace the transitions of a PDTMC by abstract transitions carrying the correct probabilities of reaching target states. The correctness of this abstraction is formulated in the following theorem.

Remark 15 (Abstraction/concretization) *We often call $\mathscr{D}^K$ the* concretization *and $\mathscr{D}^K_{abs}$ the* abstraction.

Theorem 3 *For a DTMC $\mathscr{D} = (S, I, P, L)$ and its abstraction $\mathscr{D}_{abs} = (S_{abs}, I_{abs}, P_{abs}, L_{abs})$ according to Definition 43 it holds for all initial states $s_I \in Init_{\mathscr{D}}$ and all absorbing states $t \in T$ that*

$$\mathbf{P}^{\mathscr{D}}(Paths^{\mathscr{D}}_{fin}(s_I, t)) = \mathbf{P}^{\mathscr{D}_{abs}}(Paths^{\mathscr{D}_{abs}}_{fin}(s_I, t)).$$

Proof 1 First note that for this setting, the bottom SCCs of the DTMC $\mathscr{D}$ are exactly the absorbing states in T. Thus, the probability of reaching a state from T is 1, see again [BK08, Theorem 10.27]. The probability $p^{\mathscr{D}}_{abs}(s_I, s_I)$ can therefore be expressed with respect to the probabilities of reaching an absorbing state without revisiting s_I:

$$p^{\mathscr{D}}_{abs}(s_I, s_I) = 1 - \sum_{t \in T} p^{\mathscr{D}}_{abs}(s_I, t). \tag{4.1}$$

To reduce notation, we define the set of paths R_{loop} looping on s_I and the set of paths R_{out} going to some $t \in T$ without revisiting s_I.

$$R_{loop} = \{s_I, s_0, \dots, s_n, s_I \in Paths^{\mathscr{D}}_{fin} \mid s_i \notin \{s_I\} \cup T, 0 \leq i \leq n\} \tag{4.2}$$

$$R_{out} = \{s_I, s_0, \dots, s_n, t \in Paths^{\mathscr{D}}_{fin} \mid s_i \notin \{s_I\} \cup T, 0 \leq i \leq n, t \in T\} \tag{4.3}$$

As the self-loop in s_I represents the paths of R_{loop}, it holds that

$$p^{\mathscr{D}}_{abs}(s_I, s_I) = \mathbf{P}(R_{loop}). \tag{4.4}$$

We now have for all $s_I \in Init_{\mathscr{D}}$ and $t \in T$:

$$\begin{aligned}
& \mathbf{P}^{\mathscr{D}}(Paths^{\mathscr{D}}_{fin}(s_I, t)) \\
= \;& \mathbf{P}^{\mathscr{D}}(\bigcup_{i=0}^{\infty}\{\pi_1 \cdot \dots \cdot \pi_i \cdot \pi_{out} \mid \pi_j \in R_{loop}, 1 \leq j \leq i;\ \pi_{out} \in R_{out}\}) \\
= \;& \sum_{i=0}^{\infty} \mathbf{P}^{\mathscr{D}}(\{\pi_1 \cdot \dots \cdot \pi_i \cdot \pi_{out} \mid \pi_j \in R_{loop}, 1 \leq j \leq i;\ \pi_{out} \in R_{out}\}) \\
= \;& \sum_{i=0}^{\infty} (\mathbf{P}^{\mathscr{D}}(R_{loop}))^i \cdot \mathbf{P}^{\mathscr{D}}(R_{out}) \\
= \;& \sum_{i=0}^{\infty} (p^{\mathscr{D}}_{abs}(s_I, s_I))^i \cdot \mathbf{P}^{\mathscr{D}}(R_{out}) \text{ (Equation (4.4))} \\
= \;& \frac{1}{1 - p^{\mathscr{D}}_{abs}(s_I, s_I)} \cdot \mathbf{P}^{\mathscr{D}}(R_{out}) \text{ (Geometric Series)} \\
= \;& \frac{1}{\sum\limits_{s_{out} \in T} p^{\mathscr{D}}_{abs}(s_I, s_{out})} \cdot \mathbf{P}^{\mathscr{D}}(R_{out}) \text{ (Equation (4.1))} \\
= \;& \frac{1}{\sum\limits_{s_{out} \in T} p^{\mathscr{D}}_{abs}(s_I, s_{out})} \cdot p^{\mathscr{D}}_{abs}(s_I, t) \text{ (Definition 42)} \\
= \;& P_{abs}(s_I, t) \text{ (Definition 43)} \\
= \;& \mathbf{P}^{\mathscr{D}_{abs}}(Paths^{\mathscr{D}_{abs}}_{fin}(s_I, t))
\end{aligned}$$

Thus the probabilities of reaching the absorbing states from initial states coincide in $\mathscr{D}$ and $\mathscr{D}_{abs}$. □

It remains to define the substitution of subsets of states by their abstractions. Intuitively, a subset of states is replaced by the abstraction as in Definition 43, while the incoming transitions of the initial states of the abstraction as well as the outgoing transitions of the absorbing states of the abstraction are not changed.

Definition 44 (Substitution) *Assume a DTMC $\mathscr{D} = (S, I, P, L)$, a non-absorbing set of states $K \subseteq S$, the induced DTMC $\mathscr{D}^K = (S^K, I^K, P^K, L^K)$, and the abstraction $\mathscr{D}^K_{abs} = (S^K_{abs}, I^K_{abs}, P^K_{abs}, L^K_{abs})$ of $\mathscr{D}^K$. The* substitution of $\mathscr{D}^K$ by its abstraction $\mathscr{D}^K_{abs}$ in $\mathscr{D}$ *is given by $\mathscr{D}_{K \mapsto abs} = (S_{K \mapsto abs}, I_{K \mapsto abs}, P_{K \mapsto abs}, L_{K \mapsto abs})$ with:*

- $S_{K \mapsto abs} = (S \setminus K) \cup S^K_{abs}$
- $\forall s \in S_{K \mapsto abs}.\, I_{K \mapsto abs}(s) = I(s)$
- $P_{K \mapsto abs}(s, s') = \begin{cases} P(s, s') & \text{if } s \notin K \\ P^K_{abs}(s, s') & \text{if } s \in K \wedge s' \in Out(K) \\ 0 & \text{otherwise.} \end{cases}$

Due to Theorem 3, it directly follows that this substitution does not change the satisfaction of reachability properties from input states to the absorbing states of a DTMC.

Corollary 1 *Given a DTMC $\mathscr{D} = (S, I, P, L)$ with absorbing states $T \subseteq S$ and a non-absorbing subset $K \subseteq S$ of states, it holds for all initial states $s_I \in Init_{\mathscr{D}}$ and absorbing states $t \in T$ that*

$$\mathbf{P}^{\mathscr{D}}(Paths^{\mathscr{D}}_{fin}(s_I, t)) = \mathbf{P}^{\mathscr{D}_{K \mapsto abs}}(Paths^{\mathscr{D}_{K \mapsto abs}}_{fin}(s_I, t)).$$

4.2 SCC-based model checking

In the previous section we gave the theoretical background for our model checking algorithm. Now we describe how to compute the abstractions efficiently.

As a heuristic for forming the sets of states that are abstracted, we choose an SCC-based decomposition of the graph: Tarjan's algorithm [Tar72] is called to determine the SCCs of the graph. Afterwards, for each SCC K the input states $Inp(K)$ are ignored. On the resulting decomposed graph, a new search is performed. This yields a new set of SCCs which are strongly connected subcomponents (SCSs) in the context of the original graph. This procedure is iterated until only single states remain. The subset relation forms a partial order on these sets. The smallest sets according to this partial order can only loop via their input state, otherwise there would be other included SCSs. Note, that ignoring the input states is only one possible heuristic for a decomposition of the graph. An example explaining the SCC heuristic is given later.

During the model checking, we store every (sub-)DTMC together with its abstraction. This is necessary for later concretization where it shall be possible to replace not only a concrete DTMC by its abstraction but vice versa.

Definition 45 (Abstraction pairs) *Given a DTMC $\mathscr{D}$ and a set of non-absorbing state sets $\mathscr{K} = \{K_1, \ldots, K_n\}$ with $K_i \subseteq S$ for all $1 \leq i \leq n$, let Sub be the set of all* abstraction pairs $(\mathscr{D}^{K_i}, \mathscr{D}^{K}_{abs})$.

The general model checking algorithm is depicted in Algorithm 2.

Algorithm 2 Model checking DTMCs

```
**abstract**(DTMC $\mathscr{D}$, Abstractions $Sub_{\mathscr{D}}$)
**begin**
    **for all** non-bottom SCCs $K$ in $\mathscr{D}^{S \setminus Init_{\mathscr{D}}}$ **do**
        DTMC $\mathscr{D}^{K}_{abs} := \textbf{abstract}(\mathscr{D}^{K}, Sub)$
        $\mathscr{D} := \mathscr{D}_{K \mapsto abs}$
        $Sub_{\mathscr{D}} := Sub \cup_{\mathscr{D}} (\mathscr{D}^{K}, \mathscr{D}^{K}_{abs})$
    **end for**
    States $K := \{$*non-absorbing states in* $\mathscr{D}\}$
    $\mathscr{D} := \mathscr{D}_{K \mapsto abs}$
    $Sub_{\mathscr{D}} := Sub_{\mathscr{D}} \cup (\mathscr{D}^{K}, \mathscr{D}^{K}_{abs})$
    **return** $(\mathscr{D}, Sub_{\mathscr{D}})$
**end**

**model_check**(DTMC $\mathscr{D} = (S, I, P, L), T \subseteq S$)
**begin**
    Abstractions $Sub_{\mathscr{D}} := \emptyset$
    $(\mathscr{D}_{abs}, Sub_{\mathscr{D}}) := \textbf{abstract}(\mathscr{D}, Sub_{\mathscr{D}})$
    **return** $\Big( \sum_{s_I \in Init_{\mathscr{D}}} I(s_I) \cdot \big( \sum_{t \in T} P_{abs}(s_I, t) \big), \mathscr{D}_{abs}, Sub_{\mathscr{D}} \Big)$
**end**
```

Parameters

$\mathscr{D}$ is the input DTMC with all BSCCs being single absorbing states

$T \subseteq S$ is the set of absorbing target states

$Sub_{\mathscr{D}}$ stores the substitution pairs according to Definition 45

Return values

abstract returns the abstracted DTMC $\mathscr{D}$ and the abstraction pairs $Sub_{\mathscr{D}}$ as in Definition 45

model_check returns the probability of reaching target states from the initial state

Variables

K is the current set of states to be abstracted, here a non-absorbing SCC of $\mathscr{D}$

$\mathscr{D}^K_{abs}$ stores the abstracted DTMC according to Definition 43

$\mathscr{D}_{K\mapsto abs}$ stores the substitution of the induced DTMC $\mathscr{D}^K$ by $\mathscr{D}^K_{abs}$ according to Definition 44

Procedure

The recursive method abstract(DTMC $\mathscr{D}$, Abstractions $Sub_{\mathscr{D}}$) computes the abstraction $\mathscr{D}_{abs}$ by iterating over all SCCs of the graph when ignoring the input states of $\mathscr{D}$ (Line 1). For each SCC K, the abstraction $\mathscr{D}^K_{abs}$ of the induced DTMC $\mathscr{D}^K$ is computed by a recursive call of the method (Line 2, Definitions 41, 43). Afterwards, $\mathscr{D}^K$ is substituted by its abstraction inside $\mathscr{D}$ (Line 3, Definition 44). Finally, the abstraction of $\mathscr{D}$ is computed and returned (Line 9, Definition 43). Note, that in case of multiple input states $\mathscr{D}$ might still have loops on the input states. This dedicated computation is explained later. This method is called by the model checking method (Line 2) which yields the abstract system $\mathscr{D}_{abs}$, in which transitions lead only from the initial states to the absorbing states. All transitions are labeled with the corresponding reachability probabilities. Then the whole reachability probability is computed by building the sum of these transition probabilities (Line 3). The algorithm returns this reachability property, the abstract DTMC and the abstraction pairs stored in $Sub_{\mathscr{D}}$.

The correctness of Algorithm 2 follows directly from Theorem 3 and Corollary 1: We successively substitute sets of states K according to Definition 43 until only edges leading from the initial states to the target states remain carrying the total reachability probabilities. As we consider only finite states-spaces here, there is a finite number of SCSs in the graph, therefore the algorithm terminates.

What remains to be explained is the computation of the abstract probabilities $p^{\mathscr{D}}_{abs}$. This can be done by standard model checking as explained in Section 2.3.1.1 applied to parts of the DTMC. However, we present a different strategy.

We distinguish the cases where the set K has one or multiple input states.

One input state. Recall the set of paths R_{loop} looping on s_I and the set of paths R_{out} going to some $t \in T$ without revisiting s_I as in Equations 4.2 and 4.3, see the proof for Theorem 3.

Consider a DTMC $\mathscr{D}^K$ induced by K with one initial state s_I and the set of absorbing states $T = \{t^1, \ldots, t^n\}$. We determine the probabilities $p^{\mathscr{D}^K}_{abs}(s_I, t^i)$ for all $1 \leq i \leq n$. As $K \setminus Inp(K)$ has no non-trivial SCSs, the set R_{out} of outgoing paths consists of finitely many loop-free paths. The probability is computed by the following set of equations for all $s \in S^K$:

$$p_{abs}^{\mathscr{D}^K}(s, t^i) = \begin{cases} 1, & \text{if } s = t^i, \\ \sum\limits_{s' \in (succ(s) \cap K \cup Out(K)) \setminus Inp(K)} P^K(s, s') \cdot p_{abs}^{\mathscr{D}^K}(s', t^i), & \text{otherwise.} \end{cases} \tag{4.5}$$

These probabilities can be computed by direct or indirect methods for solving linear equation systems, see, e. g. [QSS00, Chapters 3,4]. Note that also the state elimination as in [HHZ10] can be applied here.

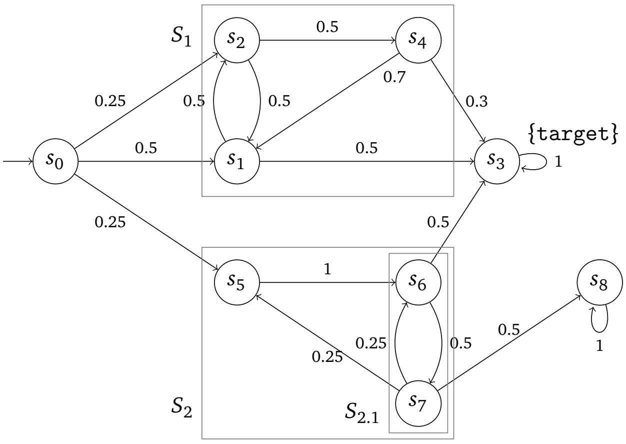
The probabilities of the abstract DTMC $\mathscr{D}_{abs}^K = (S_{abs}^K, I_{abs}^K, P_{abs}^K, L_{abs}^K)$ as in Definition 43 can now directly be computed for all $s_I \in Init_{\mathscr{D}_{abs}^K}$ and $Out(K)$ in $\mathscr{D}$:

$$P_{abs}^{\mathscr{D}^K}(s_I, t) = \frac{p_{abs}^{\mathscr{D}^K}(s_I, t)}{\sum_{t' \in T} p_{abs}^{\mathscr{D}^K}(s_I, t')} \tag{4.6}$$

Remark 16 *In case there is only one absorbing state $t \in S$, we have $p_{abs}^{\mathscr{D}^K}(s_I, t) = 1$. This is directly exploited without further computations.*

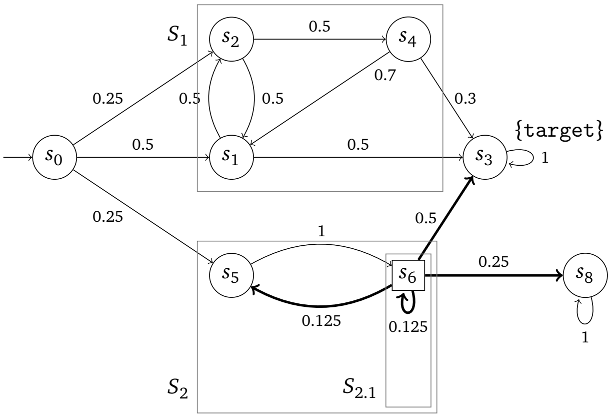
Multiple input states. Given a DTMC $\mathscr{D}^K = (S^K, I^K, P^K, L^K)$ with the set of initial states $S_I = \{s_I^1, \ldots, s_I^m\}$ with $I^K(s_I^i) > 0$ for all $1 \leq i \leq m$ and the set of absorbing states $T = \{t^1, \ldots, t^n\}$. The intuitive idea would be to maintain a copy of $dtmc^K$ for each initial state and handle the other initial states as inner states in this copy. Then, the method as described in the previous paragraph could be used. However, this would be both very time and memory consuming. Therefore, we first formulate the linear equation system as in Equation (4.5). All variables $p_{abs}^{\mathscr{D}^K}(s, s')$ with $s' \in K \setminus Inp(K)$ are eliminated from the equation system. For each of the variables $p_{abs}^{\mathscr{D}^K}(s_I, s')$, the equation system is then solved separately by eliminating all other variables.

Example 10 *we now give a conclusive example about SCC-based model checking. Consider the DTMC $\mathscr{D}$ as in Figure 4.2(a) on Page 59.*



The rectangles indicate the SCC-decomposition of $\mathcal{D}$ with SCCs $S_1 = \{s_1, s_2, s_4\}$ and $S_2 = \{s_5, s_6, s_7\}$. If the input states s_1 and s_2 of S_1 are ignored and another search is performed for $S_1 \setminus \{s_1, s_2\}$, no further SCC is found. In the case of $S_2 \setminus \{s_5\}$, one further SCC $S_{2.1} = \{s_6, s_7\} \subseteq S_2$ is found.

First, we abstract the SCS $S_{2.1}$ as already done in Example 9.



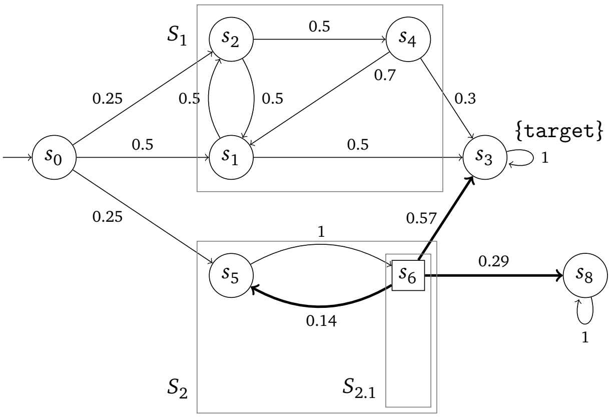
The picture shows the intermediate abstraction for $S_{2.1}$. We depict abstract states hiding SCCs by a rectangular shape and abstract transitions by drawing them thicker then the other ones. The probabilities were already given in Example 9, however, for the sake of completeness we recall them here:

$$
\begin{aligned}
p_{s_6,s_3} &= p_{abs}^{\mathcal{D}^K}(s_6, s_3) = 0.5 & \qquad p_{s_6,s_5} &= p_{abs}^{\mathcal{D}^K}(s_6, s_5) = 0.125 \\
p_{s_6,s_6} &= p_{abs}^{\mathcal{D}^K}(s_6, s_6) = 0.125 & \qquad p_{s_6,s_8} &= p_{abs}^{\mathcal{D}^K}(s_6, s_8) = 0.25
\end{aligned}
$$

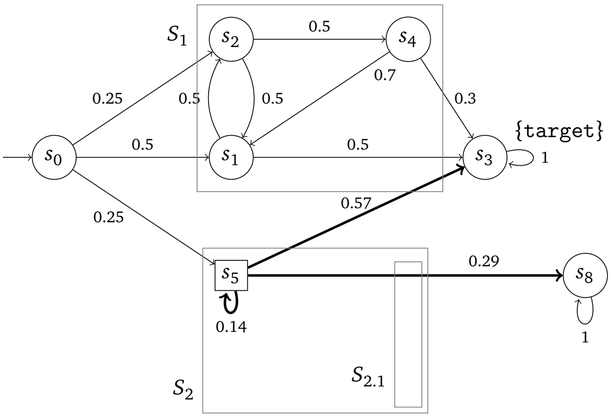
Now we can compute the final abstraction for s_6 where the probability of returning to s_6 is distributed to the outgoing transitions by scaling by the cumulated outgoing probability $p_{out(s_6)} =$

$0.5 + 0.125 + 0.25 = 0.875$:

$$\hat{p}_{s_6,s_3} = 0.5\ /p_{out(s_6)} \approx 0.57 \qquad \hat{p}_{s_6,s_5} = 0.0625\ /p_{out(s_6)} \approx 0.14$$
$$\hat{p}_{s_6,s_8} = 0.25\ /p_{out(s_6)} \approx 0.29$$



Using the results of abstracting $S_{2.1}$, the containing SCC S_2 is now handled.

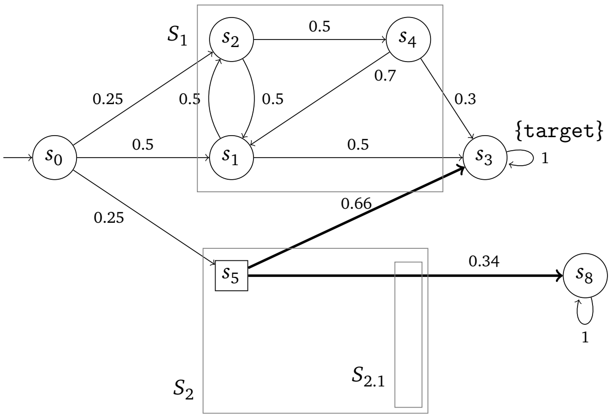


The only input state is s_5, so this will be the abstract state. We have probabilities for the intermediate abstraction:

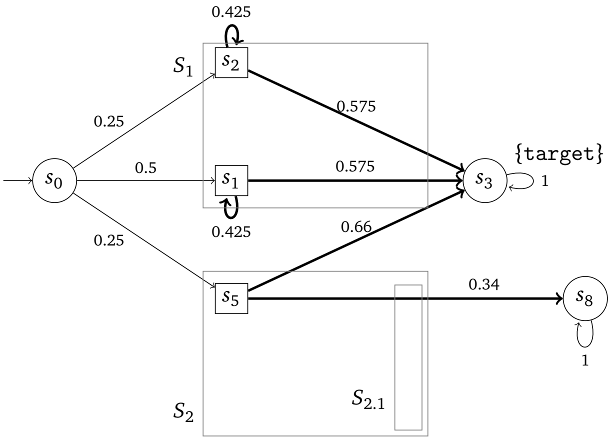
$$p_{s_5,s_3} = p_{abs}^{\mathcal{D}^K}(s_5,s_3) = 0.57 \qquad p_{s_5,s_5} = p_{abs}^{\mathcal{D}^K}(s_5,s_5) = 0.14$$
$$p_{s_5,s_8} = p_{abs}^{\mathcal{D}^K}(s_5,s_8) = 0.29$$

The outgoing accumulated outgoing probability of s_5 is $p_{out(s_5)} = 0.57 + 0.29 = 0.86$. This leads to *the final abstraction of s_5:*

$$\hat{p}_{s_5,s_3} \;=\; 0.57\;/p_{out(s_5)} \approx 0.66 \qquad\qquad \hat{p}_{s_5,s_8} \;=\; 0.308\;/p_{out(s_5)} \approx 0.34$$



We now handle the more complicated case of SCC S_1 which has two input states, s_1 and s_2.



For s_1, all paths that lead to the only output state s_3 without returning to s_1 have to be considered; that includes the paths that visit s_2. We have the following paths and their probabilities:

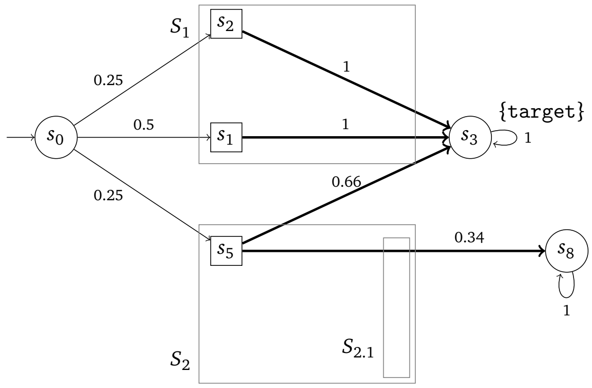
$$\pi_1 = s_1, s_2, s_4, s_3 \qquad Pr^{\mathcal{D}}(\pi_1) = 0.075$$
$$\pi_2 = s_1, s_3 \qquad Pr^{\mathcal{D}}(\pi_2) = 0.5$$

For s_2, the computation is analogous:

$$\pi'_1 = s_2, s_1, s_3 \qquad Pr^{\mathcal{D}}(\pi'_1) = 0.25$$
$$\pi'_2 = s_2, s_4, s_1, s_3 \qquad Pr^{\mathcal{D}}(\pi'_2) = 0.175$$
$$\pi'_3 = s_2, s_4, s_3 \qquad Pr^{\mathcal{D}}(\pi'_3) = 0.15$$

The outgoing probabilities of the abstract states are now computed by adding the path probabilities, yielding the intermediate abstractions:

$$p_{s_1,s_3} = p_{abs}^{\mathscr{D}^K}(s_1,s_3) = 0.575 \qquad p_{s_1,s_1} = p_{abs}^{\mathscr{D}^K}(s_5,s_5) = 0.425$$
$$p_{s_2,s_3} = p_{abs}^{\mathscr{D}^K}(s_2,s_3) = 0.575 \qquad p_{s_2,s_2} = p_{abs}^{\mathscr{D}^K}(s_2,s_2) = 0.425$$



For both states, the cumulated outgoing probability is $p_{out(s_1)} = p_{out(s_2)} = 0.575$. *As* s_3 *is the only output state, the probability of reaching* s_3 *is* 1 *for both states:*

$$\hat{p}_{s_1,s_3} = 0.575\ /p_{out(s_1)} = 1 \qquad \hat{p}_{s_2,s_3} = 0.575\ /p_{out(s_2)} = 1$$

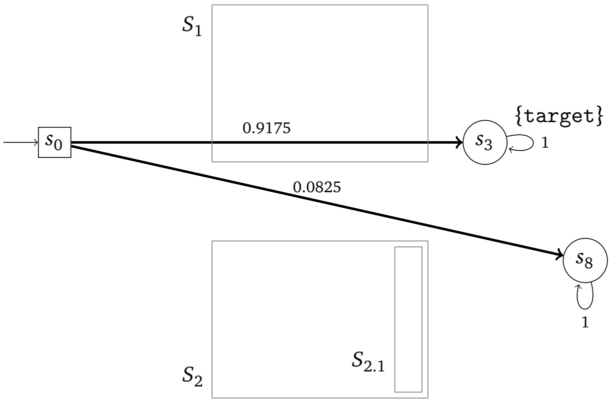
For this acyclic graph, which for convenience we denote by $\mathscr{D}'$, *it remains to compute the path abstraction. The paths to be considered are:*

$$\pi_1 = s_0, s_2, s_3 \qquad Pr^{\mathscr{D}'}(\pi_1) = 0.25$$
$$\pi_2 = s_0, s_1, s_3 \qquad Pr^{\mathscr{D}}(\pi_2') = 0.5$$
$$\pi_3 = s_0, s_5, s_3 \qquad Pr^{\mathscr{D}}(\pi_3') = 0.1675$$

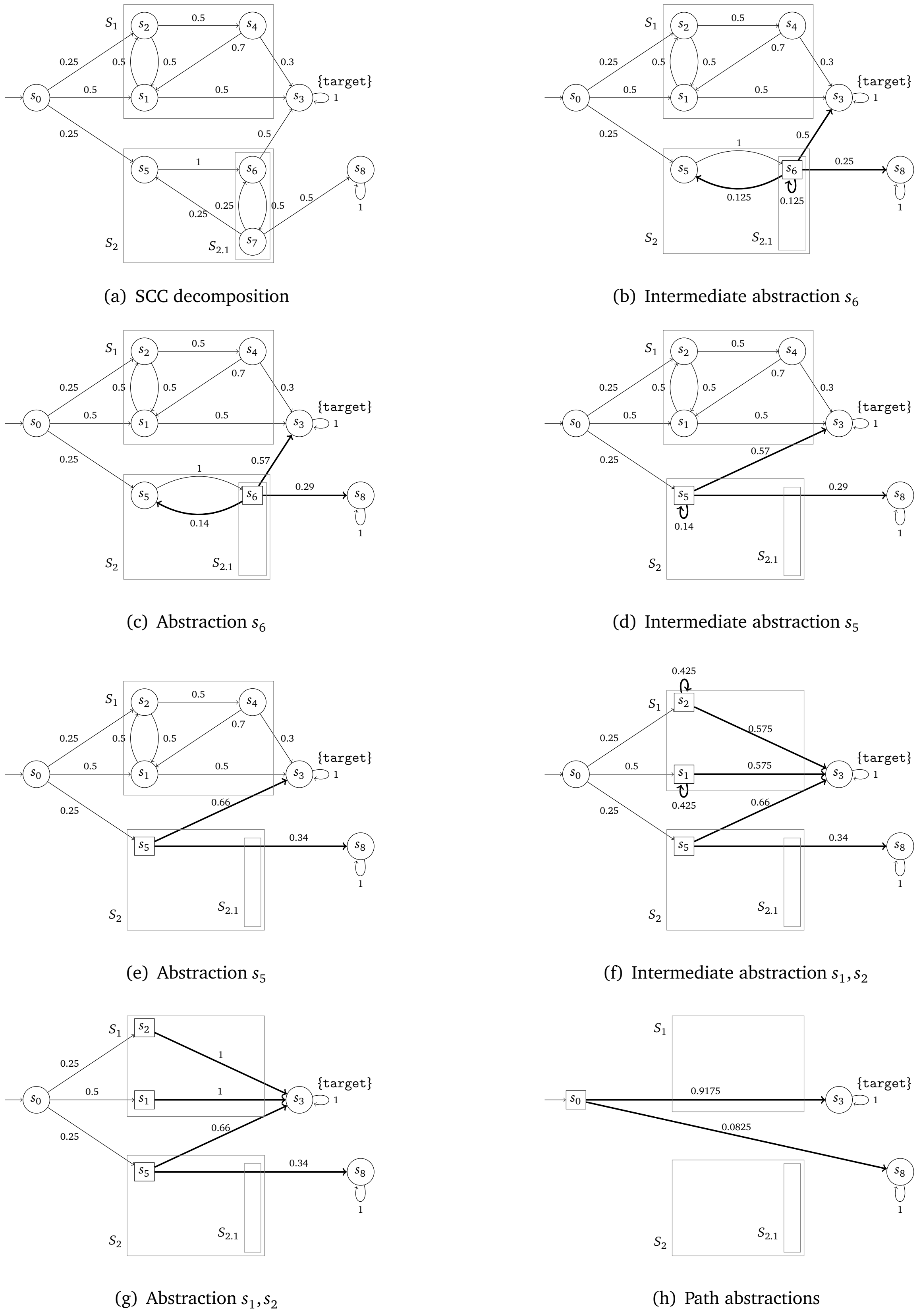
These paths form the set $Paths^{\mathscr{D}'}(s_0, s_3)$, *i. e., the set of paths that lead from the initial to the target states. Therefore, their combined probability mass* 0.9175 *corresponds to the probability of reaching state* s_3 *in* $\mathscr{D}'$ *and by assuming the correctness of the abstraction also in* $\mathscr{D}$. *We have the model checking result:*

$$Pr^{\mathscr{D}}(\lozenge s_3) = 0.9175$$

The resulting abstract graph is depicted below.



For the reader's convenience, we again depict the whole model checking procedure in Figure 4.4.



(a) SCC decomposition

(b) Intermediate abstraction s_6

(c) Abstraction s_6

(d) Intermediate abstraction s_5

(e) Abstraction s_5

(f) Intermediate abstraction s_1, s_2

(g) Abstraction s_1, s_2

(h) Path abstractions

Figure 4.4: SCC-based model checking

4.3 Extension to PDTMCs

We now present the extension of the SCC-based model checking approach to parametric discrete-time Markov Chains (PDTMCs). This approach was published in [2] together with a novel factorization technique for polynomials, which is out of the scope of this thesis.

Basically, all concepts introduced in Section 4.1 can directly be adapted for PDTMCs. The crucial point is to assume for every PDTMC and intermediate computation that all parameters can only be instantiated in a way such that a well-defined DTMC results. Formally, we instantiate variables by defining an *evaluation*.

Definition 46 (Evaluation) *Let V be a set of variables. An* evaluation *u* of *V is a function $u\colon V \to \mathbb{R}$. The evaluation $g[V/u]$ of a polynomial $g \in Pol_V$ under u substitutes each $x \in V$ by $u(x)$, using the standard semantics for $+$ and $\cdot$. For $f = \frac{g_1}{g_2} \in Rat_V$ we define $f[V/u] = \frac{g_1[V/u]}{g_2[V/u]} \in \mathbb{R}$ if $g_2[V/u] \neq 0$.*

In our setting, $g_2[V/u] \neq 0$ means that g_2 is not evaluated to 0. Given an evaluation function, all parameters occurring in the rational functions of a PDTMC can be instantiated as follows.

Definition 47 (Evaluated PDTMC) *For a PDTMC $\mathscr{R} = (S, V, I, \mathfrak{P}, L)$ and an evaluation u, the* evaluated PDTMC *is the tuple $\mathscr{D}_u = (S_u, I_u, P_u, L_u)$ given by:*

- $S_u = S$
- $I_u\colon S_u \to \mathbb{R}$ *with* $I_u(s) = I(s)[V/u]$
- $P_u\colon S \times S \to \mathbb{R}$ *with* $P_u(s,s') = \mathfrak{P}(s,s')[V/u]$ *for all* $s, s' \in S_u$
- $L_u = L$

An evaluation u substitutes each parameter by a real number yielding concrete values for all transition probabilities. By ensuring that the parameters are assigned values in such a way that the evaluated PDTMC is a DTMC, this directly induces a probability measure for the evaluated PDTMC. Due to algorithmic reasons, we allow only stochastic distributions for the initial distribution and the transition probability matrix.

Remark 17 (Partial evaluation) *It is also possible to define partial evaluations where not all of the parameters are replaced by real numbers. However, for the algorithms we propose this is not necessary.*

Definition 48 (Well-defined evaluation) *Let $\mathscr{R} = (S, V, I, \mathfrak{P}, L)$ be a PDTMC. An evaluation u of V is* well-defined *if for the evaluated PDTMC $\mathscr{D}_u = (S_u, I_u, P_u, L_u)$ it holds that:*

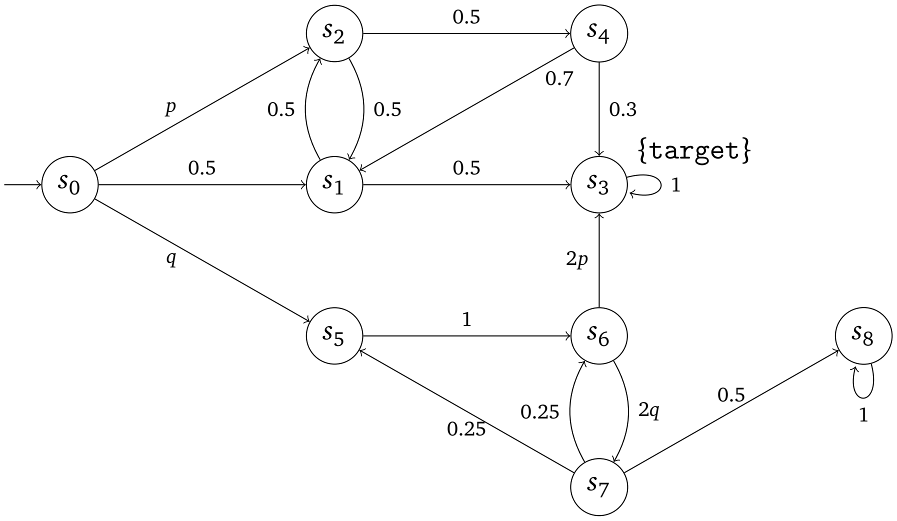


Figure 4.5: Example PDTMC with parameters p and q

- $I_u : S_u \to [0,1]$ *with* $\sum_{s \in S_u} I_u(s) = 1$
- $P_u : S_u \times S_u \to [0,1]$ *with* $\forall s \in S_u.\ \sum_{s' \in S_u} P_u(s,s') = 1$

A well-defined evaluation u is called graph preserving, *if it holds that:*

- $\forall s \in S.\, I(s) \neq 0 \implies I_u(s) > 0$
- $\forall s,s' \in S.\, P(s,s') \neq 0 \implies P_u(s,s') > 0$

Note that we require $P_u(s,s') \in \mathbb{R}$ for well-defined evaluations, i. e., no division by 0 will occur during the computations. For our algorithms, we always require the evaluation u to be graph-preserving, i. e., $\mathcal{G}_{\mathcal{R}} = \mathcal{G}_{\mathcal{D}_u}$. This is necessary as by altering the graph certain states might become unreachable.

Example 11 (PDTMC) *Consider the PDTMC in Figure 4.5 as in Example 5 on Page 31. A well-defined evaluation for these parameters would impose the following constraints:* $p + q + 0.5 = 1$ *and* $2p + 2q = 1$. *Intuitively, by increasing values of p the probability of reaching the target state will increase. An evaluation u with* $u(p) = 0.25$ *and* $u(q) = 0.25$ *is well-defined and graph preserving.*

The goal is now to compute *reachability probabilities* for PDTMCs. The common approach is to compute a rational function which describes the probability to reach a set of target states from the initial states, see [HHZ10] and Section 3.3. The formal definition of the model checking problem for parametric DTMCs reads as follows.

Definition 49 *Given a PDTMC* $\mathcal{R} = (S, V, I, \mathfrak{P}, L)$ *where all BSCCs are absorbing states* $T \subseteq S$, *the* parametric probabilistic model checking problem *is to find for each initial state* $s_I \in Init_{\mathcal{R}}$ *and each* $t \in T$ *a rational function* $f_{s_I,t} \in Rat_V$ *such that for all graph-preserving evaluations*

$u : V \to \mathbb{R}$ and the evaluated PDTMC $\mathscr{R}_u = (S_u, V_u, I_u, \mathfrak{P}_u, L_u)$ it holds that $f_{s_I,t}[V/u] = \Pr^{\mathscr{R}_u}\big(Paths^{\mathscr{R}_u}(s_I, t)\big)$.

Given the functions $f_{s_I,t}$ for all $s_I \in Init_{\mathscr{R}}$ and $t \in T$, the probability of reaching a state in T from an initial state is $\sum_{s_I \in Init_{\mathscr{R}}} I(s_I) \cdot \left(\sum_{t \in T} f_{s_I,t}\right) \in Rat_V$.

we now discuss under which prerequisites the SCC-based abstraction can be directly adapted to PDTMCs using the aforementioned concepts of well-defined evaluations.

First, the adaption of Definition 41 for forming an induced PDTMC $\mathscr{R}^K$ from a subset of states $K \subseteq S$ is considered. This is straightforward, as we only operate on the induced graph of the PDTMC or DTMC, respectively. We have to take care that for the induced PDTMC it is required that $I(s) \neq 0$ for all states $s \in S$. In this context this means that the rational function $I(s)$ cannot be simplified to 0. Moreover, in a corresponding parameter instantiation it has to be taken care of that no value less or equal to 0 is derived which is ensured by requiring graph-preserving evaluations.

Consider now Definition 42 and Definition 43 concerning abstracted DTMCs. For a PDTMC $\mathscr{R}$ and two states $s, s' \in S$, the abstract probability $p^{\mathscr{R}}_{abs}(s, s')$ is computed by multiplication and addition of the rational functions describing the corresponding transition probabilities along paths from s to s'. Definition 43 also involves the division of rationals functions.

Remark 18 (Operations on rational functions) *Note that performing operations on rational functions such as multiplication and especially addition is quite costly. The latter involves the computation of the greatest common divisor (gcd). Applying the approaches as described for DTMCs directly to PDTMCs does not scale very well. Therefore, these operations were sped up by maintaining a (partial) factorization of every occurring polynomial. For details we refer to [2].*

Theorem 3 holds for PDTMCs under the condition that only graph-preserving evaluations are used to instantiate parameters when evaluating the resulting rational function. This evaluation can be done by a *SAT modulo theories* solver which can handle non-linear arithmetic over the reals [JdM12]. In order to allow only graph-preserving evaluations, we perform a preprocessing where conditions are added according to Definition 48 as well as the ones from Equation 4.6. In case the solver returns an evaluation which satisfies the resulting constraint set, the property is satisfied. If the solver returns unsatisfiability, the property does not hold. If the solver is capable of returning a minimal unsatisfiability core, i. e., a minimal parameter evaluation for which the constraints are already violated, this can serve as a *counterexample* to the property.

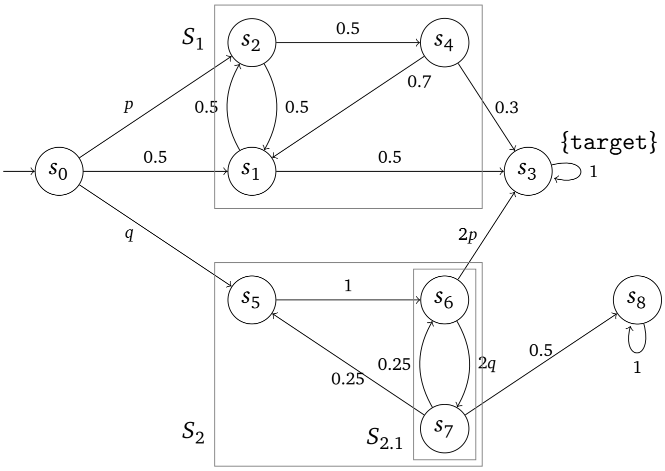
Example 12 *Reconsider the PDTMC as in Example 5 on Page 20 to be seen below. This is a parametric version of the DTMC in Example 10. We are again interested in computing the probability of reaching the absorbing state s_3 from the initial state s_0. In a preprocessing step, we identify the following constraints that need to be satisfied by any graph-preserving evaluation for this PDTMC:*

$$p + q + 0.5 = 1 \tag{4.7}$$

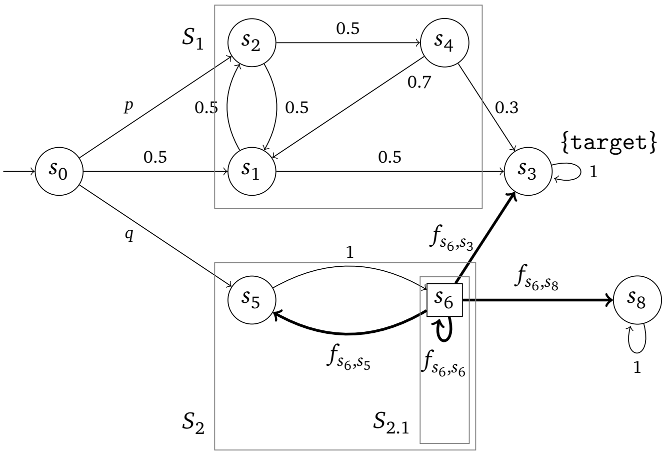
$$2p + 2q = 1 \tag{4.8}$$

$$p > 0 \land p \leq 1 \tag{4.9}$$
$$q > 0 \land q \leq 1 \tag{4.10}$$
$$2p > 0 \land 2p \leq 1 \tag{4.11}$$
$$2q > 0 \land 2q \leq 1 \tag{4.12}$$

Note that some of these constraints might be redundant. we now again show all the (identical) steps of the SCC-based model checking tailored to computing rational functions.



SCC $S_{2.1}$ is abstracted first.

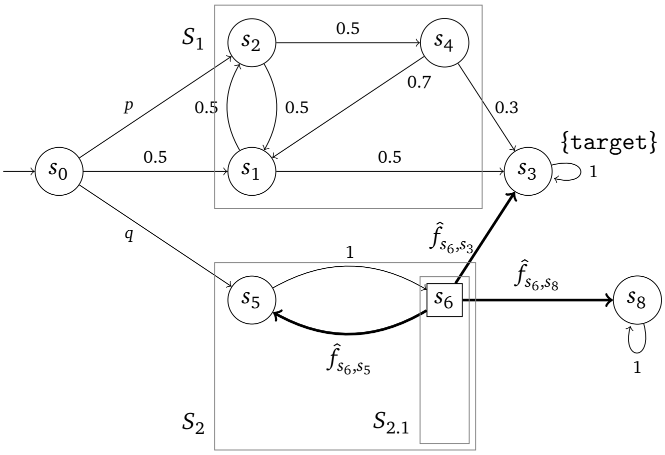


The intermediate abstraction for $S_{2.1}$ is as follows. $f_{s,s'}$ denotes the rational function representing the probability of going from state s to s'. The cumulated outgoing probability of s_5 is $f_{out(s_6)} = 2.5p + q$. Subtracting this function from 1 yields the probability of returning to s_6.

$$\begin{aligned} f_{s_6,s_3} &= p_{abs}^{\mathscr{D}^K}(s_6,s_3) = 2p & f_{s_6,s_5} &= p_{abs}^{\mathscr{D}^K}(s_6,s_5) = 0.5q \\ f_{s_6,s_6} &= p_{abs}^{\mathscr{D}^K}(s_6,s_6) = 1-(f_{s_6,s_3}+f_{s_6,s_5}+f_{s_6,s_8}) & f_{s_6,s_8} &= p_{abs}^{\mathscr{D}^K}(s_6,s_8) = q \end{aligned}$$

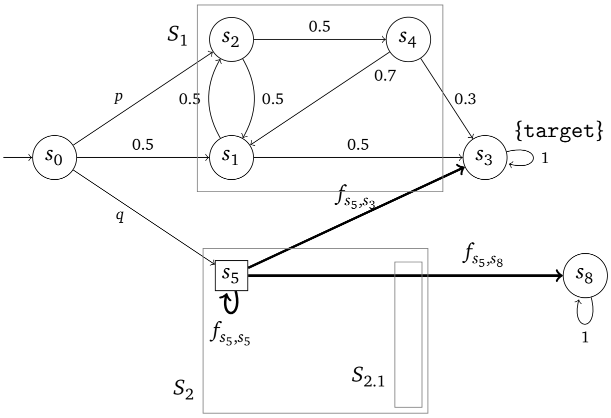
The final abstraction for s_6 is built by distributing the probability of returning to s_6 to the outgoing *transitions. This is—as for mere DTMCs—done by scaling via the cumulated outgoing probability $f_{out(s_6)}$:*

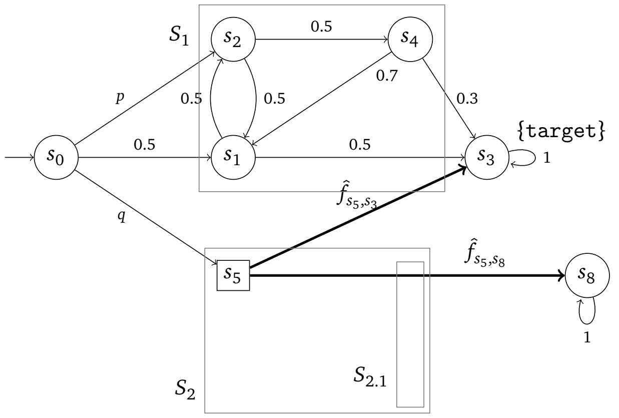
$$\hat{f}_{s_6,s_3} = \frac{2p}{2p+1.5q} \qquad \hat{f}_{s_6,s_5} = \frac{0.5q}{2p+1.5q} \qquad \hat{f}_{s_6,s_8} = \frac{q}{2p+1.5q}$$



Using the results of abstracting $S_{2.1}$, the functions for SCC S_2 can now be computed. The functions for the intermediate abstraction of s_5 are:

$$f_{s_5,s_3} = \hat{f}_{s_6,s_3} = \frac{2p}{2p+1.5q} \qquad f_{s_5,s_5} = 1-(f_{s_5,s_3}+f_{s_5,s_8})$$
$$f_{s_5,s_8} = \hat{f}_{s_6,s_8} = \frac{q}{2p+1.5q}$$

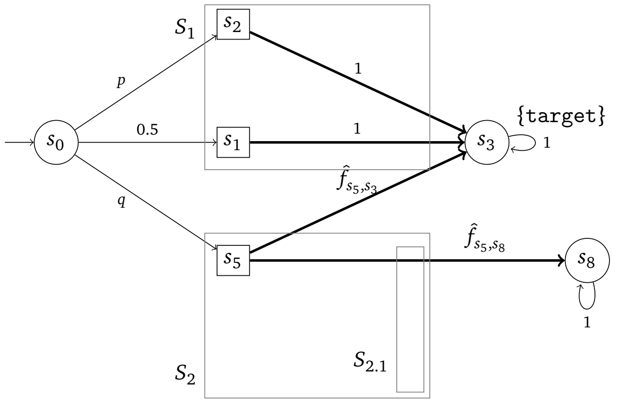




The accumulated outgoing probability of s_5 *is* $f_{out(s_5)} = f_{s_5,s_3} + f_{s_5,s_8} = \frac{2p+q}{2p+1.5q}$. *This leads to the final abstraction of* s_5*:*

$$\hat{f}_{s_5,s_3} = \frac{f_{s_5,s_3}}{f_{out(s_5)}} = \frac{2p \cdot \cancel{(2p+1.5q)}}{\cancel{(2p+1.5q)} \cdot (2p+q)} \qquad \hat{f}_{s_5,s_8} = \frac{f_{s_5,s_8}}{f_{out(s_5)}} = \frac{q \cdot \cancel{(2p+1.5q)}}{\cancel{(2p+1.5q)} \cdot (2p+q)}$$

SCC S_1 *with the two input states* s_1 *and* s_2 *does not contain any parameters. Therefore, we will have the result as in Example 10 where we don't need any additional constraints.*



The functions, in this case being constant, are as follows.

$$\hat{f}_{s_1,s_3} = 1 \qquad \hat{f}_{s_2,s_3} = 1$$

Now we compute the path abstraction for the acyclic graph $\mathscr{R}'$. *The paths leading to the target state* s_3 *are:*

$$\pi_1 = s_0, s_2, s_3 \qquad Pr^{\mathscr{R}'}(\pi_1) = p$$

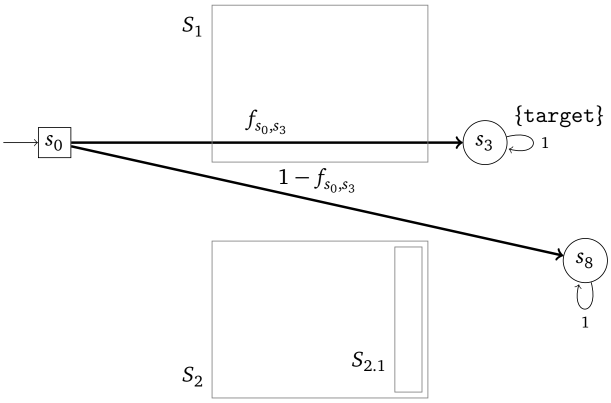
$$\pi_2 = s_0, s_1, s_3 \qquad Pr^{\mathscr{R}'}(\pi_2') = 0.5$$

$$\pi_3 = s_0, s_5, s_3 \qquad Pr^{\mathscr{R}'}(\pi_3') = q \cdot \hat{f}_{s_5,s_3} = \frac{2pq}{2p+q}$$

These paths form the set $Paths^{\mathscr{R}'}(s_0, s_3)$*, i. e., the set of paths that lead from the initial to the target* *state. Therefore, their combined probability mass corresponds to the probability of reaching state s_3 in $\mathscr{R}'$ and by assuming the correctness of the abstraction also in $\mathscr{R}$. We have the model checking result:*

$$\begin{aligned}
Pr^{\mathscr{D}}(\Diamond s_3) &= Pr^{\mathscr{R}'}(\pi_1) + Pr^{\mathscr{R}'}(\pi_2) + Pr^{\mathscr{R}'}(\pi_3) && (4.13)\\
&= p + 0.5 + \frac{2pq}{2p+q} && (4.14)\\
&= \frac{p(2p+q) + 0.5(2p+q) + 2pq}{2p+q} && (4.15)\\
&= \frac{2p^2 + 3pq + p + 0.5q}{2p+q} && (4.16)\\
&= f_{s_0,s_3} && (4.17)
\end{aligned}$$

The resulting abstract graph is depicted below.



Assume now an upper probability bound $\lambda \in [0,1] \subseteq \mathbb{R}$. If we find a satisfying assignment for the parameters p and q such that for the rational function it holds that $f_{s_0,s_3} \leq \lambda$ with respect to constraints 4.7- 4.12, the property $\mathbb{P}_{\leq\lambda}(\Diamond s_3)$ is satisfied at the initial state s_0 of $\mathscr{R}$.

CHAPTER 5

Path-based counterexample generation

Summary In this chapter we present concepts for the computation of counterexamples for DTMCs and violated probabilistic reachability properties. All methods are heuristic algorithms which are based on the search for certain paths of a DTMC. A counterexample is now not longer solely based on a set of paths whose combined probability mass violates a certain property, see Section 2.5. Instead, we introduce a sort of symbolic representation by the notion of a *critical subsystem*, which is a critical part of the original system where the property at hand is already violated. Critical subsystems were introduced in [AL10, 10].

Following the SCC-based abstraction scheme, see Chapter 4, we develop the notion of *hierarchical counterexamples*. Embedded into a hierarchical abstraction scheme, critical subsystems are computed on the abstraction. These abstract subsystems can hide the loop-structure of the original system to any degree. If more details are needed, a refinement is possible up to the original, concrete system. Together with dedicated path searching algorithms, this was presented in ATVA 2011 [10].

Two main search algorithms have been investigated: The *global search approach*, where most probable paths are found that lead from the initial state to target states of an input DTMC, and the *fragment search approach*, where most probable path fragments connect already examined parts of original DTMC. Both of these approaches have been introduced in [10, 16] for explicit graph representations and embedded in the hierarchical DTMC-abstraction. Experimentally, improvements in orders of magnitude could be achieved in comparison to other approaches.

These concepts were adapted to symbolic graph representations on the one hand utilizing SAT solvers for a bounded model checking approach and on the other hand using symbolic graph algorithms. A first version of these extensions was presented in [7]. The results presented in this thesis are improvements that are published in [3]. With these improvements, the generation of counterexamples is pushed to DTMCs with billions of states.

Outline First, we introduce the notion of critical subsystems in Section 5.1. Hierarchical counterexamples are introduced in Section 5.2. A framework for generating critical subsystems on explicit graph representations is given in Section 5.3 as well as a framework for symbolic graph representations in Section 5.4. we present methods to generate these subsystems based on explicit graph search in Section 5.5, by SAT-based algorithms in Section 5.6, and by symbolic graph algorithms in Section 5.7.

Background The following foundations are needed in this section: Model checking reachability properties of DTMCs (Section 2.3.1.1), basic operations on BDDs and MTBDDs (Section 2.4.1), and symbolic representations of DTMCs (Section 2.4.2). Furthermore, we use Dijkstra's algorithm for computing the shortest path in a weighted directed graph [Dij59]. Finally, we need an algorithm for computing the k shortest paths, in particular the recursive enumeration algorithm by Jimenez and Marzal [JM99].

5.1 Counterexamples as critical subsystems

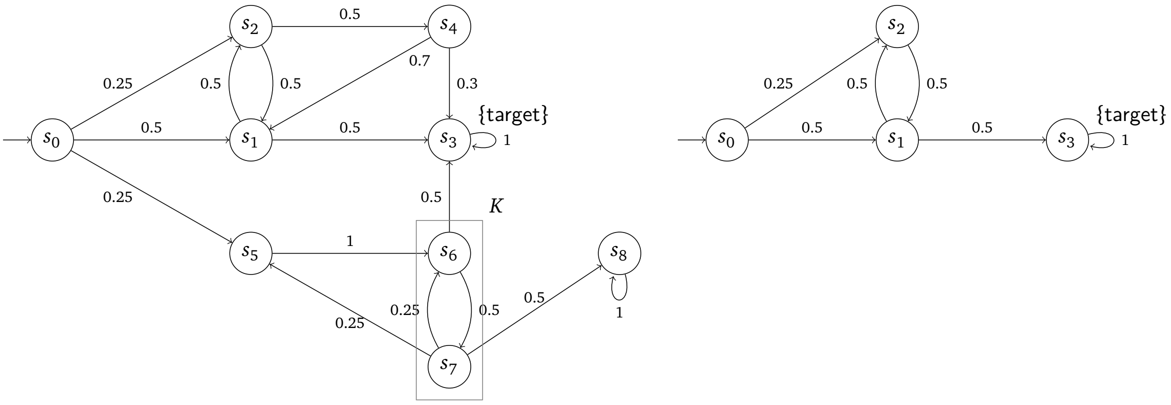
As mentioned before, the representation of counterexamples as a set of paths, see Section 2.5, has two major drawbacks: The number of paths needed might be very large or even infinite, and as a consequence, the number of search iterations in terms of path searches is equally high. Consider therefore again Example 7 on Page 47, where for a relatively simple system an infinite number of paths is needed to form a counterexample.

An alternative is to use what we call *critical subsystems*, which are subsystems of DTMCs, see Definition 6 on Page 22. These subsystems also violate the property that is violated by the original DTMC. As we see in our experimental evaluations, it is often possible to generate critical subsystems whose size is smaller by orders of magnitude in comparison to the input system. Thereby, the *critical part* of the original system leading to the violation is highlighted. Recall that we only consider counterexamples for reachability properties, see Section 2.5.

Definition 50 (Critical subsystem) *Given a DTMC $\mathscr{D} = (S, I, P, L)$ and a PCTL state formula $\psi = \mathbb{P}_{\leq\lambda}(\lozenge T)$ such that $\mathscr{D} \not\models \psi$. A* critical subsystem *of $\mathscr{D}$ for ψ is a subsystem $\mathscr{D}' \sqsubseteq \mathscr{D}$ with $\mathscr{D}' \not\models \psi$.*

If all relevant paths for violating the property inside the subsystem are considered, these paths form a counterexample in the classical sense as in Definition 35 on Page 45. Thereby, a critical subsystem can be seen as a symbolic representation of a counterexample.

Definition 51 (Induced counterexample) *For a DTMC $\mathscr{D}$, a PCTL state formula $\psi = \mathbb{P}_{\leq\lambda}(\lozenge T)$, and a critical subsystem $\mathscr{D}' \sqsubseteq \mathscr{D}$, a counterexample is given by the set of paths $C_\psi := Paths_{fin}^{\mathscr{D}'}(Init_{\mathscr{D}}, T)$.*



(a) DTMC $\mathcal{D}$

(b) Subsystem $\mathcal{D}' \sqsubseteq \mathcal{D}$

Figure 5.1: DTMC $\mathcal{D}$ and subsystem $\mathcal{D}' \sqsubseteq \mathcal{D}$

Example 13 *Consider the DTMC of Figure 2.1 on Page 20 again depicted in Figure 5.1(a). The probability of reaching the target state s_3 is 0.9175. We are interested in a counterexample for the property $\psi = \mathbb{P}_{\leq 0.4}(\lozenge s_3)$. Building a smallest path-based counterexample as in [HKD09], see Section 3.1.1, would yield the following 4 most probable paths and their probabilities:*

$$\begin{aligned} \pi_1 &= s_0, s_1, s_3 & \quad Pr^{\mathcal{D}}(\pi_1) &= 0.25 \\ \pi_2 &= s_0, s_5, s_6, s_3 & \quad Pr^{\mathcal{D}}(\pi_2) &= 0.125 \\ \pi_1 &= s_0, s_2, s_1, s_3 & \quad Pr^{\mathcal{D}}(\pi_1) &= 0.0625 \end{aligned}$$

These paths together have probability 0.4375, so this set forms a counterexample for ψ.

As an alternative, consider the subsystem $\mathcal{D}' \sqsubseteq \mathcal{D}$ in Figure 5.1(b). The probability of reaching s_3 inside $\mathcal{D}'$ is $0.41\bar{6}$. So, this small subsystem is critical *for ψ. For higher probability bounds near the actual model checking probability, the set of paths would grow rapidly, while the subsystem can never be larger than the original system in terms of the number of states.*

Remark 19 (Size of induced counterexample) *Even if a critical subsystem for a DTMC is minimal in size, e. g., in terms of the number of states or transitions, this does not induce a smallest or minimal counterexample as in [HKD09], see Section 3.1. Consider therefore again Example 13, where all paths from s_0 to s_3 inside $\mathcal{D}'$ form the induced counterexample. As there are infinitely many of such shortest evidences, see Definition 35 on Page 45, the size of the counterexample is not minimal.*

Critical subsystems can be generated in various ways. In this chapter, they are computed incrementally by adding transitions or states to an initial (possibly) empty) subsystem until it becomes critical. In every computation step, a set of transitions or states of the original system is used to build a subsystem which is induced by this set and the previous subsystem.

We have to conceptually distinguish subsystems induced by transitions or by states. If a set of transitions is considered, the subsystem consists if the source and destination states connected by these transitions. For states, we take the states and all transitions connecting them.

We first define the extension of a subsystem by a set of transitions. Recall that $E_{\mathscr{D}}$ denotes the set of transitions of a DTMC $\mathscr{D}$, see Definition 5 on Page 21.

Definition 52 (Extension of subsystem by transitions) *Let* $\mathscr{D} = (S, I, P, L)$ *and* $\mathscr{D}' = (S', I', P', L')$ *be DTMCs with* $\mathscr{D}' \sqsubseteq \mathscr{D}$ *and let* $E_e \subseteq E_{\mathscr{D}}$ *be a set of transitions of* $\mathscr{D}$. *extend*$(\mathscr{D}', E_e) = \mathscr{D}'' = (S'', I'', P'', L'')$ *is the* extension of $\mathscr{D}'$ by E_e *with:*

- $S'' = S' \cup \{s \in S \mid \exists s' \in S. ((s, s') \in E_e \vee (s', s) \in E_e)\}$
- $I''(s) = I(s)$ *for all* $s \in S''$
- $P''(s, s') = \begin{cases} P(s, s') & \text{for all } (s, s') \in E_e \cup E_{\mathscr{D}'} \\ 0 & \text{for all } (s, s') \in (S'', S'') \setminus (E_e \cup E_{\mathscr{D}'}) \end{cases}$
- $L''(s) = L(s)$ *for all* $s \in S''$

Intuitively, exactly those transitions are added to the subsystem that are in E_e together with their source and destination states. Only original transitions from $\mathscr{D}$ can be added.

Analogously, we define the extension by a set of states. The only difference is that now *all* induced edges are added, i. e., all edges that are adjacent to states of the extended subsystem.

Definition 53 (Extension of a subsystem by states) *Let* $\mathscr{D} = (S, I, P, L)$ *be a DTMC and* $\mathscr{D}' = (S', I', P', L')$ *be a subsystem* $\mathscr{D}' \sqsubseteq \mathscr{D}$ *of* $\mathscr{D}$ *and let* $S_e \subseteq S$ *be a set of states of* $\mathscr{D}$. *extend*$(\mathscr{D}', S_e) = \mathscr{D}'' = (S'', I'', P'', L'')$ *is the* extension of $\mathscr{D}'$ by S_e *with:*

- $S'' = S' \cup S_e$
- $I''(s) = I(s)$ *for all* $s \in S''$
- $P''(s, s') = P(s, s')$ *for all* $s, s' \in S''$
- $L''(s) = L(s)$ *for all* $s \in S''$

The states from S_e are added to the subsystem $\mathscr{D}'$. All transitions that are between both the new and old states of $\mathscr{D}'$ are implicitly added, including self-loops.

Analogously to extending a system, a DTMC can be restricted to a certain set of edges or states. We give the formal definition of the restriction by transitions.

Definition 54 (Restriction of a DTMC by transitions) *Let $\mathscr{D} = (S, I, P, L)$ be a DTMC and $E_r \subseteq E_{\mathscr{D}}$ be a set of transitions of $\mathscr{D}$. restrict$(\mathscr{D}, E_r) = \mathscr{D}' = (S', I', P', L')$ is the restriction of $\mathscr{D}$ to E_r with:*

- $S' = \{s \in S \mid \exists s' \in S.\, (s, s') \in E_r \lor (s', s) \in E_r\}$
- $I'(s) = I(s)$ *for all* $s \in S'$
- $P'(s, s') = \begin{cases} P(s, s') & \text{if } (s, s') \in E_r \\ 0 & \text{otherwise} \end{cases}$
- $L'(s) = L(s)$ *for all* $s \in S'$

Remark 20 *The extension and the restriction of DTMCs pose no formal problems as we allow sub-stochastic distributions for the DTMCs, see Definition 3 on Page 19. Therefore it does not matter if the sum of outgoing probabilities of a state is less than* 1 *or even* 0*.*

5.2 Hierarchical counterexample generation

Based on the SCC abstraction described in Chapter 4 we present a new method for generating counterexamples on possibly abstract input DTMCs. A counterexample, here a critical subsystem, is computed in a *hierarchical* manner by offering the possibility of starting the computation on simple abstract graphs and concretizing important parts in order to explore the system in more detail.

This allows to search for counterexamples on very large input graphs, as the abstract input systems are often very small and simply structured. As concretization up to the original system can be restricted to the critical parts, no information is lost and yet not the whole system has to be explored.

We first introduce the formalisms that are needed to perform both the abstraction and reverse operation, the *concretization*, according to the SCC-based abstraction. Using this, we present the general algorithm for the hierarchical counterexample generation.

5.2.1 Concretizing abstract states

Let us recall Chapter 4, where the substitution of a non-absorbing set of states by its abstraction was introduced, see Definition 44 on Page 63. In order to allow for a more general setting, we can identify induced DTMCs as in Definition 41, see Page 58, and replace them either by their abstractions or by their concretizations, respectively.

Definition 55 (DTMC substitution) *Assume a DTMC $\mathscr{D} = (S, I, P, L)$, a non-absorbing set of states $K \subseteq S$ and the induced DTMC $\mathscr{D}^K = (S^K, I^K, P^K, L^K)$. Let $\mathscr{D}' = (S', I', P', L')$ be a*

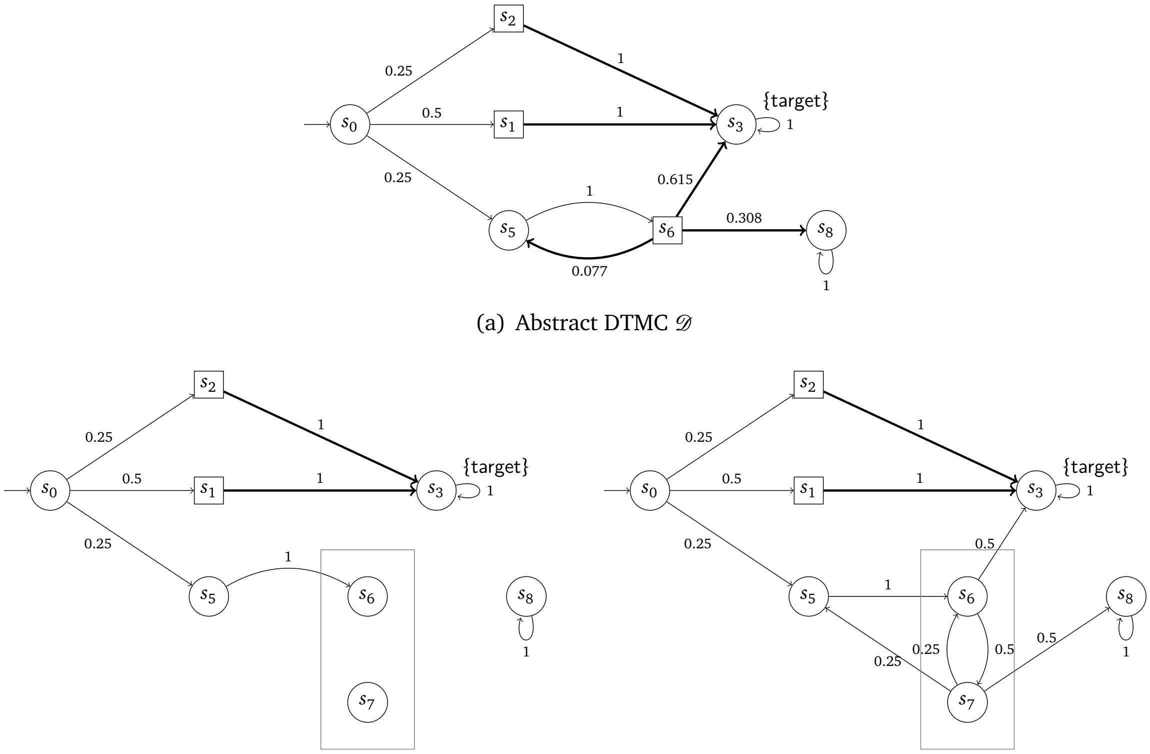
DTMC with $Init_{\mathscr{D}'} = Init_{\mathscr{D}^K}$, $Out_{\mathscr{D}}(K) \subseteq S'$ absorbing in $\mathscr{D}'$, and $S' \setminus Out_{\mathscr{D}}(K)$ non-absorbing and disjoint from $S \setminus K$. Then the substitution of $\mathscr{D}^K$ by $\mathscr{D}'$ in $\mathscr{D}$ *is given by* $\mathscr{D}_{\mathscr{D}^K \mapsto \mathscr{D}'} = (S_{\mathscr{D}^K \mapsto \mathscr{D}'}, I_{\mathscr{D}^K \mapsto \mathscr{D}'}, P_{\mathscr{D}^K \mapsto \mathscr{D}'}, L_{\mathscr{D}^K \mapsto \mathscr{D}'})$ *with:*

- $S_{\mathscr{D}^K \mapsto \mathscr{D}'} = (S \setminus K) \cup S'$
- $\forall s, s' \in S_{\mathscr{D}^K \mapsto \mathscr{D}'}. I_{\mathscr{D}^K \mapsto \mathscr{D}'}(s) = I(s)$
- $P_{\mathscr{D}^K \mapsto \mathscr{D}'}(s,s') = \begin{cases} P(s,s') & \text{if } s \notin K \\ P'(s,s') & \text{if } s,s' \in S' \\ 0 & \text{otherwise} \end{cases}$
- $L_{\mathscr{D}^K \mapsto \mathscr{D}'}(s) = \begin{cases} L(s) & \text{if } s \notin K \\ L'(s) & \text{if } s \in S' \\ \emptyset & \text{otherwise.} \end{cases}$

Note that $S^K = K \uplus Out_{\mathscr{D}}(K)$, see Definition 41 on Page 58. Having this, we now present a method to compute the concretization of an abstract DTMC $\mathscr{D}$ resulting from the SCC-based model checking procedure, see Algorithm 2 on Page 64. Recall that we stored every abstraction pair for the DTMC $\mathscr{D}$ in the set $Sub_{\mathscr{D}}$, see Definition 45 on Page 64.

Remark 21 (Abstract states) *An abstract DTMC consists of one or more input states that have transitions to other states. These states are replaced by their concretization. In what follows we also speak of* abstract states.

We now present the concretization algorithm dedicated to the hierarchical counterexample generation. Intuitively, a number of abstract states are replaced by their concrete counterpart according to the abstraction pairs that were saved during SCC-based model checking. In the resulting DTMC, which we call $\mathscr{D}_{max}$, the probabilities of reaching target states is preserved so this concretized system can be seen as an upper bound on a critical subsystem that could be computed. In order to achieve a refined solution, we also maintain a DTMC $\mathscr{D}_{min}$ where all concretized parts are “cut out”. This forms a lower bound on the critical subsystem we want to compute. The algorithm is given in Algorithm 3.



(a) Abstract DTMC $\mathscr{D}$

(b) DTMC $\mathscr{D}_{min}$ after concretization of s_6

(c) DTMC $\mathscr{D}_{max}$ after concretization of s_6

Figure 5.2: Concretizing state s_6, resulting DTMCs $\mathscr{D}_{min}$ and $\mathscr{D}_{max}$

Algorithm 3

```
Concretize(DTMC $\mathscr{D}$, Abstractions $Sub_{\mathscr{D}}$)
begin
    Boolean concretized := false
    DTMC $\mathscr{D}_{min} := \emptyset$        DTMC $\mathscr{D}_{max} := \mathscr{D}$
    State $s_a := \bot$
    while true do
        $s_a :=$ FindAbstractState($\mathscr{D}, Sub_{\mathscr{D}}$)
        if ($s_a = \bot$) then
            return (concretized, $\mathscr{D}_{min}, \mathscr{D}_{max}$)
        else
            concretized := true
            Let $(\mathscr{D}_{abs}, \mathscr{D}_{conc}) \in Sub_{\mathscr{D}}$ such that $s_a \in Init_{\mathscr{D}_{abs}}$
            $\mathscr{D}_{min} :=$ Restrict($\mathscr{D}, E_{\mathscr{D}} \setminus E_{\mathscr{D}_{abs}}$)
            $\mathscr{D}_{max} := \mathscr{D}_{\mathscr{D}_{abs} \mapsto \mathscr{D}_{conc}}$
        end if
    end while
end
```

Parameters

$\mathscr{D}$ is an abstract DTMC $\mathscr{D} = (S, I, P, L)$ which results from SCC-based model checking, see Chapter 4.

$Sub_{\mathscr{D}}$ contains all abstraction pairs $(\mathscr{D}_{abs}, \mathscr{D}_{conc})$ resulting from SCC-based model checking, see Definition 45 on Page 64.

Variables

`concretized` is a flag and indicates whether at least one state was concretized or not.

$\mathscr{D}_{min}$ represents the DTMC $\mathscr{D}$ where all abstract states due for concretization are removed. Intuitively, this DTMC has "holes" at these parts such that $\mathscr{D}_{min}$ puts a lower bound on the reachability probability with respect to the concretization.

$\mathscr{D}_{max}$ represents the DTMC $\mathscr{D}$ where the chosen abstract states are replaced by their concretizations.

s_a represents the current abstract state which is to be concretized. Its reachability probability is the maximal one that can be achieved by a counterexample.

Return value

The algorithm returns the DTMCs $\mathscr{D}_{min}$ and $\mathscr{D}_{max}$ as well as the Boolean variable `concretized` indicating whether at least one state was concretized.

Methods

`FindAbstractState(DTMC` $\mathscr{D}$`, Abstraction` $Sub_{\mathscr{D}}$`)` chooses heuristically an abstract state s_a of $\mathscr{D}$ which is one of the initial states of an abstract DTMC. Details on suitable heuristics are discussed later.

`Restrict(`$\mathscr{D}, E_{\mathscr{D}} \setminus E_{\mathscr{D}_{abs}}$`)` restricts the DTMC with respect to a set of transitions, see Definition 54 on Page 85.

Procedure

First, `concretized` is set to `false` (Line 1). Two DTMCs $\mathscr{D}_{min}$ and $\mathscr{D}_{max}$ are created, the former one is empty, the latter one is the input DTMC (Lines 2-3). The while-loop runs until `FindAbstractState()` returns no abstract state, i. e., $s_a = \bot$ (Line 6). If this is the case, the resulting DTMCs $\mathscr{D}_{max}$ and $\mathscr{D}_{min}$ are returned (Line 7). If no abstract state was chosen during the procedure, `concretized` has never been assigned `true`. In case an abstract state $s_a \in S$ was chosen, `concretized` is assigned `true` (Line 1). Then, the abstraction pair $(\mathscr{D}_{abs}, \mathscr{D}_{conc})$ is selected from $Sub_{\mathscr{D}}$ such that s_a is one of their input states (Line 10). The result is saved in $\mathscr{D}_{min}$ (Line 11). Finally, the abstraction $\mathscr{D}_{abs}$ is replaced by its concretization $\mathscr{D}_{conc}$ (Line 12). This is saved in $\mathscr{D}_{max}$.

Lemma 1 (Correctness of `Concretize`) *Assume a DTMC $\mathscr{D} = (S, I, P, L)$, a set $Sub_{\mathscr{D}}$ according to Definition 45 on Page 64 and a set of absorbing target states $T \subseteq S$ in $\mathscr{D}$. Let ($\mathtt{true}$,$\mathscr{D}_{min}$,$\mathscr{D}_{max}$) be the result of the method* `Concretize(`$\mathscr{D}$`,`$Sub_{\mathscr{D}}$`)`*. It holds for all $s_I \in Init_{\mathscr{D}}$ and for all $t \in T$ that*

$$Pr^{\mathscr{D}}(Paths_{fin}^{\mathscr{D}}(s_I, t)) = \mathbf{P}^{\mathscr{D}_{max}}(Paths_{fin}^{\mathscr{D}_{max}}(s_I, t)).$$

The correctness of this lemma follows straightforward from the assumption that all pairs $(\mathscr{D}_{abs}, \mathscr{D}_{conc}) \in Sub$ have the same reachability probabilities, see Theorem 3 on Page 61 and Corollary 1 on Page 63.

Example 14 *To explain the procedure of concretizing states, consider the DTMC $\mathscr{D}$ depicted in Figure 5.2(a), where states s_1,s_2 and s_6 are abstract while s_0 and s_5 have already been concretized. For the abstraction procedure, see Example 10 on Page 66.*

Assume, s_6 is chosen to be concretized. The resulting DTMCs $\mathscr{D}_{min}$ and $\mathscr{D}_{max}$ are depicted in Figures 5.2(b) and 5.2(c). All abstract transitions leaving s_6 are removed in $\mathscr{D}_{min}$ yielding a "hole" of transitions indicated by the grey rectangle. In $\mathscr{D}_{max}$ the original transitions of s_6 and s_7 replace the abstract ones. Now, $\mathscr{D}_{min}$ can be locally extended inside this hole by transitions of $\mathscr{D}_{max}$.

5.2.1.1 How to choose abstract states

In our tool `COMICS` [6] the user has the possibility to choose the states that are to be concretized via the graphical user interface. Besides, we offer several heuristics that govern how these states are chosen and how many states are chosen. For instance, all available abstract states are ordered with respect to the combined probability of their adjacent edges in the current subsystem. The states are then chosen in descending order. In our benchmarks, it performed best to concretize $\sqrt{n}$ out of n available abstract states.

5.2.2 The hierarchical algorithm

We have now collected the needed formalisms for computing hierarchical counterexamples. The method we present here uses as blackbox a method to compute critical subsystems as introduced in Section 5.1. Various variants of this method are introduced in the remainder of this thesis. To recall the general idea, consider Figure 5.3. First, SCC-based model checking is applied to decide whether in the DTMC $\mathscr{D}$ the probability to reach target states from T exceeds the bound $\lambda \in \mathbb{Q}$ or not. Moreover, $\mathscr{D}$ is abstracted. If the probability is smaller than or equal to λ, the property is not violated and no counterexample is computed. If—on the other hand—the probability is greater than λ, a first critical subsystem on the abstract DTMC is computed. If no states shall be concretized, this subsystem is returned. If states are to be concretized, a new critical subsystem

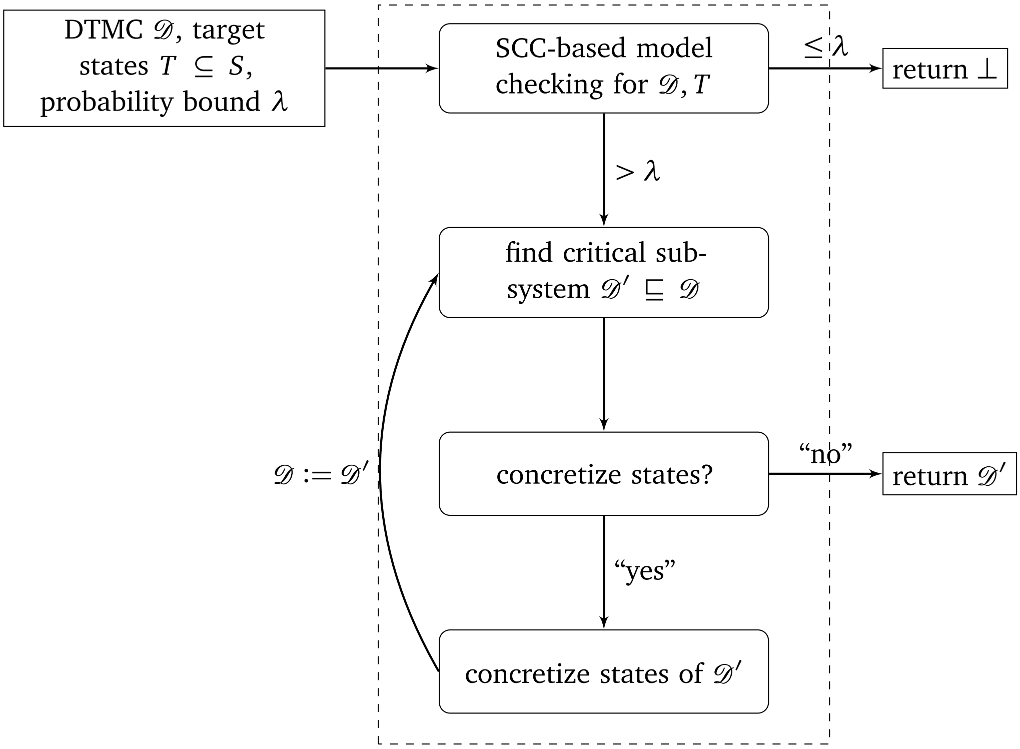

DTMC $\mathscr{D}$, target states $T \subseteq S$, probability bound λ
SCC-based model checking for $\mathscr{D}, T$
$\leq \lambda$
return $\perp$
$> \lambda$
find critical subsystem $\mathscr{D}' \sqsubseteq \mathscr{D}$
$\mathscr{D} := \mathscr{D}'$
concretize states?
"no"
return $\mathscr{D}'$
"yes"
concretize states of $\mathscr{D}'$

Figure 5.3: Hierarchical counterexample generation

is computed on the concretized DTMC. States are concretized, if the intermediate result is too coarse. Then, parts of the subsystem are refined by concretizing. This is done until no further refinement is possible, i. e., the system is concrete, or some heuristics or the user decide that no further refinement is necessary. The method is formalized in Algorithm 4.

Algorithm 4

```
**SearchAbstractCex**(DTMC $\mathcal{D}_{conc} = (S_{conc}, I_{conc}, P_{conc}, L_{conc})$, $T \subseteq S_{conc}$, $\lambda \in [0,1] \subseteq \mathbb{Q}$)
begin
    result $\in [0,1] \subseteq \mathbb{Q}$      DTMC $\mathcal{D}, \mathcal{D}_{min}, \mathcal{D}_{max}$      Abstractions $Sub_{\mathcal{D}}$
    Boolean concretized:=false
    (result, $\mathcal{D}, Sub_{\mathcal{D}}$) := SCCModelCheck($\mathcal{D}_{conc}, T$)
    if result$\leq \lambda$ then
        return $\perp$
    else
        $\mathcal{D}$ := FindCriticalSubsystem($\mathcal{D}_{min}, \mathcal{D}, T, \lambda$)
        while true do
            (concretized, $\mathcal{D}_{min}, \mathcal{D}_{max}$) := Concretize($\mathcal{D}, Sub_{\mathcal{D}}$)
            if (concretized = false) then
                return $\mathcal{D}_{max}$
            else
                $\mathcal{D}$ := FindCriticalSubsystem($\mathcal{D}_{min}, \mathcal{D}_{max}, T, \lambda$)
            end if
        end while
    end if
end
```

Parameters

$\mathcal{D}_{conc}$ is the original concrete input DTMC where all target states are absorbing.

$T \subseteq S$ is the set of absorbing target states.

$\lambda \in [0,1] \subseteq \mathbb{Q}$ is the probability bound that shall be met.

Variables

result $\subseteq [0,1] \subseteq \mathbb{Q}$ is a rational number that will be assigned the model checking result.

All other variables are already explained for the SCC-based model checking (Algorithm 2) on Page 64 and the concretization method (Algorithm 3) on Page 87.

Return value

If the reachability property is violated, the algorithm returns the possibly abstract critical subsystem $\mathcal{D}_{max}$.

Methods

`SCCModelCheck(`DTMC $\mathscr{D}_{conc}$, States T) calls SCC-based model checking (Algorithm 2, Page 64), for the DTMC $\mathscr{D}_{conc}$ and the set of target states T.

`Concretize(DTMC` $\mathscr{D}$, `Abstraction` $Sub_{\mathscr{D}}$) calls the concretization method (Algorithm 3, Page 87) for the DTMC $\mathscr{D}$ and the abstraction pairs $Sub_{\mathscr{D}}$.

`FindCriticalSubsystem(DTMC` $\mathscr{D}_{min}$, `DTMC` $\mathscr{D}_{max}$, `States` T, $\lambda \in [0,1] \subseteq \mathbb{Q}$) returns a critical subsystem of $\mathscr{D}_{max}$ for target states T and probability bound λ. For some implementations $\mathscr{D}_{min}$ is used to build this subsystem.

Procedure

First, SCC-based model checking returns the model checking result, the abstract DTMC $\mathscr{D}$ and the set of abstraction pairs $Sub_{\mathscr{D}}$ (Line 3). If the model checking result is positive (Line 4), there is no counterexample and the algorithm terminates without result (Line 5). Initially, a critical subsystem is computed for the abstract DTMC $\mathscr{D}$ 7, where $\mathscr{D}_{min}$ is empty.

The main loop runs until either the user does not wish to further concretize the system or the critical subsystem stored in $\mathscr{D}_{max}$ is fully concretized (Line 11). Inside the loop first `Concretize` is called to refine the current abstract system by concretizing zero, one or more abstract states (Line 9). A triple is returned, consisting of the flag `concretized` and the DTMCs $\mathscr{D}_{min}$ and $\mathscr{D}_{max}$. If no state was concretized, the search stops and the current critical subsystem is returned (Line 11). If this happens in the first run of the loop, no state at all was concretized. In this case, the abstract system consisting only of transitions from initial states to target states is the result. Otherwise, a search for a critical subsystem is performed (Line 13).

As the motivation for the hierarchical counterexample generation is the possibility to have abstract counterexamples where a user can examine small system parts, it seems most suited to rely on actual user input. For instance, a user might decide that he doesn't more details and therefore concretization. To embed this in a fully automatic procedure, a user can also fix the maximal number of states that are visible beforehand. Then, the method terminates the search if this number is reached. These aspects are considered in our tool `COMICS`. The tool has a graphical user interface where the current abstraction is displayed and the user can guide the whole search process. If the GUI is not used, he can set different parameters to influence the grade of concretization. Details are given later when the tool is presented.

5.3 Framework for explicit graph representations

We present the general framework to incrementally determine critical subsystems for explicit representations. For an overview, consider Figure 5.4. In every iteration a set of states or transitions of the original system is selected by heuristics. The current subsystem is extended

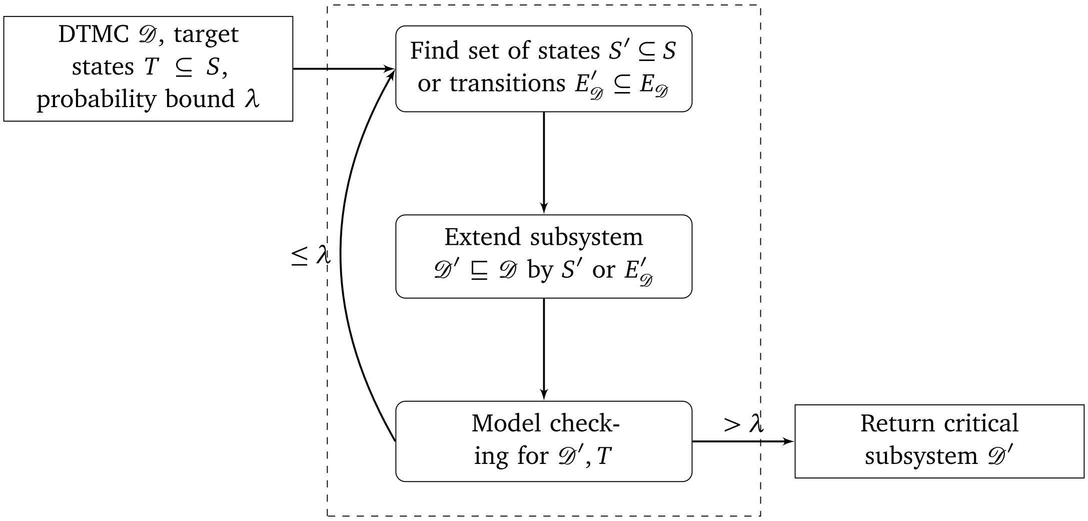

DTMC $\mathscr{D}$, target states $T \subseteq S$, probability bound λ
Find set of states $S' \subseteq S$ or transitions $E'_{\mathscr{D}} \subseteq E_{\mathscr{D}}$
Extend subsystem $\mathscr{D}' \sqsubseteq \mathscr{D}$ by S' or $E'_{\mathscr{D}}$
$\leq \lambda$
Model checking for $\mathscr{D}', T$
$> \lambda$
Return critical subsystem $\mathscr{D}'$

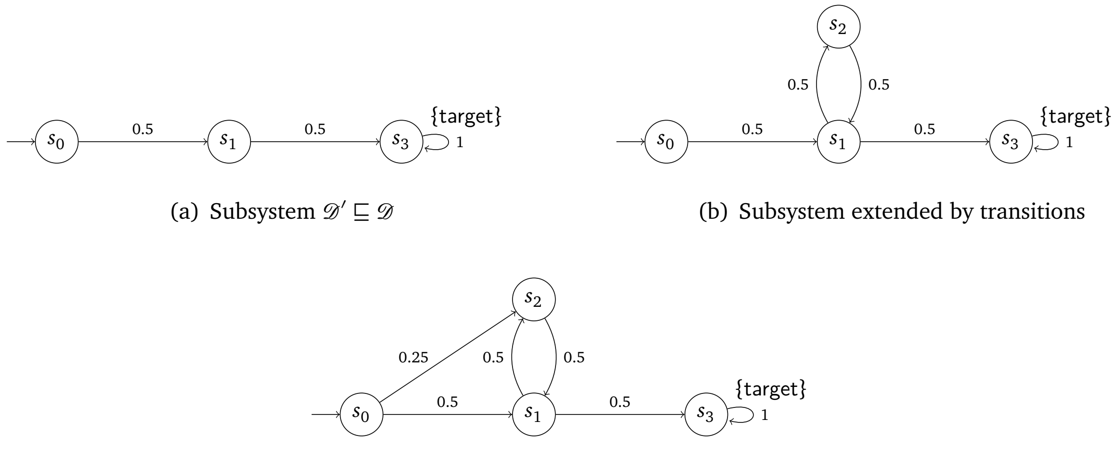
Figure 5.4: Incremental generation of critical subsystems

via these sets until a certain probability is reached. Formally, we use Definition 52 on Page 84 and Definition 53 on Page 84. In general, the extension by states or transitions differs in the following way: If a set of states is added to the subsystem and thereby adding all transitions between old and new states, more probability mass might be “added” in one step, while the extension by transitions might induce finer results with respect to the probability bound. The following example illustrates this.

Example 15 (Extension by states vs. extension by transitions) *Assume a subsystem $\mathscr{D}'$ of the DTMC $\mathscr{D}$ from Example 1 on Page 20 in Figure 5.5(a). We can now extend $\mathscr{D}'$ by a set of transitions $E_e \subseteq E_{\mathscr{D}}$, say $E_e = \{(s_1,s_2),(s_2,s_1)\}$. The result is depicted in Figure 5.5(b). Contrary, if we just extend $\mathscr{D}'$ by one state, say $S_e = \{s_2\}$, the result also contains all induced transitions of s_0,s_1,s_3 and the new state s_2. The result is depicted in Figure 5.5(c).*

Remark 22 *The extension by states was not used in [10]. It showed to be way more efficient, as many transitions are induced by the states, including self-loops. The number of iterations was significantly reduced.*

For the sake of completeness, we parametrize the algorithm by the mode of extension. Let the variable type Mode have the values “`trans-extension`” or “`state-extension`”, i. e., Mode $\texttt{m} \in \{\texttt{trans-extension}, \texttt{state-extension}\}$, depending on whether subsystems shall be extended by states or by transitions. The framework is depicted in Algorithm 5.



(a) Subsystem $\mathscr{D}' \sqsubseteq \mathscr{D}$

(b) Subsystem extended by transitions

(c) Subsystem extended by states

Figure 5.5: The extension of DTMC $\mathscr{D}' \sqsubseteq \mathscr{D}$ by states and transitions

Algorithm 5 Incremental generation of critical subsystems

```
FindCriticalSubsystem(DTMC $\mathscr{D} = (S, I, P, L)$, $T \subseteq S$, $\lambda \in [0,1] \subseteq \mathbb{Q}$, Mode m)
begin
    DTMC subSys := ∅        States newStates := ∅        Transitions newTrans := ∅
    if ModelCheck($\mathscr{D}$, $T$) > $\lambda$ then
        while ModelCheck(subSys, $T$) ≤ $\lambda$ do
            if m = state-extension then
                newStates := FindNewStates($\mathscr{D}$, subSys)
                if newStates ≠ ∅ then
                    subSys := Extend(subSys, newStates)
                end if
            end if
            if m = trans-extension then
                newTrans := FindNewTransitions($\mathscr{D}$, subSys)
                if newTrans ≠ ∅ then
                    subSys := Extend(subSys, newTrans)
                end if
            end if
        end while
    end if
    return subSys
end
```

Parameters

$\mathscr{D}$ is the input DTMC $\mathscr{D} = (S, I, P, L)$ for which a critical subsystem shall be computed.

λ is the upper probability bound for reaching target states. The PCTL property is $\mathbb{P}_{\leq\lambda}(\Diamond T)$.

$T \subseteq S$ is the set of absorbing target states.

`m` is the mode deciding whether the subsystem is to be extended by states or transitions.

Variables

`subSys` represents a DTMC that forms a subsystem of $\mathscr{D}$.

`newStates` $\subseteq S$ is a set of states by which the subsystem is extended.

`newTrans` $\subseteq E_{\mathscr{D}}$ is a set of transitions by which the subsystem is extended.

Return value

If a counterexample exists, the algorithm returns the critical subsystem `subSys`.

Methods

`ModelCheck`(DTMC `subSys`, States T) performs probabilistic model checking for reachability of the target states T for the input system `subSys`, see Section 2.3.1.1. Also the SCC-based model checking, see Chapter 4, can be used. However, we assume only the resulting probability to be returned here.

`FindNewStates(DTMC` $\mathscr{D}$, `DTMC` $\mathscr{D}'$) returns a set of states of the original system $\mathscr{D}$ depending on the current subsystem $\mathscr{D}'$. The method might for instance only return states that are not already contained in $\mathscr{D}'$. Implementations are discussed in the following sections.

`FindNewTransitions(DTMC` $\mathscr{D}$, `DTMC` $\mathscr{D}'$) returns a set of transitions of $\mathscr{D}$ depending on the current subsystem $\mathscr{D}'$.

`Extend`(DTMC `subSys`, States `newStates`) extends the current subsystem by the states stored in `newStates`, see Definition 53 on Page 84.

`Extend`(DTMC `subSys`, Transitions `newTrans`) extends the subsystem by transitions from `newTrans`, see Definition 52 on Page 84.

Procedure

The algorithm proceeds as follows. First, the variables are initialized (Lines 1-3). If `ModelCheck`$(\mathscr{D}, T)$ shows that the probability for reaching T exceeds λ, the property is violated and the search for a counterexample is to be started (Line 2). Otherwise, the algorithm just terminates, returning an empty subsystem indicating that no counterexample exists. The condition of the `while`-loop invokes model checking for the current subsystem `subSys` and the target states T(Line 3). Let `subSys` $= \mathscr{D}' = (S', I', P', L')$. The result is the probability of reaching target states in $T \cap S'$ from initial states $Init_{\mathscr{D}} \cap S'$, i. e., those target states and initial states that are contained in the subsystem $\mathscr{D}'$. If either no target or no initial state is contained in the subsystem, the reachability probability will be 0. The loop runs until a value is returned which is greater than λ. In this case, the current subsystem is *critical*. While the subsystem has not enough probability mass, the subsystem is extended. If the mode is to extend by states (Line 4), method `FindNewStates` returns a set of states `newStates` (Line 4). If `newStates` is not empty, the current subsystem is extended by states (Line 7). For the extension by transitions, the procedure is analogous (Lines 10-12).

Remark 23 *In our implementations, the method* `FindNewStates` *is based on path searching algorithms. The subsystem is then extended by all states or transitions on a path. Nevertheless, this method might also choose a number of states by an arbitrary heuristic. It might also be the case, that no new states are returned, i. e., the subsystem is not extended. We therefore need to make sure that the implementations of* `FindNewStates` *eventually explore the whole input system in order to guarantee termination.*

5.3.1 Heuristics for model checking

Calling a model checking algorithm in each iteration is quite costly. Depending on the input system, in our implementation we extend the subsystem several times until we invoke model checking. One might think of many different heuristics for the number of iterations until model checking is performed. We tried the following, where $\lambda \in [0, 1] \subseteq \mathbb{Q}$ is the probability bound, $prob_{cur}$ is the current probability of the subsystem, $prob_{old}$ is the probability of the subsystem when we last performed model checking and n is the number of iterations we suspend model checking.

- Apply model checking every n iterations until $prob = 0.9 \cdot \lambda$, then in every step. Thereby, n is chosen *statically*, e. g., with respect to the size of the input system.

- Compute the number of iterations n' to suspend model checking *dynamically* by

$$n' = \frac{\lambda - prob_{cur}}{prob_{cur} - prob_{old}} \cdot n$$

Note that in our algorithms, one iteration corresponds to one path search.

5.4 Framework for symbolic graph representations

We now present our framework for the generation of counterexamples for probabilistic reachability properties suited to use *symbolic data structures*. For the symbolic representation of a DTMC as input, a critical subsystem is computed which is again symbolically represented. A method returning states of the input DTMC is again seen as blackbox here. Implementations of this method are presented later. As for explicitly represented DTMCs, see Section 5.3, the critical subsystem is initially empty and will incrementally be extended by states that occur on found paths.

For simplicity, we restrict input DTMCs $\mathscr{D} = (S, I, P, L)$ to having one unique initial state $s_I \in S$. This poses no restriction, see Remark 5 on Page 25. The algorithm for finding a symbolic counterexample is depicted in Algorithm 6. Recall the symbolic representation of DTMCs, see Section 2.4.2.

Algorithm 6 Incremental generation of symbolic critical subsystems

```
FindCriticalSubsystem(MTBDD $\widehat{P}$, BDD $\widehat{I}$, BDD $\widehat{T}$, $\lambda \in [0,1] \subseteq \mathbb{Q}$)
begin
    BDD subSysStates := $\emptyset$        BDD newStates := $\emptyset$        MTBDD subSys := $\emptyset$
    if SymbModelCheck($\widehat{P}$,$\widehat{I}$,$\widehat{T}$) $> \lambda$ then
        while SymbModelCheck(subSys,$\widehat{I}$,$\widehat{T}$) $\leq \lambda$ do
            newStates := FindNewStates($\widehat{P}$,$\widehat{I}$,$\widehat{T}$,subSys)
            if newStates $\neq 0$ then
                subSysStates := subSysStates $\cup$ newStates
                subSys := ToTransitionBDD(subSysStates) $\cdot \widehat{P}$
            end if
        end while
    end if
    return subSys
end
```

Parameters

$\widehat{P}$ is the symbolic representation of the transition probability matrix of the input DTMC defined over variable sets *Var* and *Var′*.

$\widehat{I}$ is the BDD representing the initial state of the input DTMC defined over variables *Var*.

$\widehat{T}$ is the BDD representing the set of target states defined over variables *Var*.

λ is the probability bound out of $[0,1] \subseteq \mathbb{Q}$ which should be exceeded by the resulting critical subsystem.

Variables

`subSysStates` is a BDD used to symbolically represent the set of states which are part of the current subsystem.

`newStates` is a BDD used to store the states occurring on a path or on a set of paths which extend the current subsystem.

`subSys` stores the transition MTBDD of the current subsystem.

Return value

If a counterexample exists, the algorithm returns an transition MTBDD representing a critical subsystem.

Methods

`SymbModelCheck(MTBDD` $\widehat{P}$`, BDD` $\widehat{I}$`, BDD` $\widehat{T}$`)` performs probabilistic model checking for symbolically represented DTMCs [BCH+97, Par02], see Section 2.4.2, and returns the probability of reaching target states in $\widehat{T}$ from initial states in $\widehat{I}$ via transitions in $\widehat{P}$.

`FindNewStates(MTBDD` $\widehat{P}$`, BDD` $\widehat{I}$`, BDD` $\widehat{T}$`, MTBDD subSys)` invokes a method computing a set of states which occur on a path or a set of paths leading through the DTMC represented by the transition MTBDD $\widehat{P}$, the initial state $\widehat{I}$, and the set of target states $\widehat{T}$. Which paths are found next may depend on the current subsystem `subSys` and therefore on the set of previously found paths. Different symbolic implementations of this method are discussed in the Sections 5.6 and 5.7.

`ToTransitionBDD(BDD subSysStates)` first computes a BDD `subSysStates`$'$ by renaming each variable $x \in \mathit{Var}$ occurring in `subSysStates` to $x' \in \mathit{Var}'$ and returns the transition BDD `subSys` in which there is a transition between all pairs of states occurring in `subSysStates`, i. e., $(\texttt{subSys})(\nu_{s_1,s_2}) = 1$ iff $\texttt{subSysStates}(\nu_{s_1}) = \texttt{subSysStates}(\nu_{s_2}) = 1$. Intuitively, this yields a BDD inducing the complete directed graph over `subSysStates`, i. e., all states of the subsystem are connected to each other. Multiplying this BDD with the transition probability matrix $\widehat{P}$ removes all transitions from $\widehat{P}$ which do not connect two states of the subsystem. Recall Section 2.4.1 for details on BDDs.

Procedure

The algorithm proceeds similar to Algorithm 5 for explicit graph representations on Page 94. Nevertheless, we shortly explain the crucial steps. First, the three empty objects `subSysStates`, `newStates`, and `subSys` are created (Line 3). If `SymbModelCheck(`$\widehat{P},\widehat{I},\widehat{T}$`)` shows that λ is exceeded, the reachability property is violated and the search for a counterexample starts (Line 2). Otherwise, the algorithm just terminates, returning an empty subsystem since no counterexample exists. The condition of the `while`-loop invokes model checking for the current subsystem `subSys` and the initial states and target states (Line 3). The loop runs until `SymbModelCheck(subSys,`$\widehat{I},\widehat{T}$`)` returns a value which is greater than λ, in which case the current subsystem is *critical*. In every iteration, first the method `FindNewStates(`$\widehat{P},\widehat{I},\widehat{T}$`,subSys)` returns a set of states which occur on a path or a set of paths through the system (Line 4). If this set is not empty, the current set of states is extended by these new states (Line 6). Afterwards, the current subsystem is extended (Line 7) in the following way: `ToTransitionBDD(subSysStates)` generates a transition relation between all pairs of found states. Multiplying the resulting BDD and the original transition MTBDD $\widehat{P}$ induces a probability matrix $P' \subseteq P$ restricted to transitions between the states in `subSysStates`. These transitions define the updated subsystem `subSys`.

Remark 24 *The extension as performed in this framework is a symbolic implementation of the extension of a subsystem by states, see Definition 53 on Page 84.*

5.5 Explicit path searching algorithms

We now present two search concepts, namely the *global search* and the *fragment search*. The first one lists paths leading from initial states to target states of the system in descending order of their probability, as proposed in [HKD09]. The second one was developed in [10] and dynamically finds most probable path fragments that extend the current subsystem.

We give the general concepts in detail and explain how they are implemented for explicit graph representations. We use well-known graph algorithms which are not explained in detail here.

5.5.1 Explicit global search

For finding paths in descending order of their probability we follow the way proposed in [HKD09], see Section 3.1. Here, we call this approach fitted to the generation of critical subsystems a *global search*.

By transforming a DTMC into a weighted directed graph where the weights are the negative logarithm of the corresponding probabilities, the k *shortest paths* are the associated k *most probable* paths. As we cannot fix beforehand how many paths are needed for a counterexample, the k is determined according to an external condition on the fly. As already explained in Section 3.1.1, we therefore use the k shortest paths algorithm by Jiminez and Marzal [JM99].

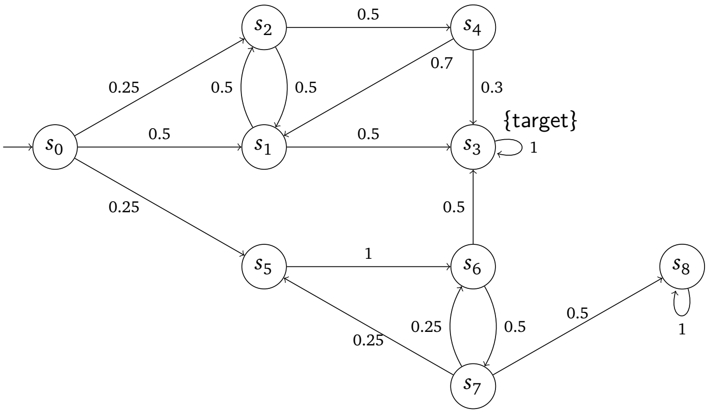
After a path has been found, the framework for explicit counterexample generation, see Algorithm 5 on Page 94, extends the current subsystem by the states or transitions on this path. This procedure is repeated until the subsystem has enough probability mass to be critical. If this is the case, the k shortest path algorithm terminates.

To describe the next path to be found formally, let $\mathcal{D} = (S, I, P, L)$ be a DTMC and $T \subseteq S$ a set of target states. Let $F \subseteq \mathit{Paths}^{\mathcal{D}}_{\mathit{fin}}(\mathit{Init}_{\mathcal{D}}, T)$ be the set of paths that were already found. The candidates for the next global path are therefore $C_{glob} = \mathit{Paths}^{\mathcal{D}}_{\mathit{fin}}(\mathit{Init}_{\mathcal{D}}, T) \setminus F$. The *next path* π_{next} is then given by

$$\pi_{next} := \arg\max_{\pi \in C_{glob}} \mathbf{P}^{\mathcal{D}}(\pi).$$

Note that this path is not unique.

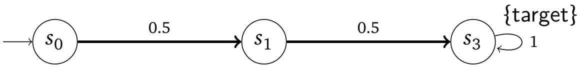
Example 16 *Consider again the DTMC $\mathcal{D}$ from Example 1 on Page 20:*



The model checking probability of reaching the target state s_3 is $Pr^{\mathcal{D}}_{s_0}(\lozenge \mathsf{target}) = 0.9175$. We are interested in a counterexample for the violated *property $\mathbb{P}_{\leq 0.7}(\lozenge \mathsf{target})$. Subsystems will always be extended by states. The most probable path for this system is*

$$\pi_1 = s_0, s_1, s_3, \quad \mathbf{P}^{\mathcal{D}}(\pi_1) = 0.25$$

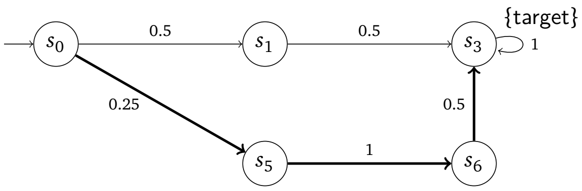
Thus, we will extend the currently empty subsystem $\mathcal{D}'$ by the states of π_1. We will always highlight the latest paths by thick edges in the subsystem. The reachability probability in this first subsystem is $Pr^{\mathcal{D}'}_{s_0}(\lozenge \mathsf{target}) = 0.25 < 0.7$.



Now, the set of already found paths is $F = \{(s_0, s_1, s_3)\}$. The candidate set for the next most probable path is therefore $C_{glob} = \mathit{Paths}^{\mathcal{D}}_{\mathit{fin}}(s_0, s_3) \setminus F$. We have:

$$\pi_2 = s_0, s_5, s_6, s_3, \quad \mathbf{P}^{\mathcal{D}}(\pi_2) = 0.125$$

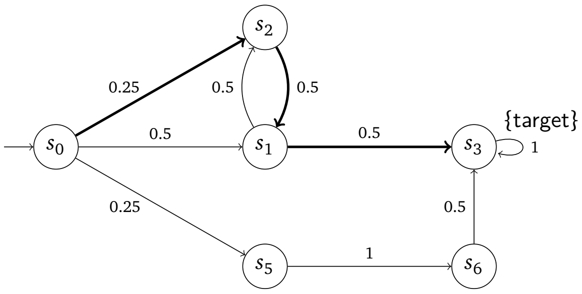
The subsystem is then extended by these states. The probability is now $Pr^{\mathcal{D}'}_{s_0}(\lozenge \mathsf{target}) = 0.375$ which is still not high enough.



The next path is

$$\pi_3 = s_0, s_2, s_1, s_3, \quad \mathbf{P}^{\mathscr{D}}(\pi_3) = 0.0625$$

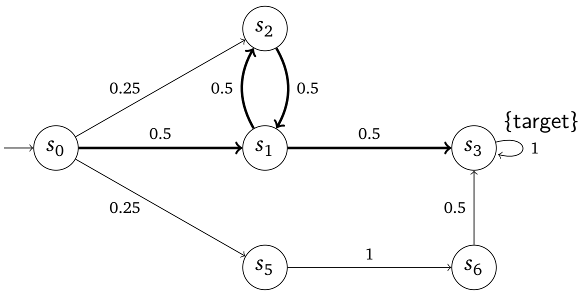
This effectively extends the subsystem by state s_2 as the other states are already included. The model checking result is now $Pr^{\mathscr{D}'}_{s_0}(\lozenge\text{target}) = 0.542$, so another search iteration is started.



The next path is the first one containing a loop and has the same probability as π_3:

$$\pi_4 = s_0, s_1, s_2, s_1, s_3, \quad \mathbf{P}^{\mathscr{D}}(\pi_4) = 0.0625$$

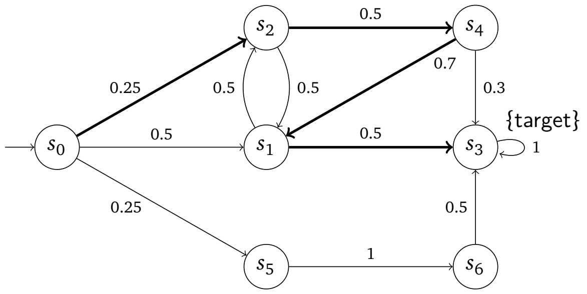
As there are no new states on the path, the subsystem is not extended and the model checking result remains 0.524*:*



The next path is:

$$\pi_5 = s_0, s_2, s_4, s_1, s_3, \quad \mathbf{P}^{\mathscr{D}}(\pi_5) = 0.04375$$

Although this path has a relatively small probability, state s_4 is added which induces three transitions yielding the probability mass $Pr^{\mathscr{D}'}_{s_0}(\lozenge\text{target}) = 0.875$ for the subsystem. Therefore, the subsystem is critical and the search is terminated with the following final subsystem representing a counterexample:



The problem of the global search approach is illustrated by Example 16. Paths containing loops can be found. For large systems, where most of the paths have very small probability, many paths are found that only differ in the iteration of loops. As these paths do not extend the subsystems, this is unwanted behavior which cannot easily be prevented.

Adaption for hierarchical counterexample generation The global search can directly be used for generating hierarchical counterexamples. Consider the DTMCs $\mathscr{D}_{max}$ and $\mathscr{D}_{min}$ as described in Section 5.2. Paths from $\mathscr{D}_{max}$ according to the global search are incrementally added to $\mathscr{D}_{min}$ by the explicit framework, see Algorithm 5 on Page 94. Note that the "holes" in $\mathscr{D}_{min}$ resulting from concretization steps are not particularly treated here.

5.5.2 Explicit fragment search

In contrast to the global search approach, the fragment search does not aim at finding paths from initial to target states but to identify connected fragments of the search graph. This is achieved by searching for the most probable path that starts and ends in states of the current subsystem. In the context of the whole graph, these paths are called *path fragments*.

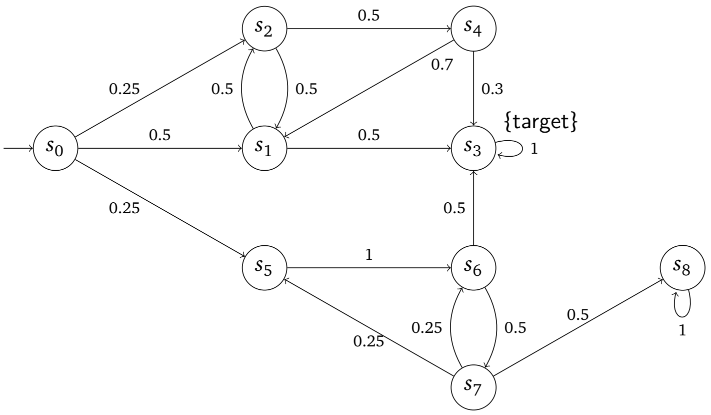
Let $\mathscr{D}' = (S', I', P', L')$ with $\mathscr{D}' \sqsubseteq \mathscr{D}$ be the current subsystem for $\mathscr{D}$ and the set of absorbing target states T. The candidate set for the next path fragment is $C_{frag} = \{\pi \in Paths^{\mathscr{D}}_{fin}(S' \setminus T, S') \setminus Paths^{\mathscr{D}'}_{fin}\}$, i.e., the finite paths of the original DTMC $\mathscr{D}$ that both start and end in a state of $\mathscr{D}'$ while the starting states are not target states. The next path π_{next} is then given by

$$\pi_{next} := \arg\max_{\pi \in C_{frag}} \mathbf{P}^{\mathscr{D}}(\pi)$$

Initially, we set $S' := Init_{\mathscr{D}} \cup T$. Therefore, the first path is a most probable path leading from an initial state $s_I \in Init_{\mathscr{D}}$ to a target state $t \in T$.

Remark 25 *Algorithmically, for every state of $\mathscr{D}'$ Dijkstra's algorithm has to be called which is of course very expensive. In our implementation we use a simple trick: We introduce a dummy state which has transitions of equal probability leading to each state of $\mathscr{D}'$. Then, the most probable path is determined starting from this dummy state which speeds up the computation by orders of magnitude.*

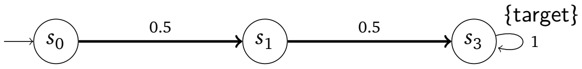
Example 17 *We give an example similar to Example 16 on Page 100 for the global search approach by means of DTMC $\mathscr{D}$ from Example 1 on Page 20:*



The model checking probability of reaching the target state s_3 is $Pr^{\mathscr{D}}_{s_0}(\lozenge\text{target}) = 0.9175$. We are again interested in a counterexample for the violated *property $\mathbb{P}_{\leq 0.7}(\lozenge s_3)$. Subsystems are extended by states. As the initial subsystem is empty except for initial and target states, the first path is the same as for the global search:*

$$\pi'_1 = s_0, s_1, s_3, \quad \mathbf{P}^{\mathscr{D}}(\pi'_1) = 0.25$$

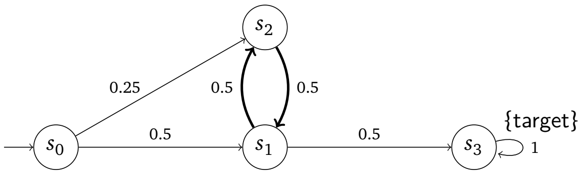
The reachability probability in the first subsystem $\mathscr{D}'$ is $Pr^{\mathscr{D}'}_{s_0}(\lozenge\text{target}) = 0.25 < 0.7$.



Now, we search for path fragments starting and ending in states s_0, s_1 or s_3. The most probable path fragment is the loop:

$$\pi'_2 = s_1, s_2, s_1, \quad \mathbf{P}^{\mathscr{D}}(\pi'_2) = 0.25$$

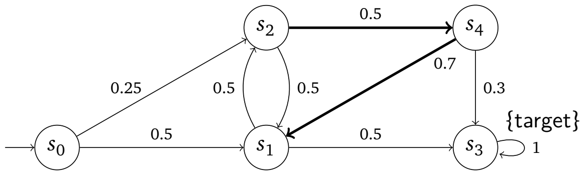
The resulting subsystem $\mathscr{D}'$ has a reachability probability of $Pr^{\mathscr{D}'}_{s_0}(\lozenge\text{target}) = 0.542$ as three transitions are induced:



As the probability is not yet high enough, the search for the next fragment is started, now search *for fragments connecting states s_0, s_1, s_2, s_3. The most probable fragment is:*

$$\pi'_3 = s_2, s_4, s_1, \quad \mathbf{P}^{\mathscr{D}}(\pi'_3) = 0.35$$

By effectively adding state s_4 to the system, three transitions are induced and the resulting subsystem has already a reachability probability of $Pr^{\mathscr{D}'}_{s_0}(\lozenge\text{target}) = 0.75$ which renders the subsystem critical and terminates the search:



Example 17 illustrates the advantage of the fragment search approach: In every iteration at least one new state is added to the system. Therefore, the number of search iteration is bounded by the number of states of the input system. However, the search itself is quite costly as mentioned before.

Adaption for hierarchical counterexample generation Originally, the fragment search was explicitly developed for the concept of hierarchical counterexample generation. By starting with the DTMC $\mathscr{D}_{min}$ as initial subsystem, the "holes" which are formally exactly the transitions that were removed by the last concretization step, are then extended by transitions from $\mathscr{D}_{max}$. For details, see again Section 5.2. As thereby transitions are incrementally added via these local parts of the original DTMC, we call the fragment search in this context the *local search*.

5.6 Path searching by bounded model checking

As described in Chapter 3, Wimmer et. al. adapted bounded model checking for digital circuits to generate counterexamples for DTMCs and reachability properties in [WBB09]. A SAT solver is used to enumerate arbitrary paths starting in an initial state and ending in a certain target state. This search is *uninformed* regarding the probability of the paths that were found. Every path is processed to determine its probability and to detect loops, which can be connected to other paths that visit one of their states and contribute to the probability mass. The drawbacks of this approach are on the one hand the uninformed search which may lead to a very high number of paths that need to be found while only a few paths of high probability would suffice. On the other hand, paths are enumerated explicitly which equalizes the advantage of symbolic methods, i. e., the handling of very large systems.

The input for all search algorithms is a symbolic representation of a DTMC $\mathscr{D} = (S, I, P, L)$ together with a set of target states T, as described in Section 2.4.2: An BDD $\widehat{I}$ for the initial state $s_I \in S$, a BDD $\widehat{T}$ for the set of target states T, and an MTBDD $\widehat{P}$ over variable set $Var =$

$\{\sigma_1, \ldots, \sigma_m\}$ and $\mathit{Var}' = \{\sigma'_1, \ldots, \sigma'_m\}$ for the transition probability matrix P. As in bounded model checking the probabilities are no considered, we transform $\widehat{P}$ into the BDD describing the induced non-probabilistic transition relation $\widehat{P}_{\text{bool}}$, without any transitions leaving target states from T.

Having this symbolic representation, first, Tseitin's transformation [Tse83] is applied to generate formulae in conjunctive normal form (CNF) for the BDDs $\widehat{I}$, $\widehat{T}$ and $\widehat{P}_{\text{bool}}$. We denote the resulting CNF predicates by $\check{I}(\mathit{Var})$, $\check{T}(\mathit{Var})$ and $\check{P}(\mathit{Var}, \mathit{Var}')$, respectively.

5.6.1 Adaption of global search

We utilize the path search as in [WBB09] for our first implementation of the `FindNewStates`-method of the symbolic framework, see Algorithm 6 in Section 5.4. Thereby, a critical subsystem is generated incrementally by the states that occur on paths that are computed by the SAT solver. This is a global search approach as already described for explicit graph representations in Section 5.5.1, while not the most probable paths are found but arbitrary paths leading from initial states to target states.

The BMC formula is parametrized in $k \in \mathbb{N}$ and uses the CNF predicates $\check{I}(\mathit{Var})$, $\check{T}(\mathit{Var})$ and $\check{P}(\mathit{Var}, \mathit{Var}')$:

$$\mathit{BMC}(k) \;=\; \check{I}(\mathit{Var}_0) \,\wedge\, \bigwedge_{i=0}^{k-1} \check{P}(\mathit{Var}_i, \mathit{Var}_{i+1}) \,\wedge\, \check{T}(\mathit{Var}_k) \tag{5.1}$$

The set of solutions for $\mathit{BMC}(k)$ corresponds to the set of paths of length k from the initial state to a target state, where for each $i = 0, \ldots, k$ the set $\mathit{Var}_i = \{\sigma_{i,1}, \ldots, \sigma_{i,m}\}$ of Boolean variables is used to encode the state at depth i on a path. We overload $\mathit{BMC}(k) \subseteq \mathit{Paths}_k^{\mathscr{D}}$ to describe this set of paths.

Let $\nu\colon \bigcup_{i=0}^{k} \mathit{Var}_i \to \{0,1\}$ be a satisfying assignment of $\mathit{BMC}(k)$. The i^{th} state on the path is then encoded by $\nu_i\colon \mathit{Var}_i \to \{0,1\}$ with $\nu_i(\sigma_j) = \nu(\sigma_{i,j})$ for each $j = 1, \ldots, m$. If there is no satisfying assignment, there is no such path.

Usually multiple paths need to be found in order to form a counterexample or a critical subsystem, thus the solver has to enumerate satisfying solutions for $\mathit{BMC}(k)$, $k = 0, 1, \ldots$, until enough probability mass has been accumulated.

Remark 26 (Stochastically independent paths) *As target states have no outgoing transitions, a path ends at the first target state that is reached. Thus, two different paths from the initial state to a target state are never prefixes of each other, i. e., their corresponding cylinder sets are disjoint and their joint probability is the sum of their individual probabilities.*

5.6.1.1 Exclusion of paths

To assure that a path is not considered several times, each time a solution is found it is excluded from further search by adding new clauses to the SAT solver's clause database. Assume that the solver has found a solution $\nu : \bigcup_{i=0}^{k} Var_i \rightarrow \{0,1\}$ for $BMC(k)$. The found path is uniquely described by the following conjunction:

$$\bigwedge_{i=0}^{k} \bigwedge_{j=1}^{m} \sigma_{i,j}^{\nu(\sigma_{i,j})} \tag{5.2}$$

where $\sigma_{i,j}^1 = \sigma_{i,j}$ and $\sigma_{i,j}^0 = \neg\sigma_{i,j}$. To exclude the found path from the solution space of $BMC(k)$, the negation of the above conjunction according to deMorgan's law is added to the solver's clause database:

$$\bigvee_{i=0}^{k} \bigvee_{j=1}^{m} \sigma_{i,j}{}^{1-\nu(\sigma_{i,j})} \tag{5.3}$$

This ensures that for a new path at least one state variable has to be assigned differently as it is done by ν.

5.6.1.2 Termination

Termination of the iterative construction of a critical subsystem is guaranteed, as the SAT solver ultimately finds *all* finite paths of length k. Eventually, the subsystem will consist of all states that are part of paths from initial to target states. This subsystem induces the whole probability mass of reaching a target state in the original system. As the counterexample generation in Algorithm 6 on Page 97 only starts if the probability bound is exceeded, the probability mass of this system will also exceed the bound. Therefore, the algorithm always terminates.

Example 18 *Assume the symbolic representation of the DTMC $\mathcal{D}$ from Figure 2.1 as explained in Example 6 in Section 2.4. We use the same set of variables $Var = \{\sigma_1, \sigma_2, \sigma_3\}$ while we add another index for the depth of the path at which each variable is used to encode a state. For example, the formula $\sigma_{2,1}^0 \wedge \sigma_{2,2}^1 \wedge \sigma_{2,3}^0$ encodes state s_2 at depth 2 of a path. As the shortest path leading from the initial state s_0 to the target state s_3 has length 2, there will be no satisfying assignments for $BMC(0)$ and $BMC(1)$. For $k = 2$, the formula*

$$\underbrace{\sigma_{0,1}^0 \wedge \sigma_{0,2}^0 \wedge \sigma_{0,3}^0}_{s_0} \quad \wedge \quad \underbrace{\sigma_{1,1}^0 \wedge \sigma_{1,2}^0 \wedge \sigma_{1,3}^1}_{s_1} \quad \wedge \quad \underbrace{\sigma_{2,1}^0 \wedge \sigma_{2,2}^1 \wedge \sigma_{2,3}^1}_{s_3}$$

encodes the first path $\pi_1 = s_0, s_1, s_3$ of the global search explained for explicit graph representations in Section 5.5.1, see Example 16 on Page 100. The predicates $\check{P}$, $\check{I}$, and $\check{T}$ are all satisfied. Adding the negation of this formula to $BMC(2)$ prevents the SAT solver from finding this path again.

5.6.2 Adaption of fragment search

The previously described approach of using a SAT solver to find paths leading from the initial state of the DTMC to the target states is now extended according to the *fragment search* approach as described in Section 5.5.2. We therefore aim at finding *path fragments* that extend the already found system iteratively.

The intuition is as follows: We first define a maximal length of paths $n_{\max} \in \mathbb{N}$ which can be increased if necessary. This $n_{\max}$ might, e. g., correspond to the length of the longest loop-free path leading from one of the initial states to a target state. In this case, ultimately all reachable states will be visited.

Consider in the following a DTMC $\mathcal{D} = (S, s_I, P, L)$ with a single initial state s_I and a *single target state* $t \in S$, i. e., $T = \{t\}$. If the DTMC has originally several initial states, a transformation as in Remark 5 on Page 25 can be applied. A similar construction is possible for target states. In the first search iteration 0, the CNF formula given to the SAT solver is satisfied if and only if the assignment corresponds to a path of arbitrary but bounded length $n_{\max}$ through the input DTMC $\mathcal{D}$ leading from the initial state s_I to the target state t. This path is returned to the symbolic framework, see Algorithm 6 on Page 97, and thereby induces the initial subsystem. Subsequently, this system is extended by paths whose first and last states are included in the current subsystem, while all states in between are fresh states.

To accomplish this by means of an incremental method without completely restarting the solving process in every iteration, we need to consider already found states for all possible depths d of a path $\pi = s_0, \ldots, s_n$, $0 \leq d \leq n$. It needs to be ensured that at least one new state is found in every iteration. Recall that the state at depth d of a path π is the d^{th} state on this path. First, we need to identify states of the original DTMC by the assignment of variables: For a state s_d let $v_s^d \colon \mathit{Var}_d \to \{0,1\}$ be the unique assignment of Var_d corresponding to state s_d.

We introduce a *state flag* f_s^d for each state s and each depth d. This flag is assigned 1 if and only if the assignment of the state variables at depth d corresponds to the state s. Thereby, we can "switch" the occurrence of a state s at level d by setting its flag f_s^d to 0 or 1.

$$f_s^d \longleftrightarrow (\sigma_{d,1}^{v_s^d(\sigma_{d,1})} \wedge \cdots \wedge \sigma_{d,m}^{v_s^d(\sigma_{d,m})}) \tag{5.4}$$

Furthermore, we need to be able to reason about whole sets of states. Using the state flags, we introduce the notion of *state set flags*: K_j^d describes the whole set of states that have been found so far, namely in the iterations $0, \ldots, j$ of the search process (again in terms of the variables Var_d for depth d). Note, that in our setting these are exactly the states of the current subsystem `subSys` after iteration j. We define $K_{-1}^d := \texttt{false}$. Assume that in iteration j of the search process the path $\pi_j = s_0 s_1 \ldots s_n$ is found. We then define

$$K_j^d \longleftrightarrow \left(K_{j-1}^d \vee \bigvee_{i=0}^{n} f_{s_i}^d\right) \tag{5.5}$$

This formula evaluates to `true` if and only if either one of the state flags $f_{s_0}^d, \ldots f_{s_n}^d$ is `true` or if one of the state flags that are implicitly given by the previous state set flag K_{j-1}^d is `true`. K_j^d is thereby `true` if and only if the assignment corresponds to one of the states that were encountered so far including the current path π_j.

We are now ready to introduce the bounded model checking formulae that enable the adaption of the fragment search to bounded model checking.

First search iteration $j = 0$**:** In the first search iteration $j = 0$ we need a formula which is true iff the variable assignment corresponds to a path of arbitrary length—bounded by n_{max}—leading from the initial state to the target state of the DTMC. In more detail, this is done by enforcing that up to length $n_{\max}$ at some depth there will occur the target state on the path and that before this target state occurs, transitions will be taken, starting in one of the initial states.

$$\begin{aligned} BMC^0_{frag} &= \check{I}(Var_0) \wedge \bigvee_{i=0}^{n_{\max}} \check{T}(Var_i) \wedge \\ &\bigwedge_{i=0}^{n_{\max}-1} \Big[\big(\neg\check{T}(Var_i) \rightarrow \check{P}(Var_i, Var_{i+1})\big) \wedge \big(\check{T}(Var_i) \rightarrow (Var_i = Var_{i+1})\big) \Big] \end{aligned} \tag{5.6}$$

Assume an assignment $\nu\colon \bigcup_{i=0}^{k} Var_i \rightarrow \{0,1\}$ that satisfies BMC^0_{frag}. This corresponds to a path $s_0, \dots, s_{n_{max}}$, where the i^{th} state on the path is encoded by $\nu_i\colon Var_i \rightarrow \{0,1\}$ with $\nu_i(\sigma_j) = \nu(\sigma_{i,j})$ for each $j = 1, \dots, m$ and $0 \le i \le n_{\max}$. If there is no satisfying assignment, there is no such path and $n_{\max}$ has to be increased or the search has to be terminated.

BMC^0_{frag} ensures that the first state s_0 is the initial state as variables from Var_0 have to be assigned such that $\check{I}(Var_0)$ is satisfied. One of the states $s_0, \dots s_n$ has to be the target state, as at least for one i the variables from Var_i have to be assigned such that $\check{T}(Var_i)$ is satisfied.

If a state s_i is not the target state, i. e., Var_i is not assigned such that $\check{T}(Var_i)$ is satisfied, a valid transition will be taken to the next state. Therefore, Var_i and Var_{i+1} have to be assigned such that $\check{P}(Var_i, Var_{i+1})$ is satisfied. On the other hand, if s_i is the target state, Var_{i+1} is assigned exactly as Var_i. This implies, that all variables from Var_h with $h > i$ will be assigned the same value as the variables Var_i and thereby satisfy $\check{T}(Var_h)$, which creates an implicit self-loop on the corresponding target state s_i. This is done since otherwise the solver would be free to assign arbitrary values to the states following the target state which would not be desired behavior. The path returned to Algorithm 6 then ends with the first target state that is encountered.

Search iterations $j > 0$**:** For the following iterations we require that each solution corresponds to a path fragment that starts and ends with a state of the current subsystem, i. e., the states found so far. Furthermore, it shall contain at least one new state in between. We make use of the previously defined state set flags K^j_d, for $j > 0$, see Formula 5.5, to describe the states that occurred so far.

$$\begin{aligned} \mathit{BMC}^{j}_{\mathit{frag}} &= K^0_{j-1} \;\wedge\; \check{P}(\mathit{Var}_0, \mathit{Var}_1) \;\wedge \neg K^1_{j-1} \;\wedge\; \bigvee_{d=2}^{n_{\max}} K^d_{j-1} \\ &\quad \wedge \bigwedge_{d=1}^{n_{\max}-1} \Big[\Big(\neg K^d_{j-1} \rightarrow \check{P}(\mathit{Var}_d, \mathit{Var}_{d+1}) \Big) \;\wedge\; \Big(K^d_{j-1} \rightarrow \mathit{Var}_d = \mathit{Var}_{d+1} \Big) \Big] \end{aligned} \tag{5.7}$$

$\mathit{BMC}^{j}_{\mathit{frag}}$ ensures that the first state s_0 of a solution path $\pi_j = s_0, \ldots, s_{n_{\max}}$ is contained in the set of previously found states, as the variables of Var_0 have to be assigned such that K^0_{j-1} is satisfied. From the first state, a transition is taken to a state s_1 that has not been found yet. This is ensured as *Var* has to be assigned such that $\check{P}(\mathit{Var}_0, \mathit{Var}_1)$ is satisfied. One of the states $s_2, \ldots, s_{n_{\max}}$ has to be in the set of states found so far again, so there has to be a d with $2 \leq d \leq n_{\max}$ such that Var_d is assigned such that K^d_{j-1} is `true`.

As in Formula 5.6 for the first iteration, we enforce regular transitions between states that are not target states: If Var_d doesn't satisfy K^d_{j-1}, the transition predicate $\check{P}(\mathit{Var}_d, \mathit{Var}_{d+1})$ has to be true. If Var_d *does* satisfy K^d_{j-1}, all following variable sets are assigned as Var_d enforcing again an implicit self-loop on the corresponding states.

5.6.2.1 Termination

Termination is guaranteed, as the length of the paths is bounded by $n_{\max}$ and in each iteration a new state is found. If no further satisfying assignment exists, either all states of the DTMC $\mathcal{D}$ have been encountered, or $n_{\max}$ has to be increased. However, the diameter, i. e., the longest cycle-free path of the underlying graph, is an upper bound on the length of loop-free paths from initial states to target states. Therefore, $n_{\max}$ needs to be increased only finitely many times, such that a critical subsystem is always determined in finite time.

Example 19 *Consider again the assignment* $\sigma_{0,1} \mapsto 0, \sigma_{0,2} \mapsto 0, \sigma_{0,3} \mapsto 0, \sigma_{1,1} \mapsto 0, \sigma_{1,2} \mapsto 0, \sigma_{1,3} \mapsto 1, \sigma_{2,1} \mapsto 0, \sigma_{2,2} \mapsto 1, \sigma_{2,3} \mapsto 1$ *which encodes the first path* $\pi'_1 = s_0, s_1, s_3$ *for the fragment search as in Example 17 on Page 103. Having this in search iteration* 0*, Formula 5.6 is satisfied, as for the variables from* $\mathit{Var}_0 = \{\sigma_{0,1}, \sigma_{0,2}, \sigma_{0,3}\}$ *the assignment encodes the initial state, and for variables from* $\mathit{Var}_2 = \{\sigma_{2,1}, \sigma_{2,2}, \sigma_{2,3}\}$ *the target state is assigned. For the states encoded by the variables* Var_0 *and* Var_1*, which are not target states, transitions are available leading to the state at the next depth. As the variables from* Var_2 *are assigned to a target state, all the following variable sets* Var_m *with* $2 \leq m \leq n$ *will be assigned equally, thereby again encoding the target state. This causes an implicit self-loop on the target state. According to Formula 5.5, we build the state set variables:*

$$K^0_0 \longleftrightarrow (f^0_{s_0} \vee f^0_{s_1} \vee f^0_{s_3}), \quad K^1_0 \longleftrightarrow (f^1_{s_0} \vee f^1_{s_1} \vee f^1_{s_3}), \quad K^2_0 \longleftrightarrow (f^2_{s_0} \vee f^2_{s_1} \vee f^2_{s_3}) \,.$$

Intuitively, K^d_0 *is true for* $0 \leq d \leq 2$ *iff the variables at depth* d *are assigned to any state out of* s_0*,* s_1

or s_3.

For search iteration 1*, consider the assignment* $\sigma_{0,1} \mapsto 0, \sigma_{0,2} \mapsto 0, \sigma_{0,3} \mapsto 1, \sigma_{1,1} \mapsto 0, \sigma_{1,2} \mapsto 1, \sigma_{1,3} \mapsto 0, \sigma_{2,1} \mapsto 0, \sigma_{2,2} \mapsto 0, \sigma_{2,3} \mapsto 1$*. This encodes the second path $\pi' = s_1, s_2, s_1$ of the fragment search, see again Example 17. Formula 5.7 is satisfied: The variables from Var_0 are assigned such that K_0^0 is true as $f_{s_1}^0$ is true for* $\sigma_{0,1} \mapsto 0, \sigma_{0,2} \mapsto 0, \sigma_{0,3} \mapsto 1$*; a valid transition leads from s_1 to s_2; s_2 satisfies $\neg K_0^1$, and at $d = 2$ again a state satisfying K_0^2 is assigned, namely again s_1. For state s_2—not satisfying K_0^1—a valid transition is taken. Once K_0^d for* $0 < d \leq n$ *is satisfied, all states at the following depths are assigned the same, again creating an implicit self-loop.*

5.6.3 Heuristics for probable paths

As mentioned before, a drawback of the SAT-based search strategies is that paths are found without considering their probability beforehand. If paths or transitions with higher probabilities are preferred, the process can be accelerated.

We therefore developed a heuristic which guides the SAT solver to choose assignments that induce more probable path fragments. SAT solvers have efficient variable selection strategies, i. e., strategies to decide which variable should be assigned next during the solving process. We modify the choice of the *value* the solver assigns to the selected variable in order to prefer paths with higher probabilities.

The decision how to assign a variable is based on the transition probabilities. If a variable $\sigma_{i+1,j}$ is to be assigned at depth $0 < i + 1 \leq n$, its value partly determines s_{i+1}, being the destination state of a transition with source s_i. We choose the value for $\sigma_{i+1,j}$ which corresponds to the state s_{i+1} to which the transition with the highest probability can be taken under the current partial assignment). This can be applied for several consecutive transitions in the future up to the complete path. However, as this computation is very expensive, we restrict the number of time steps we look ahead. For our test cases, assigning variables for 3 possible consecutive transitions in one step led to the best results.

Example 20 *Consider the DTMC $\mathcal{D}$ from Figure 2.1 on Page 20. Assume the binary encoding as described in Example 6 on Page 42. In the table below, a partial assignment ν_{part} of the variables for a state s_i and its successor s_{i+1} is shown; "?" indicates, that this variable is not yet assigned, "next" indicates that this variable will be assigned next.*

	s_i			s_{i+1}		
	$\sigma_{i,1}$	$\sigma_{i,2}$	$\sigma_{i,3}$	$\sigma_{i+1,1}$	$\sigma_{i+1,2}$	$\sigma_{i+1,3}$
ν_{part}	1	1	?	*next*	?	?

The current assignment determines state s_i to either be s_4 or s_5. Assigning 1 to the *next* variable, $\sigma_{i+1,1}$, which is the first variable for the successor state s_{i+1}, would lead to state s_6, inducing the transition from s_5 to s_6 having probability 1. As the most probable transition outgoing from s_4 or

s_5 would be the one leading to state s_1 with probability 0.7, we guide the SAT solver to assign a 1 here such that the transition (s_5,s_6) is chosen.

5.7 Symbolic path searching algorithms

In this section we present *BDD-based graph algorithms* to implement the path searching procedure `FindNewStates(...)` as invoked by Algorithm 6, see Section 5.7. We first explain how one can find the most probable path of a symbolically represented DTMC using a set-theoretic variant of Dijkstra's algorithm, called Flooding Dijkstra [GSS10]. This method is extended to allow the computation of the k most probable paths of a DTMC. This procedure can be directly embedded into the symbolic framework resulting in a *symbolic global search*. However, the direct application leads to an exponential blow-up of the search graph. Therefore we introduce an improved variant which—amongst other improvements—avoids this growth, called *adaptive global search*. Afterwards we present a new search method which symbolically searches for the most probable path fragments that extend the current subsystem. We call this approach the *symbolic fragment search*.

5.7.1 Flooding Dijkstra algorithm

The *Flooding Dijkstra* algorithm was introduced in [GSS10]. The algorithm computes a *shortest* path, which is in our context a *most probable* path, from the initial state of a DTMC to a target state. This is done by a forward fixpoint computation: For all states s of the DTMC an under-approximation of the largest probability of a path leading from the initial state s_I to s is iteratively improved.

Initially, the under-approximation is set to 1 for the initial states and 0 for all other states. An *update set*, which initially consists of the initial state, stores those states whose approximation was improved. This improvement needs to be propagated to their successors. The difference to the standard Dijkstra algorithm [Dij59] for computing shortest paths in a directed graph is that Flooding Dijkstra updates the approximations of the successors of *all* states from the update set in each iteration, instead of restricting the propagation to an *optimal* element with the minimal currently known cost (highest probability). That means, in contrast to the depth-first search of the standard Dijkstra algorithm, the Flooding Dijkstra algorithm operates in a breadth-first-style over sets of states. Therefore it can be efficiently implemented using MTBDD operations. The algorithm maintains a *directed acyclic graph* (DAG) which contains all most probable paths of minimal length from the initial state to all other states (with respect to the current approximation). These paths need to be saved in a DAG and not as a tree, as there may be two or more paths of the same (highest) probability and length leading to the same state. After the fixpoint is reached, i.e., when the approximation becomes exact, the last step of the algorithm extracts a single *most probable path*.

For details on the differences between the Flooding and standard Dijkstra variant we refer to the dissertation of Johann Schuster [Sch12, Sec. 5.2.1].

The Flooding Dijkstra algorithm is sketched in Algorithm 7. Assume a DTMC $\mathscr{D} = (S, I, P, L)$ and a set of target states $T \subseteq S$.

Algorithm 7 The Flooding Dijkstra algorithm for symbolic DTMCs

```
**FloodingDijkstra**(MTBDD $\widehat{P}$, BDD $\widehat{I}$, BDD $\widehat{T}$)
**begin**
    BDD $UD := \widehat{I}$        MTBDD $PR_1 := \widehat{I}$        MTBDD $PR_2 := \emptyset$
    MTBDD $SP := \emptyset$        MTBDD $SPG := \emptyset$
    **while** $UD \neq \emptyset$ **do**
        $PR_2 :=$ CalcProbs($UD, PR_1, \widehat{P}$)
        $UD :=$ GetStates($PR_2, PR_1$)
        $PR_1 :=$ UpdatePR($UD, PR_1, PR_2$)
        $SPG :=$ UpdateSPG($UD, \widehat{P}, SPG$)
    **end while**
    $SP :=$ GetPath($SPG, \widehat{I}, \widehat{T}$)
    **return** $(SP, SPG)$
**end**
```

Parameters

$\widehat{P}$ is an MTBDD representing the transition probability matrix P defined over variable sets *Var* and *Var'*.

$\widehat{I}$ is a BDD representing the set of initial states $Init_{\mathscr{D}} \subseteq S$ of $\mathscr{D}$.

$\widehat{T}$ is a BDD representing the set of target states $T \subseteq S$.

Variables

UD is a BDD that stores the *update set* of those states that gained higher probabilities in the last iteration.

PR_1 is an MTBDD that stores the probability approximations *before* improved probabilities are propagated to successor states.

PR_2 is an MTBDD that stores the probability approximations *after* a propagation step.

SPG is a an MTBDD representing a *directed acyclic graph* (*DAG*) which contains all most probable paths of minimal length in $\mathscr{D}$.

SP is a an MTBDD representing a most probable paths of $\mathscr{D}$

Return value

Returned is both the DAG *SPG* and the path *SP*.

Methods

`CalcProbs(BDD` UD`, MTBDD` PR_1`, MTBDD` $\widehat{P}$`)` propagates the improved probability values of states in *UD* to their successors. It calculates the maximal currently known path probability to go from the initial state s_I to a state in *UD* (as stored in PR_1) for all states s' with at least one predecessor $s \in UD$ and from there in one step to s' (according to $\widehat{P}$). The name “flooding” indicates that hereby the maximum is formed over all states $s \in UD$ with $P(s,s') > 0$.

Using (MT)BDD operations this is done as follows: PR_1 stores the probabilities of the most probable paths detected so far. Initially, only the initial state has probability 1 and all other states 0. $PR_1 \cdot UD$ restricts the probabilities to the states in *UD* that shall be updated. $PR_1 \cdot UD \cdot \widehat{P}$ yields an MTBDD defined over *Var* and *Var*$'$. For an assignment $\nu_{s,s'}$ this MTBDD gives the probability to go from s_I to s (according to PR_1) and then takes the direct transition from s to s'. We quantify over the source states of the transitions, i. e., the variables *Var*, taking the maximum over all possibilities. Since the resulting MTBDD is defined over *Var*$'$, we rename these variables to *Var*. This yields PR_2.

`GetStates(MTBDD` PR_2`, MTBDD` PR_1`)` determines those states whose probability approximations were improved during the last propagation step. The resulting BDD contains those states whose probability in PR_2 is higher than in PR_1. In detail, the operation APPLY(>, PR_2, PR_1) is carried out.

`UpdatePR(BDD` UD`, MTBDD` PR_1`, MTBDD` PR_2`)` computes the maximum over the MTBDDs PR_1 and PR_2, where *UD* is assumed to contain those states whose values in PR_2 are higher than in PR_1. This function is implemented using ITE(UD,PR_2,PR_1).

`UpdateSPG(BDD` UD`, MTBDD` $\widehat{P}$`, MTBDD` SPG`)` maintains the DAG according to the improved probabilities for the update set *UD*. Those transitions of *SPG* that lead to a state in *UD*, i.e., to a state whose probability was improved, are removed. The transitions that cause the higher probabilities in PR_2 are added.

`GetPath(MTBDD` SPG`, BDD` $\widehat{I}$`, BDD` $\widehat{T}$`)` extracts one most probable path from the DAG *SPG* by walking backward from $\widehat{T}$ to one of its predecessors until $\widehat{I}$ is reached.

Procedure

First, the empty BDD and MTBDD variables are created (Line 1- 3). The while-loop runs until no further states can be improved, i. e., $UD = \emptyset$. Inside the loop, first the probabilities are propagated and PR_2 is computed (Line 4). This is used to calculate the improved states and save them in the update set (Line 5). The maximum over the probabilities is computed and saved in PR_1 (Line 6). Finally, the DAG is updated according to the improved states and saved in *SPG* (Line 7).

When the while-loop terminates, the MTBDD *SPG* contains, for each state s, *all* most probable paths with a minimal number of transitions from s_I to s. One of these paths is extracted (Line 9). For both path searching methods that we describe in the following, it is often beneficial not to return only a single path, but all paths in *SPG* to a target state. Then we perform a backward breadth-first search in *SPG* starting from $\widehat{T}$ in order to have only states from which the target state is reached inside *SPG*. Therefore, the algorithm returns both *SP* and *SPG* (Line 10).

Example 21 *If the Flooding Dijkstra algorithm is run on the DTMC from Figure 2.1 on Page 20, a DAG containing all paths of maximal probability from the initial state to all other states is computed. This graph and the most probable path to the target state s_3 of probability 0.25 and length 2 are depicted below. The framed values above the nodes of the DAG show the computed probability values from PR_1. The path is determined by invoking a backward breadth-first search from the target state. Please note that in case there was another path of the same probability and length to the target state s_3 in the DTMC, there would be another path from s_0 to s_3 in the DAG. Please note also that standard Dijkstra would compute the same result, only the way of computation differs.*

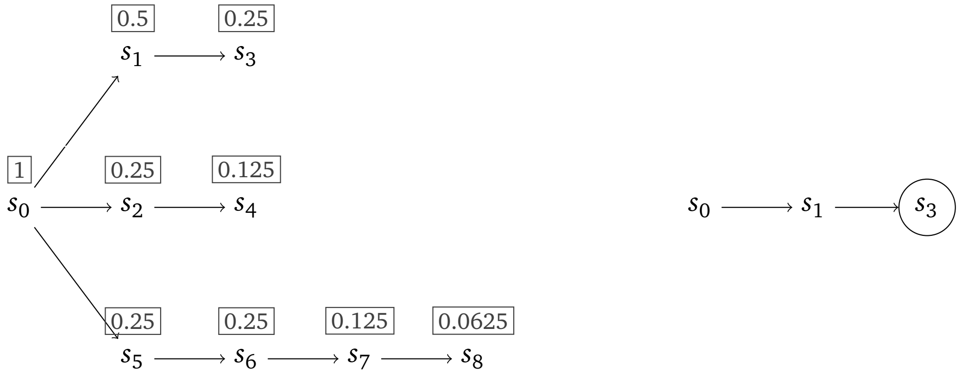


Figure 5.6: Result of the Flooding Dijkstra Algorithm for DTMC $\mathscr{D}$

5.7.2 Adaptive symbolic global search

In [GSS10], a symbolic version of a k-shortest path search was presented. This corresponds to the k most probable paths, leading from the initial state to a target state ordered with respect to their probabilities. Utilized for a counterexample search, the value of k is not fixed beforehand but the search terminates if enough probability mass is accumulated [HKD09]. The main components are the calculation of a most probable path by the Flooding Dijkstra, see Section 5.7.1, and a transformation of the DTMC such that the most probable path in the altered system corresponds to the second-most probable path in the original system. For details on the MTBDD operations we refer to the appendix of [GSS10].

5.7.2.1 Symbolic global search

The adaption to our symbolic framework for the computation of critical subsystems is straightforward. Intuitively, for every new path the states on this path are available in BDD-representation and returned to Algorithm 6 on Page 97 as the BDD `newStates`. As long as still further states are needed to form a critical subsystem, the k-shortest path search continues to deliver the next shortest path. This adaption is shown in Algorithm 8.

Algorithm 8 The global search algorithm for symbolic DTMCs

```
SymbolicGlobalSearch(MTBDD $\widehat{P}$, BDD $\widehat{I}$, BDD $\widehat{T}$, BDD $SP$)
begin
    MTBDD $SPG$
    if $SP \neq 0$ then
        $(\widehat{P}, \widehat{I}, \widehat{T}) :=$ Change($\widehat{P}, \widehat{I}, \widehat{T}, SP$)
    end if
    $(SP, SPG) :=$ ShortestPath($\widehat{P}, \widehat{I}, \widehat{T}$)
    return $SP$
end
```

Parameters

$\widehat{P}$ is an MTBDD representing the transition probability matrix P defined over variable sets Var and Var'.

$\widehat{I}$ is a BDD representing the set of initial states $Init_{\mathcal{D}} \subseteq S$ of $\mathcal{D}$.

$\widehat{T}$ is a BDD representing the set of target states $T \subseteq S$.

SP is a BDD storing the states of the current shortest respectively most probable path.

Variables

SPG is a an MTBDD representing a *directed acyclic graph* (*DAG*) which contains all most probable paths of minimal length in $\mathcal{D}$.

Return value

This algorithm returns an MTBDD SP representing a most probable path.

Methods

`ShortestPath(MTBDD` $\widehat{P}$`, BDD` $\widehat{I}$`, BDD` $\widehat{T}$`)` is a symbolic implementation of the Flooding Dijkstra algorithm as described in Section 5.7.1. It returns a BDD that represents the states occurring on a most probable path from the initial state represented by $\widehat{I}$ to a target state from the set represented by $\widehat{T}$.

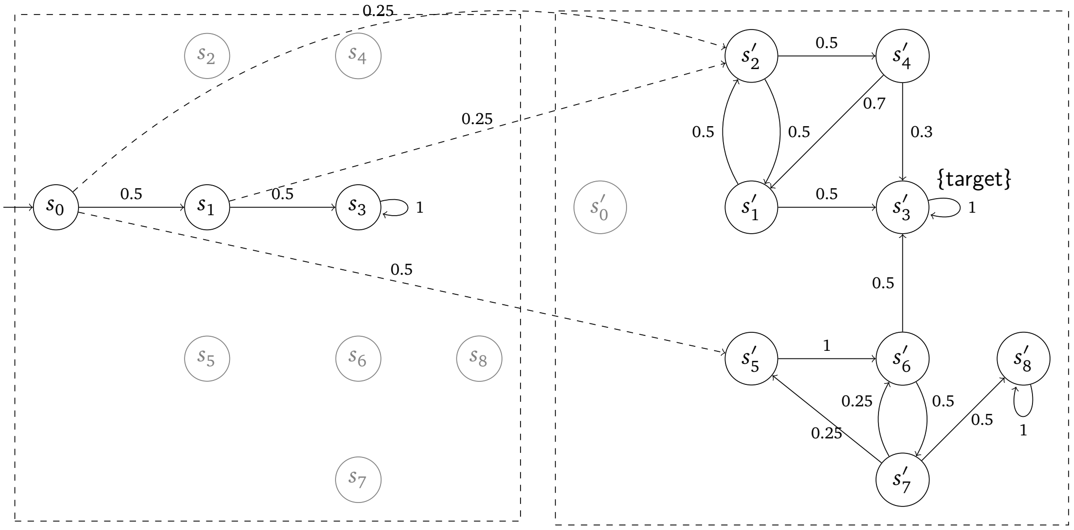


Figure 5.7: Altered system

Change(MTBDD $\widehat{P}$, BDD $\widehat{I}$, BDD $\widehat{T}$, MTBDD SP) changes the DTMC $(\widehat{P},\widehat{I},\widehat{T})$ such that the most probable path in the new DTMC corresponds to the second-most probable path of the original DTMC.

The idea is to use two copies of the DTMC. The initial state is in the first copy while the target states are in the second copy. In the first copy, only edges that belong to the shortest path *SP* remain unchanged. All other edges are redirected to the corresponding states in the second copy, in which all edges remain unchanged. Then all paths—with the exception of *SP*—lead from the initial state to a target state in this modified graph. As a consequence, every path to a target state needs to take at least one transition at a certain depth that does not occur in *SP* at the same depth. Therefore, the most probable path in the modified graph corresponds to the second-most probable path in the original model. These modifications can be performed symbolically by adding an additional state variable that indicates which copy is used. The MTBDD $\widehat{P}$ is therefore extended by two variables: One for the source and one for the target state. The adaptation of the transition relation is straightforward.

Procedure

First, the algorithm checks whether there has been a previous shortest path (Line 2). Only in that case, the system has to be altered according to the Change-method (Line 3). Otherwise, the algorithm is called for the first time and no path needs to be excluded. The current shortest path is then computed, stored in the BDD SP and returned (Lines 5- 6).

Example 22 *To explain the procedure of altering the system using the above method, assume that the first global path $\pi_1 = s_0, s_1, s_3$ of the DTMC shown in Figure 2.1 on Page 20 is found (the first global path in Example 16 on Page 100). The altered system is depicted in Figure 5.7. The two copies* *of the DTMC are marked by dashed rectangles. The initial state is still s_0 while the new target state is the copy s_3' of s_3. Only the transitions of the most probable path reside in the left copy. States, that are not reachable any more are drawn gray and we omit the transitions. The dashed transitions are the ones that do not belong to the most probable path and lead from the left to the right copy. The most probable path in this altered system is now the path $\pi_2 = s_0, s_5', s_6', s_3'$. This corresponds to the second-most probable path of the original system as in Example 16. Note, that in order to find the next path, this whole altered system is again copied.*

As the whole modified system is copied again in every iteration (after each path), this procedure leads to an exponential blow-up in the system size. The MTBDD resulting from the iterative application of this altering grows also rapidly and renders this method inapplicable to systems which require a large number of paths, as our test cases show. A further drawback is that many of the computed paths do not extend the subsystem and therefore do not lead to any progress. Nevertheless, we have implemented this approach in order to compare it to other ones, and call it *symbolic global search*.

5.7.2.2 Adaptive symbolic global search

To present a symbolic global search approach that is usable for practical instances, we developed a new improved variant. In comparison to the straightforward approach this on the one hand avoids the exponential blow-up of the system size and on the other hand saves many search iterations by adding sets of paths. Furthermore, the search algorithm uses an adaptive strategy in order to find small counterexamples. We call this approach the *adaptive symbolic global search*, depicted in Algorithm 9.

Algorithm 9 The adaptive global search algorithm for symbolic DTMCs

```
AdaptiveSymbolicGlobalSearch(MTBDD $\widehat{P}$, BDD $\widehat{I}$, BDD $\widehat{T}$, BDD subSys)
begin
    MTBDD $SPG$    MTBDD $SP$    MTBDD $\widehat{P}'$    BDD $\widehat{I}'$    BDD $\widehat{T}'$      (1)
    if subSys $= \emptyset$ then                                                   (2)
        $(\widehat{P}', \widehat{I}', \widehat{T}') := (\widehat{P}, \widehat{I}, \widehat{T})$                     (3)
    else                                                                           (4)
        $(\widehat{P}', \widehat{I}', \widehat{T}') :=$ Change($\widehat{P}$, $\widehat{I}$, $\widehat{T}$, subSys)          (5)
    end if                                                                         (6)
    $(SP, SPG) :=$ ShortestPath($\widehat{P}', \widehat{I}', \widehat{T}'$)                     (7)
    return $SPG$                                                                   (8)
end
```

Parameters Parameters are as for Algorithm 8 on Page 115: $\widehat{P}$, $\widehat{I}$, $\widehat{T}$ and subSys symbolically represent the input DTMC and its target states. Furthermore, the BDD subSys stores the current subsystem.

Variables

SP is an MTBDD storing the shortest path.

SPG is a an MTBDD representing a *directed acyclic graph* (*DAG*) which contains all most probable paths of minimal length in $\mathscr{D}$.

$\widehat{P}', \widehat{I}', \widehat{T}'$ are an MTBDD and BDDs as usual to maintain a copy of the system.

Return value

The algorithm returns the MTBDD *SPG*.

Methods The methods differ from the ones for Algorithm 8 as follows.

`ShortestPath(MTBDD` $\widehat{P}$, `BDD` $\widehat{I}$, `BDD` $\widehat{T}$`)` returns the BDD representation of the DAG *SPG* containing all most probable paths from a state of $\widehat{I}$ to a state of $\widehat{T}$. A symbolic implementation of the Flooding Dijkstra algorithm as described in Section 5.7.1 is used to obtain the DAG *SPG*. Performing a backward reachability analysis from the target states given by $\widehat{T}$ yields all states that induce the shortest path that lead to target states.

`Change(MTBDD` $\widehat{P}$, `BDD` $\widehat{I}$, `BDD` $\widehat{T}$, `MTBDD subSys):` In the improved version, the idea is to not only exclude the most probable path from the system, but all paths at once which contain only edges that are already contained in the current subsystem. This can be done by applying the transformation to the original DTMC and the current subsystem `subSys` instead of only the most probable path *SP*. This avoids doubling the search graph after each path and therefore the exponential blow-up, as in every step again the original system is used; the system size only increases linearly. Furthermore, we always obtain a path having a transition that is not yet contained in the subsystem at each time. Since the subsystem contains all transitions of the original DTMC connecting two states in the subsystem, each new transition also contributes a new state. Therefore the subsystem is extended in each iteration. Note that $\widehat{P}$ stays always unmodified.

Procedure

First, the variables *SPG*, $\widehat{P}', \widehat{I}', \widehat{T}$ are initialized (Lines 1-5). If the MTBDD `subSys` is empty, i. e., this is the first search iteration, the input system is just copied (Lines 2- 3). Otherwise, the `Change`-method is applied with respect to the current subsystem as described above (Line 5). For the resulting system the set of shortest paths of minimal length is computed (Line 7). Returned to Algorithm 6 is this set of paths, stored in the DAG *SPG* (Line 8). To further speed up the calculation, we add all states of this DAG to the subsystem at once. As we see in our experiments, the search process is accelerated by orders of magnitude.

Adding many paths to the subsystem at once involves the risk that the computed counterexample has more states than needed and is of a probability that is not close to the probability bound.

We overcome this problem by using an *adaptive search strategy*: In case the current subsystem is *critical*, i. e., its probability exceeds the probability bound, we measure the difference of these probabilities. If the difference is higher than a predefined $\delta > 0$, we perform backtracking to the state of the search procedure before the last spanning tree was added. We now add only single paths and terminate as soon as the probability bound is again exceeded.

5.7.3 Adaptive symbolic fragment search

In contrast to the previous approach, where we search for whole paths through the system, we now aim at finding most probable *path fragments* as described in Section 5.5.2. This symbolic version is depicted in Algorithm 10.

Algorithm 10 The adaptive fragment search for symbolic DTMCs

```
AdaptiveSymbolicFragmentSearch(MTBDD $\widehat{P}$, BDD $\widehat{I}$, BDD $\widehat{T}$, MTBDD subSys)
begin
    MTBDD $SPG$     MTBDD $SP$     BDD subSysStates                                  (1)
    if subSys $= \emptyset$ then                                                      (2)
        $SPG :=$ ShortestPath($\widehat{P}$, $\widehat{I}$, $\widehat{T}$)            (3)
    else                                                                              (4)
        subSysStates $:=$ ToStateBDD(subSys)                                          (5)
        $(SP, SPG) :=$ ShortestPath($\widehat{P} \setminus$ subSys, $SubSysStates$, $SubSysStates$)   (6)
    end if                                                                            (7)
    return $(SP, SPG)$                                                                (8)
end
```

Parameters All parameters are as in Algorithm 9 on Page 117 with the exception that the subsystem is now stored by an MTBDD representing the transition probability matrix instead of only the state set.

Variables

SP is an MTBDD storing the shortest path.

SPG is a an MTBDD representing a *directed acyclic graph* (*DAG*) which contains all most probable paths of minimal length in $\mathscr{D}$.

`subSysStates` is a BDD storing the states of the current subsystem.

Return value

This algorithm returns the MTBDDs *SP* and *SPG*.

Methods

`ShortestPath(MTBDD` $\widehat{P}$`, BDD` $\widehat{I}$`, BDD` $\widehat{T}$`)` returns again the DAG *SPG* describing all paths of the highest probability leading to a target state.

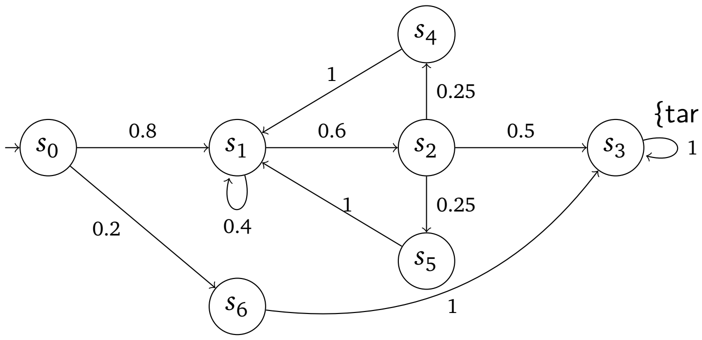
`ToStateBDD(MTBDD subSys)` computes a BDD describing all states that occur as source state or destination state for one of the transitions of `subSys`. When `subSys` is defined over the variables $Var = \{x_1, \ldots, x_n\}$ and $Var' = \{x'_1, \ldots, x'_n\}$, this is done by first building the set $OUT := \exists x'_1, \ldots, x'_n.\ \mathtt{subSys}_{\mathrm{bool}}$ of all states with an outgoing transition. Afterwards, the set $IN' := \exists x_1, \ldots, x_n.\ \mathtt{subSys}_{\mathrm{bool}}$ of states with incoming transitions is built. These resulting BDDs have to be defined over the same variable set, therefore we perform variable renaming for the set of states with incoming transitions: $IN := IN'[x'_1 \to x_1] \ldots [x'_n \to x_n]$. Building the union $IN \cup OUT$ yields the needed BDD.

Procedure

First, the variables are initialized (Lines 1- 3). The algorithm then checks whether the parameter `subSys` is empty, i. e., whether this is the first search iteration (Line 2). If this is the case, the base paths leading from the initial state to a target state are computed by invoking the most probable path search (Line 3). The resulting paths, stored in the BDD *SPG* are returned to Algorithm 6 (Line 8). If `subSys` is not empty, then a part of the subsystem has already been determined. In this case we compute the state BDD `subSysStates` by invoking `ToStateBDD(subSys)` (Line 5). The most probable path algorithm is called to find the most probable paths from a state in *SubSysStates* to a state in `subSysStates` inside the DTMC induced by $\widehat{P}$ without using direct transitions from `subSysStates` to `subSysStates` (Line 6). Note again that the resulting DAG might describe a large number of such paths.

In contrast to the symbolic global search described in Section 5.7.2, the MTBDD for the transition relation needs no significant modification. We only need to exclude the current subsystem from the further search in every iteration, which didn't lead to any remarkable overheads in our experiments. We also use the adaptive search algorithm in order to gain small critical subsystems and call this the *adaptive symbolic fragment search*.

Example 23 *To illustrate the advantages of the adaptive fragment search, consider the following toy example DTMC with a single target state s_3.*



Using the adaptive global search, first the path $\pi_1 = s_0, s_1, s_2, s_3$ of probability 0.24 *is found. The self-loop on s_1 is a transition starting and ending at states of the above path and will thus be automatically contained in the DAG SPG. The next path is $\pi_2 = s_0, s_6, s_3$ having probability* 0.2*. Each of the next steps will extend π_1 by traversing the loops $\pi_3 = s_1, s_2, s_4, s_1$ and $\pi_3 = s_1, s_2, s_5, s_1$.*

On the contrary, the fragment search will first find the path π_1 and then the path fragments π_3 and π_4, both in one step. If the probability bound was not higher than $\lambda = 0.8$, this suffices to be a critical subsystem. As in most of the available benchmarks such symmetric loop-behavior is very common, this example is illustrative.

5.8 Discussion of related work

The approaches introduced in this chapter can be roughly categorized in such that work on explicit and such that work on symbolic representations of DTMCs.

As already discussed, the explicit global search approach follows the k shortest path approach in [HKD09]. In contrast to that approach, we represent a counterexample in the more compact way of a critical subsystem. Moreover, this method is mostly superior in terms of running times. The hierarchical counterexample generation has to be compared to [ADvR08] where the SCCs of a DTMC are abstracted as a whole. The main difference lies in the finer consideration of sub-SCCs, such that the whole loop-structure of a graph is exploited.

In [AL10], also critical subsystems are used, there called *diagnostic subgraphs*. The generation is done in a different way, as no global information about the system is exploited while an a priori generation of the state space is avoided using on-the-fly algorithms.

The only other approach for the symbolic generation of counterexamples using MTBDDs is given in [GSS10]. We adopted this for the global search approach, but as opposed to their work we do not obtain an MTBDD representing a set of k shortest paths, but rather a representation as a critical subsystem of the DTMC. In addition, we improved the running time of [GSS10] by reducing the number of variable shiftings for the transition MTBDD in our implementation.

Details about the experimental comparison to other approaches are given in Chapter 8, as long as implementations are publicly available.

CHAPTER 6

Minimal critical subsystems for discrete-time Markov models

Summary In this chapter we present several dedicated approaches for the counterexample generation for DTMCs and PAs. As properties, we handle both reachability properties and ω-regular properties. Our goal is—as in the previous Chapter 5—to compute critical subsystems of the original system, see Definition 50 on Page 82. The difference is, that here we want to determine the actual *minimal* critical subsystem in terms of the number of states.

For reachability properties, we give both for DTMCs and for PAs an encoding suited for (non optimizing) SMT solvers, that—if satisfiable for input system and property—yields a critical subsystem of a certain size n in terms of the number of the states. A binary search over n is used to compute the actual *minimal critical subsystem*. In order to achieve better running times and to handle larger input systems, we define an alternative *mixed integer linear programming* [Sch86] (MILP) formulation of these problems. Together with a number of optimizations, we are able to solve this minimality problem for many benchmarks. As further result we present MILP encodings suited to compute minimal critical subsystems for DTMCs or PAs and ω-regular properties. Note that this is the first approach directly able to compute counterexamples for this kind of properties. The results presented in this chapter are summarized in [4], while there instead of PAs only MDPs were used.

Using MILP solvers not only enables to compute very small counterexamples but also to measure the quality of the heuristic approaches as described in Chapter 5. Using intermediate results without a proof of minimality, small—and in many cases already minimal—subsystems can be computed in a few seconds, which renders the MILP approach practically feasible if no proof of minimality is desired.

Background The foundations needed for this chapter are—besides the definitions of the probabilistic models that are used here—the explicit model checking of DTMCs and MDPs based on solving linear equation systems, see Sections 2.3.1.1 and 2.3.1.2. The reader is assumed to be familiar with the basics and the model checking of ω-regular properties for DTMCs and MDPs, see Section 2.3.3. For basic information about SMT solving and MILP see the introduction to solving techniques in Sections 2.6.

6.1 Minimal critical subsystems for DTMCs

In this section we present two approaches for computing a minimal critical subsystem (MCS) of a DTMCs based on SMT solving or MILP solving, respectively. Although our experiments revealed the MILP approach to be superior, we start by representing both encodings, as the SMT constraints give a better intuition on how the subsystems are computed. Besides performance, an important advantage of using MILP solvers is that during the solving process a lower bound on the optimal solution is obtained while both the current solution—the currently obtained critical subsystem—and the lower bound are successively improved. This leads to the possibility of halting the MILP solver at an arbitrary point in time and still obtaining the best solution so far, as well as a precise indication of the size of an MCS.

Additionally to the problem encodings, we provide several optimizations in the form of *redundant constraints* that are aimed at speeding up the solving process by detecting conflicts at an earlier stage.

6.1.1 Reachability properties

In the following we assume a DTMC $\mathscr{D} = (S, s_I, P, L)$ with a unique initial state and a set of target states $T \subseteq S$. All approaches are also applicable for multiple initial states, see Remark 5 on Page 25. The task is to find a critical subsystem for $\mathscr{D}$ and the reachability property $\mathbb{P}_{\leq\lambda}(\lozenge T)$. This computed subsystem should be minimal in terms of the number of states. We assume that all irrelevant states have been removed beforehand, see Definition 23 on Page 33.

Remark 27 (Minimality of subsystems) *Minimal critical subsystems are not unique. However, in our method we are able to compute a minimal critical subsystem with the highest probability of reaching a target state from the initial state (which of course is, again, not necessarily unique).*

Moreover, a minimal number of transitions is not induced. All approaches presented in the following can easily be adapted to the minimization in terms of the number of transitions.

Complexity The complexity of computing a minimal critical subsystem for reachability properties of DTMCs is to the best of our knowledge unknown. However, for arbitrary (nested) PCTL properties the problem is known to be NP-complete [CV10].

6.1.1.1 SMT formulation

Intuition We give an SMT formula over linear real arithmetic whose set of satisfying variable assignments corresponds to the set of critical subsystems of $\mathscr{D}$. It is crucial to assign correct probabilities to all states, i. e., states inside the subsystem are assigned their reachability probability to reach a target state inside the subsystem while states outside need to explicitly be assigned 0. Otherwise, they could contribute to the reachability probability inside the subsystem which would yield wrong results. Furthermore, the probability of the initial state has to exceed the probability bound λ.

Variables

$x_s \in [0,1] \subseteq \mathbb{R}$ is introduced for each $s \in S$. This *characteristic variable* is assigned 1 if and only if the corresponding state s is contained in the subsystem.

$p_s \in [0,1] \subseteq \mathbb{R}$ is to be assigned the probability of reaching a target state inside the subsystem from each $s \in S$, if and only if s is contained in the subsystem, i. e., $x_s = 1$. Otherwise this *probability variable* is assigned 0.

Constraints

$$\text{minimize} \quad \sum_{s \in S} x_s \tag{6.1a}$$

such that

$$\forall s \in T. \quad \big(x_s = 0 \land p_s = 0\big) \oplus \big(x_s = 1 \land p_s = 1\big) \tag{6.1b}$$

$$\forall s \in S \setminus T. \quad \big(x_s = 0 \land p_s = 0\big) \oplus \Big(x_s = 1 \land p_s = \sum_{s' \in succ(s)} P(s,s') \cdot p_{s'}\Big) \tag{6.1c}$$

$$p_{s_I} > \lambda \tag{6.1d}$$

where $\oplus$ denotes *exclusive or*.

Explanation As we are interested in a *minimal* critical subsystem, we have to minimize the number of x_s-variables that are assigned value 1. This corresponds to minimizing the sum over all characteristic variables x_s for $s \in S$ (Constraint 6.1a). Since most state-of-the-art SMT solvers for LRA cannot cope with the optimization of objective functions, we apply a binary search in the range $\{1, \dots, |S|\}$ to obtain the optimal value of the objective function. Starting with $k_l = 1$ and $k_u = |S|$, we iteratively search for critical subsystems whose number of states is between k_l and $k_m := k_l + (k_u - k_l)/2$. If we find such a subsystem with k states then we set k_u to $k-1$; otherwise, we set k_l to k_m+1. The search is repeated until $k_u < k_l$. The smallest k for which a solution was found yields the size of the MCS at hand.

If x_s is zero, the corresponding state s does not belong to the subsystem. Then its reachability probability is forced to be 0 (first summand in Constraint 6.1b). Otherwise, target states that are contained in the subsystem have probability 1 (second summand in Constraint 6.1b). Note that an MCS does not need to contain all target states. The reachability probability of all non-target states in the subsystem is given as the weighted sum over the probabilities of their successor states (Constraint 6.1c), see Section 2.3.1.1 for the standard computation of reachability probabilities for DTMCs. In order to obtain a critical subsystem we additionally require the probability variable p_{s_I} of the initial state to be assigned such that the "critical" probability threshold λ is exceeded. (Constraint 6.1d).

Formula size Recall our input DTMC $\mathscr{D} = (S, s_I, P, L)$ and assume the number of states to be $n_{\mathscr{D}}$ and the number of transitions $m_{\mathscr{D}}$.

Constraint 6.1b introduces a subformula of constant size for each target state $t \in T \subseteq S$. Constraint 6.1c introduces a subformula for each state $s \in S \setminus T$ while the size of these subformulae depends on the number of transitions. As in the context of the whole formula each transition occurs only once, the size of the resulting SMT formula is in $O(n_{\mathscr{D}} + m_{\mathscr{D}})$.

Soundness and completeness of the SMT encoding are stated by the following theorem. The proof is given at the end of this chapter in Section 6.3.1. Intuitively, for all encodings soundness refers to the fact that *each satisfying assignment* induces a minimal critical subsystem for the model and the property present. Furthermore, completeness states that there is a satisfying assignment for each minimal critical subsystem.

Theorem 4 *The SMT formulation* (6.1a)–(6.1d) *is sound and complete.*

6.1.1.2 MILP formulation

As mentioned before, experiments show that obtaining a solution for larger DTMCs is rather time-consuming using this SMT formulation up to being infeasible. Consider the high number of disjunctions present in this formula. This leads to the fact that when assigning variables, there are often only a few implications. Therefore, many different cases have to be tried while searching for a solution.

We now provide an MILP formulation for finding an MCS for reachability properties.

Intuition In contrast to the SMT constraints we now have to argue using upper and lower bounds on variables instead of implications. Formally, instead of disjunctions we use sums over variables. The idea of the encoding stays the same.

Variables

$x_s \in \{0, 1\} \subseteq \mathbb{Z}$ is the *characteristic variables* for each $s \in S$, now explicitly required to be integer within the range of 0 and 1.

$p_s \in [0,1] \subseteq \mathbb{R}$ is the *probability variable* for each $s \in S$ as before.

Encoding

$$\text{minimize} \quad -\frac{1}{2} p_{s_I} + \sum_{s \in S} x_s \tag{6.2a}$$

such that

$$\forall s \in T. \quad p_s = x_s \tag{6.2b}$$

$$\forall s \in S \setminus T. \quad p_s \leq x_s \tag{6.2c}$$

$$\forall s \in S \setminus T. \quad p_s \leq \sum_{s' \in succ(s)} P(s,s') \cdot p_{s'} \tag{6.2d}$$

$$p_{s_I} > \lambda \,. \tag{6.2e}$$

Explanation The probability variable p_s of a state $s \in T$ is assigned 1 iff the state is contained in the MCS, i. e., $x_s = 1$ (Constraint 6.2b). Analogously, for each state $s \in S \setminus T$ which is not included in the subsystem, i. e., $x_s = 0$, p_s is assigned 0. This is done by requiring $p_s \leq x_s$ (Constraint 6.2c). Note that for states included in the critical subsystem, this does not restrict the value of p_s. An additional upper bound on the probability p_s is given by the weighted sum of the reachability probabilities $p_{s'}$ of the successor states s' (Constraint 6.2d). The final constraint requires again the probability variable of the target state to have a value greater than the probability bound λ (Constraint 6.2e).

Constraints 6.2b–6.2e together with the same objective function as in the SMT formulation (Constraint 6.1a) already yield a minimal critical subsystem. However, the objective function can be improved in two aspects. Since Constraint 6.2d only imposes an upper bound on p_s, we only obtain a lower bound on the desired reachability probability as the value of p_{s_I} while we would like to obtain the concrete value. Additionally, we want to obtain an MCS with maximal probability. Both can be achieved by maximizing the value of p_{s_I}. To that end, we add p_{s_I} to the minimizing objective function together with a negative coefficient. A factor $0 < c < 1$ is needed because if we only subtracted p_{s_I}, then the solver could add an additional state if this would yield $p_{s_I} = 1$. Here, we choose $c = \frac{1}{2}$. This yields the objective function (Constraint 6.2a).

Formula size The number of real and integer variables as well as the number of constraints is in $O(n_{\mathcal{D}})$. As before, Constraint 6.2d depends on the number of transitions, all in all the size of the formula, i. e., the number of non-zero variable coefficients in the MILP formulation, is in $O(n_{\mathcal{D}} + m_{\mathcal{D}})$.

Theorem 5 *The MILP formulation* (6.2a)–(6.2e) *is sound and complete.*

The proof of this theorem can be found in Section 6.3.2.

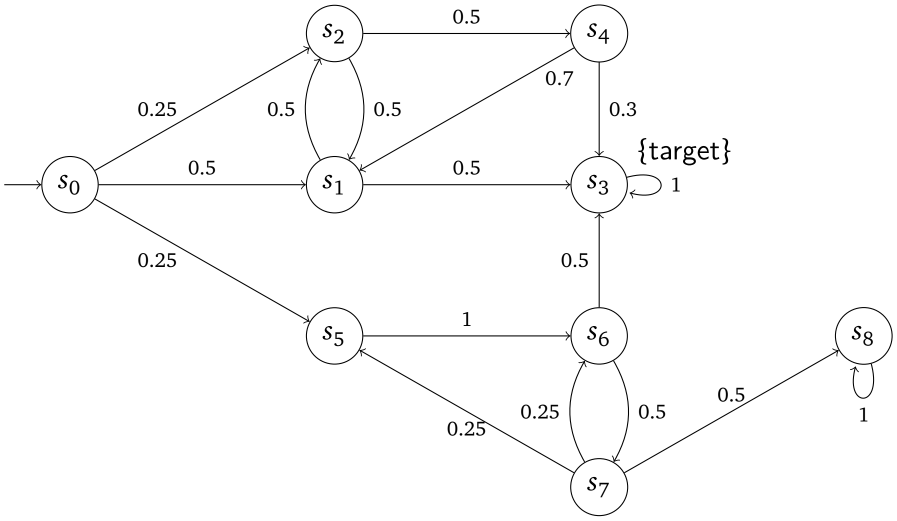


Figure 6.1: DTMC $\mathcal{D}$

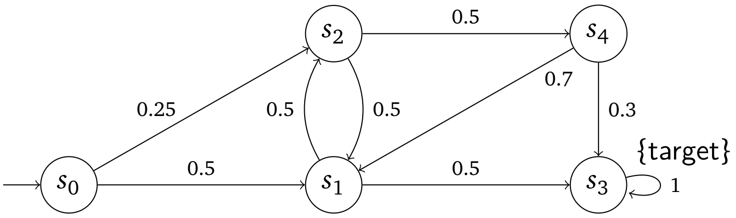


Figure 6.2: Minimal critical subsystem $\mathcal{D}' \sqsubseteq \mathcal{D}$ for property $\mathbb{P}_{\leq 0.7}(\lozenge \mathsf{target})$

Example 24 *Consider again the DTMC $\mathcal{D}$ from Example 1 on Page 20, again depicted in Figure 6.1 and the reachability property $\mathbb{P}_{\leq 0.7}(\lozenge \mathsf{target})$, which is violated for $\mathcal{D}$. The relevant states for this property, see Definition 23, consist of the set $S \setminus \{s_8\}$. For the MILP formulation we can therefore ignore s_8. Note that this will result in sub-stochastic distributions already for the input DTMC. We introduce the binary variables $x_{s_0}, \dots, x_{s_7} \in \{0,1\} \subseteq \mathbb{Z}$ and the real-valued variables $p_{s_0}, \dots, p_{s_7}$ from $[0,1] \subseteq \mathbb{R}$. The MILP is then given by:*

$$\begin{array}{lll}
\textit{minimize} & -\frac{1}{2}p_{s_0} + x_{s_0} + x_{s_1} + x_{s_2} + x_{s_3} + x_{s_4} + x_{s_5} + x_{s_6} + x_{s_7} & \\
\textit{such that} & p_{s_3} = x_{s_3} & \\
 & p_{s_0} \leq x_{s_0} & p_{s_0} \leq 0.5p_{s_1} + 0.25p_{s_2} + 0.25p_{s_5} \\
 & p_{s_1} \leq x_{s_1} & p_{s_1} \leq 0.5p_{s_2} + 0.5p_{s_3} \\
 & p_{s_2} \leq x_{s_2} & p_{s_2} \leq 0.5p_{s_1} + 0.5p_{s_4} \\
 & p_{s_4} \leq x_{s_4} & p_{s_4} \leq 0.7p_{s_1} + 0.3p_{s_3} \\
 & p_{s_5} \leq x_{s_5} & p_{s_5} \leq p_{s_6} \\
 & p_{s_6} \leq x_{s_6} & p_{s_6} \leq 0.5p_{s_3} + 0.5p_{s_7} \\
 & p_{s_7} \leq x_{s_7} & p_{s_7} \leq 0.25p_{s_5} + 0.25p_{s_6} \\
 & p_{s_1} > 0.7\ . &
\end{array}$$

Solving this MILP yields the following optimal solution:

Variable	x_{s_0}	p_{s_0}	x_{s_1}	p_{s_1}	x_{s_2}	p_{s_2}	x_{s_3}	p_{s_3}	x_{s_4}	p_{s_4}	x_{s_5}	p_{s_5}	x_{s_6}	p_{s_6}	x_{s_7}	p_{s_7}
Value	1	0.75	1	1	1	1	1	1	1	1	0	0	0	0	0	0

This solution corresponds to the MCS shown in Figure 6.2.

6.1.2 Optimizations

In the following we propose a number of optimizations in the form of *redundant* constraints that help the solver to prune the search space and to detect unsatisfiable or non-optimal branches of the search space at an early stage of the solving process. Imposing extra constraints to the MILP formulations intuitively means adding cutting planes which tighten the LP-relaxation of the MILP and may lead to better lower bounds on the optimal value.

In a nutshell, all constraints aim to a certain extent at guiding the MILP solver to only add states that are on paths from the initial state to a target state inside the MCS, as only such states will be part of an MCS.

6.1.2.1 Forward and backward constraints

Intuition We want to ensure that every non-target state has a successor state in the MCS by adding constraints which we call *forward cuts*. Likewise, we add *backward cuts*, which enforce every state except the initial state s_I to have a predecessor in the MCS. To avoid self-loops, we exclude a state itself from its successor and predecessor states.

Constraints

$$\forall s \in S \setminus T. \quad -x_s + \sum_{s' \in succ(s) \setminus \{s\}} x_{s'} \geq 0 \tag{6.3a}$$

$$\forall s \in S \setminus s_I. \quad -x_s + \sum_{s' \in pred(s) \setminus \{s\}} x_{s'} \geq 0 \tag{6.3b}$$

Explanation Both the forward cuts (Constraint 6.3a) and the backward cuts (Constraint 6.3b) are trivially satisfied if state s is not contained in the subsystem: x_s assigned 0 ensures that the left side of both inequations is at least 0. If state s is chosen to be contained in the subsystem, i. e., $x_s = 1$, then at least one successor/predecessor state s' must be contained, i. e., $x_{s'} = 1$, to achieve a positive number of successors/predecessors.

These constraints ensure that each initial state has a successor and every target state has a predecessor, while other states have at least one predecessor and successor. However, if a connected subset of inner states is selected during the solving process these constraints might be satisfied while this subset is not reachable from the initial state or cannot reach a target state.

Example 25 *Consider the DTMC $\mathscr{D}$ from Example 24 depicted in Figure 6.1 on Page 128. Assume, the characteristic variable x_{s_1} for state s_1 is assigned $x_{s_1} = 1$. Applying Constraint 6.3a yields*

$$\begin{aligned} & -x_{s_1} + x_{s_2} + x_{s_3} \geq 0 \\ \xRightarrow{x_{s_1}=1}\ & x_{s_2} + x_{s_3} \geq 1 \end{aligned}$$

which implies that either x_{s_2} or x_{s_3} needs to be assigned 1 which is true in the subsystem $\mathscr{D}' \sqsubseteq \mathscr{D}$ from Figure 6.2 on Page 128. Assume contrary now the characteristic variable x_{s_6} to be assigned $x_{s_6} = 0$. Applied to Constraint 6.3a this yields

$$\begin{aligned} & -x_{s_6} + x_{s_3} + x_{s_7} \geq 0 \\ \xRightarrow{x_{s_1}=0}\ & x_{s_3} + x_{s_7} \geq 0 \end{aligned}$$

which does not enforce a value greater than 0 for both x_{s_3} and x_{s_7}. The case for the backward constraints is analogous.

6.1.2.2 SCC constraints

Intuition To prevent the solver from selecting isolated connected sets of states, we utilize the SCC decomposition of the input DTMC $\mathscr{D}$. Let $\mathscr{S} \subseteq \mathscr{P}(S)$ be the set of all nontrivial SCCs of $\mathscr{D}$. States of an SCC $S' \in \mathscr{S}$ (not containing the initial state s_I) can be reached from outside S' through one of the input states $Inp(S')$ only. Therefore we ensure that a state of an SCC can only be selected if at least one of the SCC's input states is selected. The corresponding constraints are referred to as the *SCC input cuts*.

Analogously we define *SCC output cuts*: Paths from a state inside an SCC $S' \in \mathscr{S}$ that do not contain a target state have to go through one of the SCC's output states $Out(S')$ to actually reach a target state. Therefore, if no output state of an SCC S' is selected, we do not select any state of the SCC.

Constraints

$$\forall S' \in \{S'' \in \mathscr{S} \mid S'' \cap \{s_I\} = \emptyset\}.\, \forall s \in S' \setminus Inp(S').\quad x_s \leq \sum_{s' \in Inp(S')} x_{s'} \tag{6.4a}$$

$$\forall S' \in \{S'' \in \mathscr{S} \mid S'' \cap T = \emptyset\}.\, \forall s \in S'.\quad x_s \leq \sum_{s' \in Out(S')} x_{s'}\ . \tag{6.4b}$$

Explanation Again both the SCC input cuts (Constraint 6.4a) and the SCC output cuts (Constraint 6.4b) are trivially satisfied for a state s that is not contained in the subsystem, as $x_s = 0$ is always less or equal than the sum over the input state of the SCC. Contrary, for the input cuts, if a state $s \in S'$ is contained in the subsystem, i. e., $x_s = 1$, at least one of the input states $s' \in Inp(S')$

of the SCC also has to be contained in order to achieve a value that is greater or equal to 1. For output cuts, at least for one of the output states $s \in Out(S')$ of the SCC the characteristic variable $x_{s'}$ has to be assigned 1.

Note that using these SCC constraints, still isolated loops inside an SCC can be selected without a connection to the input and output states of the SCC.

Example 26 *Consider DTMC $\mathcal{D}$ from Figure 6.1 on Page 128 and the SCC consisting of the states s_1, s_2, s_4, where s_1 and s_2 are input states. Applying Constraint 6.4a to the characteristic variable x_{s_4} yields:*

$$\begin{aligned} & x_{s_4} \leq x_{s_1} + x_{s_2} \\ \overset{x_{s_4}=1}{\Longrightarrow} \; & x_{s_1} + x_{s_2} \geq 1 \end{aligned}$$

which implies that one of the input states needs to be included in the subsystem, which is again true for $\mathcal{D}' \sqsubseteq \mathcal{D}$ from Figure 6.2 on Page 128. If x_{s_4} is assigned 0*, the constraint is trivially true. The case for the SCC output cuts is analogous.*

6.1.2.3 Reachability constraints

We now present a set of constraints which precisely enforce the reachability of a certain set of states. An assignment will satisfy these additional constraints only if all selected states lie on a path from the initial to a target state inside the selected subsystem. Without these constraints, this is only ensured by the state-minimality as enforced by the objective function, where isolated states do not contribute to the probability of reaching target states from the initial state and are therefore not part of a minimal subsystem.

Intuition We introduce the notions of *forward* and *backward reachability constraints*. For both, the concept is to define a partial order on the states by using real-valued variables for each state: Along a loop-free path, these variables are assigned increasing values, i. e., each state has a higher value than its predecessor. Each state needs to have a transition that is part of such a path. For the forward reachability the only state that does not need to have a predecessor that is lower in the partial ordering is the initial state. Thereby, every state included in the subsystem needs to be reachable via a loop-free path from the initial state. Analogously, we define the backward reachability of target states, where every state except the target states needs a successor with a higher value associated. For this optimization we present both an SMT and an MILP encoding to ease the understanding.

Variables – forward reachability

$r^{\rightarrow}_s \in [0,1] \subseteq \mathbb{R}$ for all $s \in S \setminus \{s_I\}$ are used to associate to each state a real number. These variables define a partial order on the states with respect to $<$.

$t^{\rightarrow}_{s,s'} \in [0,1] \subseteq \mathbb{Z}$ for all $s,s' \in S \setminus \{s_I\}$ are integer variables indicating whether transition $(s,s') \in E_{\mathscr{D}}$ is chosen for a path or not. These variables are only needed for the MILP encoding.

SMT-constraints – forward reachability

$$\forall s' \in S \setminus \{s_I\}. \quad \Big(x_{s'} = 0 \vee \bigvee_{s \in pred(s')} (x_s \wedge r^{\rightarrow}_s < r^{\rightarrow}_{s'})\Big) . \tag{6.5a}$$

Explanation If $s' \in S$ is selected and reachable from s_I then there is a loop-free path $s_I = s_0, \ldots, s_n = s'$ such that $r^{\rightarrow}_{s_i} < r^{\rightarrow}_{s_{i+1}}$ for all $0 \leq i < n$ and all states on the path are selected for the subsystem, i. e., $x_{s_i} = 1$ for all $0 \leq i \leq n$. This is ensured by the fact that every state s' is either not selected ($x_{s'} = 0$) or it hast at least one predecessor s that has a lower r_s value than s'. This ensures a loop-free path, as a loop would revisit a state that already has a higher value. The path can only terminate at the initial state s_I for which this constraint is not defined. As seen by this encoding, a backward reachability from each state to the initial state is defined. However, in the intuitive sense this yields a forward reachability from the initial state.

MILP-constraints – forward reachability

$$\forall s' \in S \setminus \{s_I\}. \forall s \in pred(s'). \quad t^{\rightarrow}_{s,s'} \leq x_s \tag{6.6a}$$

$$\forall s' \in S \setminus \{s_I\}. \forall s \in pred(s'). \quad r^{\rightarrow}_s < r^{\rightarrow}_{s'} + (1 - t^{\rightarrow}_{s,s'}) \tag{6.6b}$$

$$\forall s' \in S \setminus \{s_I\}. \quad \sum_{s \in pred(s')} t^{\rightarrow}_{s,s'} = x_{s'} . \tag{6.6c}$$

Explanation Again, we encode a loop-free path $s_I = s_0, \ldots, s_n = s'$ such that $r^{\rightarrow}_{s_i} < r^{\rightarrow}_{s_{i+1}}$ for all $0 \leq i < n$. As we are not able to encode a disjunction over all predecessors where for one specific predecessor the ordering has to hold, we have to explicitly choose transitions using the $t^{\rightarrow}_{s,s'}$ variables.

In detail, each transition $(s,s') \in E_{\mathscr{D}}$ with $t^{\rightarrow}_{s,s'} = 1$ starts in a selected state s (Constraint 6.6a). If $x_{s'} = 0$ then Constraint 6.6c ensures that all variables $t_{s,s'}$ equal 0 for $s \in pred(s')$, i. e., $t_{s,s'} = 1$ implies $x_{s'} = 1$. Therefore $t_{s,s'} = 1$ implies that both s and s' are contained in the subsystem.

If a transition (s,s') is selected, i. e., $t^{\rightarrow}_{s,s'} = 1$, $r^{\rightarrow}_s < r^{\rightarrow}_{s'}$ has to hold (Constraint 6.6b), which defines the partial order on selected states. The constraints defined in Constraint 6.6c imply that from each selected state s' not being the initial state, one incoming transition $t^{\rightarrow}_{s,s'}$ has to be selected. One can show by induction that this ensures that for each selected state s' there is a path in the subsystem from s_I to s'.

The constraints defining backward reachability from target states are defined analogously:

Variables – backward reachability

$r^{\leftarrow}_{s} \in [0,1] \subseteq \mathbb{R}$ for all $s \in S \setminus T$ again define a partial order on all states except the target states.

$t^{\leftarrow}_{s,s'} \in [0,1] \subseteq \mathbb{Z}$ for all $s, s' \in S \setminus T$ for choosing transitions are again only needed for the MILP constraints.

SMT-constraints – backward reachability

$$\forall s \in S \setminus T. \quad \Big(x_s = 0 \vee \bigvee_{s' \in succ(s)} (x_{s'} \wedge r^{\leftarrow}_{s} < r^{\leftarrow}_{s'}) \Big) \ . \tag{6.7a}$$

Explanation Analogously to forward reachability from the initial state, we encode a loop-free path terminating at a target state, where every successor is required to have a higher value with respect to the partial ordering.

MILP-constraints – backward reachability

$$\forall s \in S \setminus T. \forall s' \in succ(s). \quad t^{\leftarrow}_{s,s'} \le x_{s'} \tag{6.8a}$$

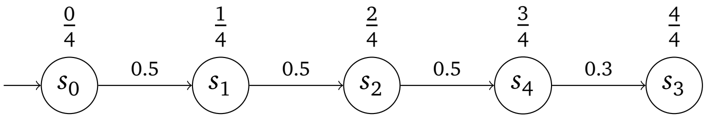
$$\forall s \in S \setminus T. \forall s' \in succ(s). \quad r^{\leftarrow}_{s} < r^{\leftarrow}_{s'} + (1 - t^{\leftarrow}_{s,s'}) \tag{6.8b}$$

$$\forall s \in S \setminus T. \quad \sum_{s' \in succ(s)} t^{\leftarrow}_{s,s'} = x_s \ . \tag{6.8c}$$

Explanation A partial ordering on all states except the target states is defined. To that end, each state s needs to have an associated loop-free path defined by the choice of transition variables $t^{\leftarrow}_{s,s'}$ which “terminates” in a target state as these are the only states where Constraint 6.8b is not defined.

During the assignment process, the forward and backward reachability constraints prevent all critical subsystems with unreachable states. However, as there are additional variables for all states and for all transitions, the usage of these cuts is expensive.

Example 27 *Consider the following path of DTMC $\mathcal{D}$ from Figure 6.1 on Page 128:*



The numbers above the states indicate possible values of the $r^{\rightarrow}_{s}$-variables satisfying the MILP forward constraints. Note that $r^{\rightarrow}_{s} \in [0,1] \subseteq \mathbb{R}$. Consider the forward constraints (6.6a)-(6.6c)*: For this path, we have for instance that $t^{\rightarrow}_{s_0,s_1} = t^{\rightarrow}_{s_1,s_2} = t^{\rightarrow}_{s_2,s_4} = t^{\rightarrow}_{s_4,s_3} = 1$. Constraint 6.6b now applied to s_2 and s_1 yields:*

$$r^{\rightarrow}_{s_1} < r^{\rightarrow}_{s_2} + (1 - t^{\rightarrow}_{s_1,s_2})$$

$$\xRightarrow{t^{\rightarrow}_{s_1,s_2}=1} r^{\rightarrow}_{s_1} < r^{\rightarrow}_{s_2}$$

which implies that the $r^{\rightarrow}$-value of s_2 has to be larger than the one for s_1. This holds for the values we assigned in our example. As for each selected state exactly one $t^{\rightarrow}$-variable is assigned 1 *by Constraint 6.6c and each state except of the initial state s_0 needs a predecessor with a smaller $r^{\rightarrow}$-value, each path has to start with the initial state.*

Remark 28 (Usage of reachability constraints) *In this section, the reachability constraints are specifically defined for the forward reachability from initial states and the backward reachability from target states. However, in the remainder we use these constraints for other sets of states, too. For instance, for Markov decision processes we need to ensure the reachability of states with certain properties. For convenience, we always reference the explanations given above.*

6.1.2.4 Correctness of the optimizations

The three kinds of optimizations we presented, namely the *forward and backward cuts*, the *SCC cuts*, and the *reachability cuts* prune the search space. However, these constraints do not restrict the solution space, i. e., each *minimal* critical subsystem satisfies these constraints. This is stated in the following theorem. The proofs are given in Section 6.3.3.

Theorem 6 *Both the SMT formulation* (6.1a)–(6.1d) *and the MILP formulation* (6.2a)–(6.2e) *together with any (combination) of the three above optimizations are sound and complete.*

6.1.3 ω-regular properties

In this section we describe an approach to enable the computation of minimal critical subsystems for ω-regular properties. we first present an SMT encoding offering an intuitive understanding of the crucial parts and then give a formulation suited for MILP. For the MILP approach we later on state its soundness and completeness.

Let in the following $\mathscr{D} = (S, s_I, P, L)$ be a DTMC with a set of atomic propositions AP. Let $\mathscr{L}$ be an ω-regular property and $\mathscr{A} = (Q, q_I, \Sigma, \delta, F)$ the corresponding DRA with $\mathscr{L}(\mathscr{A}) = \mathscr{L}$ and $\Sigma = 2^{AP}$. We assume that for s_I the probability of satisfying $\mathscr{L}$ is higher than a certain upper probability bound $\lambda \in \mathbb{Q}$, i. e., $Pr^{\mathscr{D}}_{s_I}(\mathscr{L}) > \lambda$. The goal is to compute an MCS $\mathscr{D}' \sqsubseteq \mathscr{D}$ for which $Pr^{\mathscr{D}'}_{s_I}(\mathscr{L}) > \lambda$ also holds. As always, we assume all irrelevant states to be removed.

6.1.3.1 SMT encoding

Intuition For our encoding, we proceed as in the model-checking algorithm for ω-regular properties on DTMCs as described in Section 6.1.3. First, we compute the product $\mathscr{D} \otimes \mathscr{A}$ of the DTMC $\mathscr{D}$ and the DRA $\mathscr{A}$ as in Definition 29 on Page 38. The model checking is now reduced to compute the probability of reaching *accepting BSSCs* from $\mathscr{B}$ inside the product. We denote

$T = \bigcup_{B\in\mathscr{B}} B$. This is utilized in the implicit probability computation via the SMT constraints. The critical subsystem we compute will not be a part of the product automaton but of the original DTMC. This is achieved via an implicit projection from the product.

Variables

$x_B \in \{0,1\} \subseteq \mathbb{Z}$ for all accepting BSSCs $B \in \mathscr{B}$ of $\mathscr{D} \otimes \mathscr{A}$ are *characteristic variables* indicating whether B is included in the subsystem or not. Note that either all states of an accepting BSCC are included or none of them.

$x_s \in \{0,1\} \subseteq \mathbb{Z}$ for all states $s \in S$ are *characteristic variables* indicating whether a state is included in the subsystem. Note that these variables are not defined over the product automaton $\mathscr{D} \otimes \mathscr{A}$ as the result will be projected to the DTMC.

$p_{sq} \in [0,1] \subseteq \mathbb{R}$ for every state $(s,q) \in S \times Q$ is assigned the *reachability probability* of the states of the product automaton. Let p_{sq_I} denote the variable for the initial state $(s,q)_I \in S \times Q$ of the product.

Constraints

$$\text{minimize} \quad \sum_{s\in S} x_s \tag{6.9a}$$

such that

$$p_{sq_I} > \lambda \tag{6.9b}$$

$$\forall B \in \mathscr{B}.\, \forall (s,q) \in B.\, (x_B = 0 \land p_{sq} = 0) \oplus (x_B = 1 \land p_{sq} = 1 \land x_s = 1) \tag{6.9c}$$

$$\forall (s,q) \in (S \times Q) \setminus T.\, \big(x_s = 0 \land p_{sq} = 0\big) \oplus \big(x_s = 1 \land p_{sq} = \sum_{(s',q')\in succ_{\mathscr{D}\otimes\mathscr{A}}(s,q)} P'\big((s,q),(s',q')\big) \cdot p_{s'q'}\big) \tag{6.9d}$$

Explanation First, as for mere reachability, the number of characteristic variables x_s is minimized (Constraint 6.9a) while the probability variable p_{sq_I} for the initial state of the product has to be assigned such that its value exceeds the probability bound λ rendering the subsystem critical (Constraint 6.9b).

The handling of the accepting BSSCs $B \in \mathscr{B}$ is as follows: The probability of a state $(s,q) \in B$ is 1 (i. e., $p_{sq} = 1$) if and only if that BSCC is selected, (i. e., $x_B = 1$). Furthermore, a BSCC $B \subseteq S \times Q$ can only be selected if all states $(s,q) \in B$ are selected via their projection to S, i. e., $x_s = 1$ (Constraint 6.9c).

If the projection of a state $(s,q) \in (S \times Q) \setminus T$—that does not belong to an accepting BSCC—is select for containment in the subsystem, i. e., $x_s = 1$, the probability of reaching an accepting BSCC inside the subsystem is computed. This is similar to the formulation for reachability properties (Constraint 6.9d).

6.1.3.2 MILP encoding

Intuition The MILP encoding for this problem ensures—basically similar to the corresponding SMT encoding—that accepting BSCCs $B \in \mathscr{B}$ are assigned probability 1 iff contained in the subsystem. For the other states, the probability of reaching the contained accepting BSCCs is computed, while the number of all states satisfying these requirements is minimized.

Variables All variables are as for the SMT encoding, see Section 6.1.3.1.

Constraints

$$\text{minimize} \quad -\frac{1}{2} p_{sq_I} + \sum_{s \in S} x_s \tag{6.10a}$$

such that

$$p_{sq_I} > \lambda \tag{6.10b}$$

$$\forall B \in \mathscr{B}\ \forall (s,q) \in B. \quad p_{sq} = x_B \tag{6.10c}$$

$$\forall B \in \mathscr{B}\ \forall (s,q) \in B. \quad x_s \geq x_B \tag{6.10d}$$

$$\forall (s,q) \in S_{\mathscr{D}\otimes\mathscr{A}} \setminus T. \quad p_{sq} \leq x_s \tag{6.10e}$$

$$\forall (s,q) \in S_{\mathscr{D}\otimes\mathscr{A}} \setminus T. \quad p_{sq} \leq \sum_{(s',q') \in succ_{\mathscr{D}\otimes\mathscr{A}}((s,q))} P\big((s,q),(s',q')\big) \cdot p_{s'q'} \tag{6.10f}$$

Explanation As for reachability, the number of involved states is minimized via the x_s variables while the probability of the initial state is maximized and exceeds the probability bound λ, (Constraints 6.10a and 6.10b), see Section 6.1.1.2.

For all states $(s,q) \in B \in \mathscr{B}$ that belong to an accepting BSCC the probability variable p_{sq} is assigned 1 if and only if the whole BSCC B is included (Constraint 6.10c).

A BSCC B of the product can only be selected if all of its states $(s,q) \in B$ are selected via the projection to the DTMC $\mathscr{D}$, i. e., $x_s = 1$ (Constraint 6.10d).

If the probability contribution of a state (s,q) exceeds 0, the DTMC-state s is selected (Constraint 6.10e). Using Constraint 6.10f, a lower bound on the probability of reaching accepting BSCCs inside the MCS is computed.

Formula size The MILP formulation contains $O(n_{\mathscr{D}\otimes\mathscr{A}})$ real variables, $O(n_{\mathscr{D}})$ integer variables, $O(n_{\mathscr{D}\otimes\mathscr{A}})$ constraints and $O(n_{\mathscr{D}\otimes\mathscr{A}} + m_{\mathscr{D}\otimes\mathscr{A}})$ non-zero coefficients.

The correctness of the MILP formulation, proven in Section 6.3.4, reads as follows.

Theorem 7 *The MILP formulation* (6.10a)–(6.10b) *is sound and complete.*

6.2 Minimal critical subsystems for PAs

This section addresses the generation of minimal critical subsystems for probabilistic automata. As for DTMCs we are going to present approaches for reachability properties as well as arbitrary ω-regular properties. There are two main challenges: First, we have to find a scheduler resolving the nondeterminism that induces a minimal critical subsystem. Second, for ω-regular properties we meet the problem that in contrast to DTMCs the sets of accepting states are now dependent on possible schedulers. Computing these sets is expensive, see Definition 33 on Page 39.

We refrain from giving an SMT encoding because for this computationally more involved problem this is not feasible. We present MILP encodings both for reachability and for ω-regular properties.

6.2.1 Reachability properties

Let in the following $\mathcal{M} = (S, s_I, Act, \mathcal{P}, L)$ be a PA together with a set of target states $T \subseteq S$. We assume the PCTL reachability property $\mathbb{P}_{\leq\lambda}(\Diamond T)$ for $\lambda \in \mathbb{R}$ to be violated for $\mathcal{M}$. Please recall that we sometimes write $(\alpha, \mu) = \eta \in Act \times subDistr(S)$ for an action-distribution pair. By $\eta(s)$ we denote the probability assigned to s by the distribution μ for all $s \in S$. Furthermore, we use $supp(\eta)$ to denote $supp(\mu)$.

One would suppose the critical subsystem we want to compute to be a subsystem $\mathcal{M}' \sqsubseteq \mathcal{M}$, i. e., a PA as in Definition 17 on Page 29. Consider such a PA $\mathcal{M}'$ that is a minimal critical subsystem for $\mathcal{M}$ and the property $\mathbb{P}_{\leq\lambda}(\Diamond T)$. Since $\mathcal{M}'$ is critical, it violates this property. As explained in Section 2.3.1.2, there exists a memoryless deterministic scheduler $\sigma \in Sched^{\mathcal{M}}$ inducing a DTMC $\mathcal{D}_\sigma$ with a probability mass of reaching T exceeding λ. Furthermore, since $\mathcal{M}'$ is minimal and $\mathcal{D}_\sigma$ is a subsystem of $\mathcal{M}'$, $\mathcal{D}$ is also minimal. Therefore, our task is to find a DTMC $\mathcal{D}$ with $\mathcal{D} \sqsubseteq \mathcal{M}$ of minimal size in terms of the number of states.

Complexity For PAs, the complexity of computing a minimal critical subsystem is known and stated in the following theorem.

Theorem 8 ([CV10]) *Let $\mathcal{M}$ be a PA with $\mathcal{M} \not\models \mathbb{P}_{\leq\lambda}(\Diamond T)$ and $k \in \mathbb{N}$. The problem to decide whether there exists a critical subsystem $\mathcal{M}' \sqsubseteq \mathcal{M}$ for $\mathbb{P}_{\leq\lambda}(\Diamond T)$ with at most k states is NP-complete.*

6.2.1.1 Problematic states

For DTMCs, we excluded all irrelevant states, i. e., the ones from where no target state can be reached. For PAs, we also assume that our models have only relevant states, see Definition 24 on Page 34: We basically remove all states for which no scheduler exists such that there is path that leads to a target state. However, for a relevant state s, the target states T might not be reachable under certain schedulers, which poses a problem in our computations.

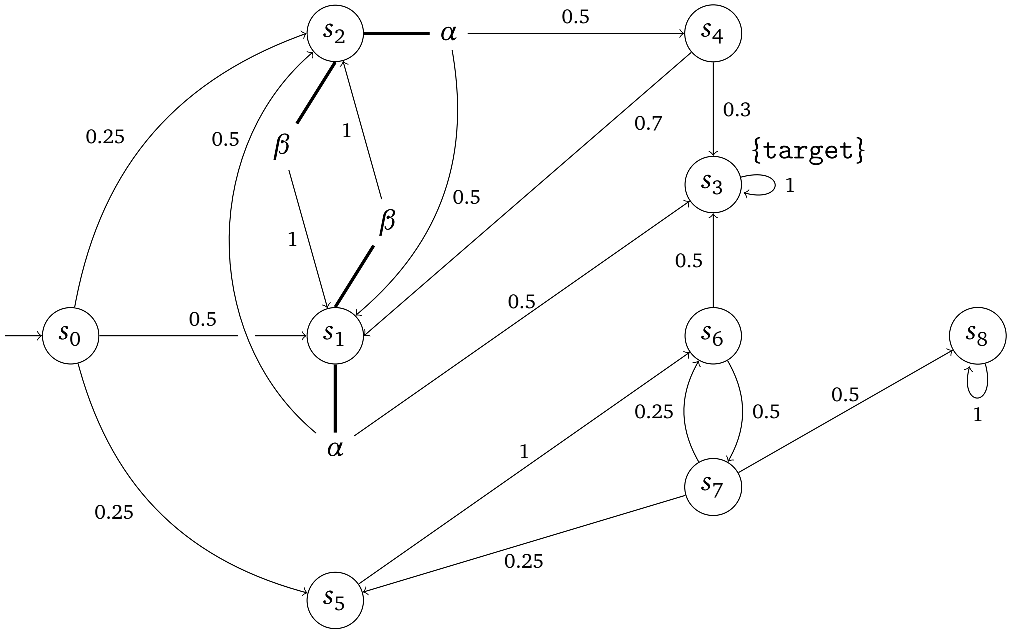


Figure 6.3: PA $\mathscr{M}$ with problematic states

Definition 56 (Problematic states and actions) *For a PA* $\mathscr{M} = (S, I, Act, \mathscr{P}, L)$ *and a set of target states* $T \subseteq S$*, the set of* problematic states *is given by* $S_{\text{probl}(T)} = \{s \in S \mid \exists \sigma \in Sched_{\mathscr{M}}.Pr_s^{\mathscr{M}^\sigma}(\Diamond T) = 0\}$*. The states* $S \setminus S_{\text{probl}(T)}$ *are called* unproblematic states.

We call $H_{\text{probl}(T)}(s) = \{\eta \in \mathscr{P}(s) \mid supp(\eta) \subseteq S_{\text{probl}(T)}\}$ *for* $s \in S_{\text{probl}(T)}$ *the set of* problematic action-distribution pairs *for problematic state s.*

When computing the probability of reaching target states, where not the actual *maximizing* scheduler is considered but—as in our case—the one inducing a minimal critical subsystem, the corresponding equation system does not have a unique solution for problematic states, as the following example shows.

Example 28 *To illustrate problematic states, consider the MDP (PA)* $\mathscr{M}$ *in Figure 6.3 as in Example 3 on Page 28. States* s_1 *and* s_2 *are both problematic since the scheduler which selects the action* β *in both states* s_1 *and* s_2 *prevents reaching the target state* s_3*. We cannot remove the outgoing transitions belonging to action* β *in a preprocessing step since a scheduler may choose* β *in one state and* α *in the other one. However, if a scheduler chooses* β *in both states, the following equations would be obtained for the probability computation as explained in Section 2.3.1:*

$$p_{s_1} = 1.0 \cdot p_{s_2}$$
$$p_{s_2} = 1.0 \cdot p_{s_1}\ .$$

A solution is $p_{s_1} = p_{s_1} = 1$ which means that the probability of reaching a target state is 1 for both states although no target state is reachable having this scheduler.

In our encoding we have to take care of these states. We do this implicitly by imposing additional constraints to assure that we consider only schedulers under which the target states are reachable from all subsystem states. Note that these constraints are not optional: Without these additional constraints, the reachability probabilities for states in a bottom SCC of the induced DTMC could be incorrectly determined to be 1 even if it does not contain a target state, leading to wrong results.

We are now ready to present the dedicated MILP encoding.

6.2.1.2 MILP encoding

Intuition The crucial part is to encode the nondeterministic choices to be made into the MILP formula, which shall be resolved by a deterministic memoryless scheduler, see Definition 21 on Page 30. This is done by introducing binary variables indicating the choice of an action-distribution pair at certain states. Moreover, the reachability of target states from problematic states has to be enforced.

Variables

$\sigma_{s,\eta} \in \{0,1\} \subseteq \mathbb{Z}$ for each state $s \in S \setminus T$ and each pair of action and distribution $\eta \in \mathscr{P}(s)$ that are available at s is a binary variable such that $\sigma_{s,\eta} = 1$ iff η is *selected* in state s by the scheduler that induces the critical subsystem.

$x_s \in \{0,1\} \subseteq \mathbb{Z}$ for each state $s \in S$ is the *characteristic variable* as for DTMCs to encode whether s belongs to the subsystem or not.

$p_s \in [0,1] \subseteq \mathbb{R}$ for each state $s \in S$ is a real-valued variable which is assigned the *probability* of s with respect to the scheduler determined by the $\sigma_{s,\eta}$ variables.

$r_s^{\leftarrow} \in [0,1] \subseteq \mathbb{R}$ for all problematic states $s \in S_{\mathrm{probl}(T)}$ are used to encode the *backward reachability* of non-problematic states.

$t_{s,s'}^{\leftarrow} \in \{0,1\} \subseteq \mathbb{Z}$ are used for the backward reachability of non-problematic states to assure the existence of a *transition* in the MCS between problematic states $s, s' \in S^{\mathscr{M}}_{\mathrm{probl}(T)}$ where the selected action-distribution pair $\eta \in H_{\mathrm{probl}(T)}(s)$ is problematic.

Constraints

$$\text{minimize} \quad -\frac{1}{2} p_{s_I} + \sum_{s \in S} x_s \tag{6.11a}$$

such that

$$p_{s_I} > \lambda \tag{6.11b}$$

$$\forall s \in T.\quad p_s = x_s \tag{6.11c}$$

$$\forall s \in S \setminus T.\quad p_s \leq x_s \tag{6.11d}$$

$$\forall s \in S \setminus T.\quad \sum_{\eta \in \mathscr{P}(s)} \sigma_{s,\eta} = x_s \tag{6.11e}$$

$$\forall s \in S \setminus T.\, \forall \eta \in \mathscr{P}(s).\quad p_s \leq (1 - \sigma_{s,\eta}) + \sum_{s' \in succ_{\mathscr{M}}(s,\eta)} \eta(s') \cdot p_{s'} \tag{6.11f}$$

$$\forall s \in S_{\text{probl}(T)}.\, \forall \eta \in H_{\text{probl}(T)}(s).\, \forall s' \in supp(\eta).\quad t^{\leftarrow}_{s,s'} \leq x_{s'} \tag{6.11g}$$

$$\forall s \in S_{\text{probl}(T)}.\, \forall \eta \in H_{\text{probl}(T)}(s).\, \forall s' \in supp(\eta).\quad r^{\leftarrow}_{s} < r^{\leftarrow}_{s'} + (1 - t^{\leftarrow}_{s,s'}) \tag{6.11h}$$

$$\forall s \in S_{\text{probl}(T)}.\, \forall \eta \in H_{\text{probl}(T)}(s).\quad (1 - \sigma_{s,\eta}) + \sum_{s' \in supp(\eta)} t^{\leftarrow}_{s,s'} \geq x_s\,. \tag{6.11i}$$

Explanation As for DTMCs, the number of states is minimized while the probability of the initial state is maximized (Constraint 6.11a). Moreover, it has to exceed the probability bound λ (Constraint 6.11b). Target states are assigned probability 1 ($p_s = 1$) if and only if they are included in the subsystem ($x_s = 1$), see Constraint 6.11c, while for all other states it is ensured that—if not included in the subsystem—they are assigned probability 0, see Constraint 6.11d.

Constraint 6.11e ensures that in each selected non-target state a single action-distribution pair is selected by the scheduler: If $x_s = 0$, no distribution on s is selected, if $x_s = 1$, the sum of all scheduler variables $\sigma_{s,\eta}$ has to be exactly 1.

Constraint 6.11f ensures the correct probability computation for non-target states $s \in S \setminus T$ and action-distribution pairs $\eta \in \mathscr{P}(s)$ available in s: If η is not selected by the scheduler ($\sigma_{s,\eta} = 0$), the constraint is trivially satisfied due to the term $(1 - \sigma_{s,\eta})$. Otherwise, the probability is computed as for DTMCs.

The following three Constraints 6.11g–6.11i ensure the backward reachability of each problematic state from a non-problematic state, see Section 6.1.2.3 for details.

To summarize, the MILP formulation yields a memoryless deterministic scheduler σ such that the probability of reaching the target states T in the DTMC induced by σ on the MCS exceeds λ. Together with the objective function, this DTMC is minimal in terms of the number of states and maximal in terms of the probability of the initial state.

Formula size The number of real-valued variables of the MILP is in $O(n_{\mathscr{M}})$. Due to the choice of action-distribution pairs, the number of integer variables depends also on the number of transitions, which yields $O(n_{\mathscr{M}} + m_{\mathscr{M}})$. The same holds for the number of non-zero coefficients in $O(n_{\mathscr{M}} + m_{\mathscr{M}})$.

The correctness of the MILP formulation is captured by the following theorem; its proof is provided in Section 6.3.5.

Theorem 9 *The MILP formulation* (6.11a)–(6.11i) *is sound and complete.*

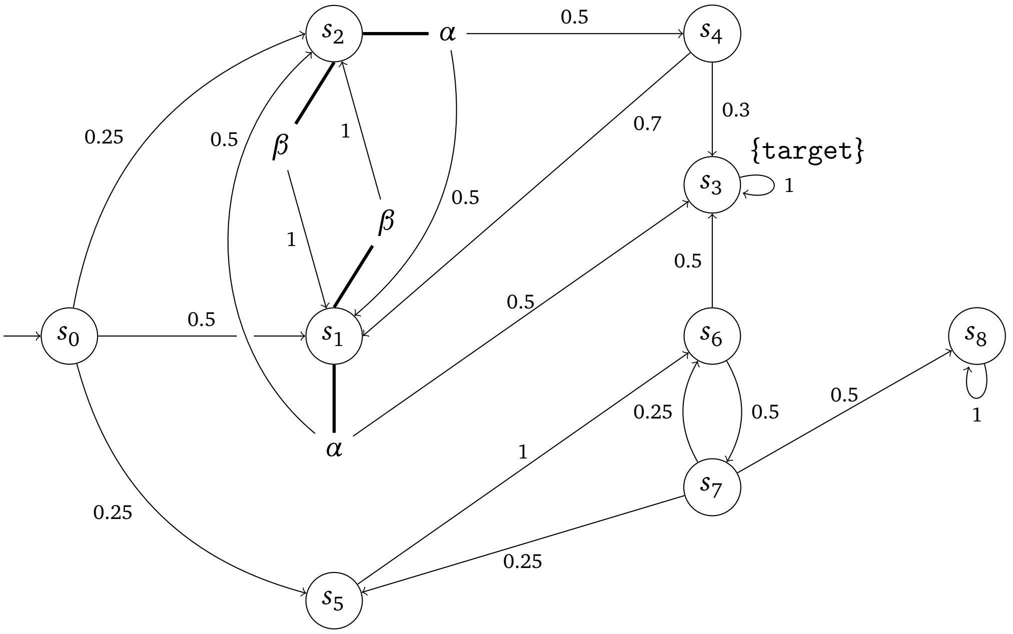


Figure 6.4: MDP (PA) $\mathcal{M}$

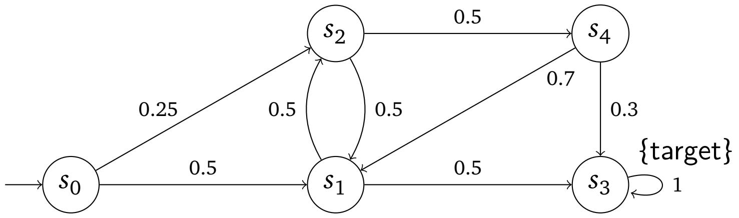


Figure 6.5: Minimal critical subsystem $\mathcal{M}' \sqsubseteq \mathcal{M}$ for property $\mathbb{P}_{\leq 0.7}(\Diamond \mathsf{target})$

Example 29 *We recall the MDP (PA) $\mathcal{M}$ as in Example 28 on Page 138, again depicted in Figure 6.4. The property $\mathbb{P}_{\leq 0.7}(\Diamond T)$ is violated for $T = \{s_3\}$. We observe that state s_8 is irrelevant for T, as it is absorbing (see Definition 24 on Page 34). As we saw in Example 28, the problematic states are $S_{\text{probl}(T)} = \{s_1, s_2\}$. As $\mathcal{M}$ is an MDP, for every action the choice of distribution is unique. we therefore refrain from using action-distribution pairs but for the sake of simplicity just actions. That means, that we now have problematic actions instead of problematic action-distribution pairs for each problematic state, which are $H_{\text{probl}(T)}(s_1) = \{\beta\}$ and $H_{\text{probl}(T)}(s_2) = \{\beta\}$.*

We use the following characteristic variables: $x_{s_0}, \ldots, x_{s_7} \in \{0,1\} \subseteq \mathbb{Z}$ for $s_0, \ldots s_7 \in S$. Accordingly, we have the real-valued probability variables $p_{s_0}, \ldots, p_{s_7} \in [0,1] \subseteq \mathbb{R}$. The scheduler variables will only be needed for the states where there are different actions, i. e., s_1 and s_2. Moreover, as for each action the choice of distribution is unique, we only need scheduler variables of the form $\sigma_{s,\alpha}$ for $s \in S$ and $\alpha \in Act$. For this example, we have: $\sigma_{s_1,\alpha}, \sigma_{s_1,\beta}, \sigma_{s_2,\alpha}, \sigma_{s_2,\beta} \in \{0,1\} \subseteq \mathbb{Z}$. For the reachability constraints (6.11g)–(6.11i) *we need the variables $r^{\leftarrow}_{s_1}, r^{\leftarrow}_{s_2} \in [0,1] \subseteq \mathbb{R}$ and*

$t^{\leftarrow}_{s_1,s_2}, t^{\leftarrow}_{s_2,s_1} \in [0,1] \subseteq \mathbb{R}$. *This will be needed if the solver chooses to take action β from one of the states s_1 or s_2, as then it will be enforced that for one state, an action α will be taken in order to ensure the reachability of a target state.*

$$
\begin{array}{lll}
\textit{minimize} & \multicolumn{2}{l}{-\frac{1}{2}p_{s_0} + x_{s_0} + x_{s_1} + x_{s_2} + x_{s_3} + x_{s_4} + x_{s_5} + x_{s_6} + x_{s_7}} \\
\textit{such that} & & \\
\end{array}
$$

$$
\begin{array}{lll}
p_{s_0} > 0.7 & & \\
p_{s_3} = x_{s_3} & & \\
p_{s_0} \leq x_{s_0} & & p_{s_0} \leq 0.5p_{s_1} + 0.25p_{s_2} + 0.25p_{s_5} \\
p_{s_1} \leq x_{s_1} & \sigma_{s_1,\alpha} + \sigma_{s_1,\beta} = x_{s_1} & p_{s_1} \leq (1 - \sigma_{s_1,\alpha}) + 0.5p_{s_2} + 0.5p_{s_3} \\
 & & p_{s_1} \leq (1 - \sigma_{s_1,\beta}) + p_{s_2} \\
p_{s_2} \leq x_{s_2} & \sigma_{s_2,\alpha} + \sigma_{s_2,\beta} = x_{s_2} & p_{s_2} \leq (1 - \sigma_{s_2\alpha}) + 0.5p_{s_1} + 0.5p_{s_4} \\
 & & p_{s_2} \leq (1 - \sigma_{s_2,\beta}) + p_{s_1} \\
p_{s_4} \leq x_{s_4} & & p_{s_4} \leq 0.7p_{s_1} + 0.3p_{s_3} \\
p_{s_5} \leq x_{s_5} & & p_{s_5} \leq p_{s_6} \\
p_{s_6} \leq x_{s_6} & & p_{s_6} \leq 0.5p_{s_3} + 0.5p_{s_7} \\
p_{s_7} \leq x_{s_7} & & p_{s_7} \leq 0.25p_{s_5} + 0.25p_{s_6} \\
\underbrace{\qquad\qquad}_{\text{(6.11b)–(6.11d)}} & \underbrace{\qquad\qquad\qquad\qquad}_{\text{(6.11e)}} & \underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{\text{(6.11f)}}
\end{array}
$$

For the encoding above note the following: For a constraint having the term $1 - \sigma_{s,\alpha}$ where we didn't introduce a σ-variable, we assume this to be trivially true and do not explicitly write it in the example. We now add the constraints needed for the reachability of target states.

$$
\begin{array}{lll}
t^{\leftarrow}_{s_1,s_2} \leq x_{s_2} & r^{\leftarrow}_{s_1} < r^{\leftarrow}_{s_2} + (1 - t^{\leftarrow}_{s_1,s_2}) & (1 - \sigma_{s_1,\beta}) + t^{\leftarrow}_{s_1,s_2} \geq x_{s_1} \\
t^{\leftarrow}_{s_2,s_1} \leq x_{s_1} & r^{\leftarrow}_{s_2} < r^{\leftarrow}_{s_1} + (1 - t^{\leftarrow}_{s_2,s_1}) & (1 - \sigma_{s_2,\beta}) + t^{\leftarrow}_{s_2,s_1} \geq x_{s_2} \\
\underbrace{\qquad\qquad}_{\text{(6.11g)}} & \underbrace{\qquad\qquad\qquad\qquad}_{\text{(6.11h)}} & \underbrace{\qquad\qquad\qquad\qquad\qquad\qquad}_{\text{(6.11i)}}
\end{array}
$$

Solving this MILP yields the following optimal variable assignment:

x_{s_0}	p_{s_0}	x_{s_1}	p_{s_1}	x_{s_2}	p_{s_2}	x_{s_3}	p_{s_3}	x_{s_4}	p_{s_4}	x_{s_5}	p_{s_5}	x_{s_6}	p_{s_6}	x_{s_7}	p_{s_7}
1	0.75	1	1	1	1	1	1	1	1	0	0	0	0	0	0

$\sigma_{s_1,\alpha}$	$\sigma_{s_1,\beta}$	$\sigma_{s_2,\alpha}$	$\sigma_{s_2,\beta}$
1	0	1	0

$t^{\leftarrow}_{s_1,s_2}$	$t^{\leftarrow}_{s_2,s_1}$
0	0

$r^{\leftarrow}_{s_1}$	$r^{\leftarrow}_{s_2}$
0	0

The resulting subsystem that corresponds to this variable assignment corresponds to the DTMC as in Example 24. The subsystem $\mathscr{M}' \sqsubseteq \mathscr{M}$ is depicted in Figure 6.5. Note that without changing the probability of the initial state and the number of states, e. g., in state s_1 action β could be chosen. In this case we had $\sigma_{s_1,\beta} = 1$ and $\sigma_{s_1,\alpha} = 0$. Furthermore, as β is a problematic action at state s_1, we had $t_{s_1,s_2} = 1$ and we would need values for the $r^{\leftarrow}$ variables, e. g., $r^{\leftarrow}_{s_1} = \frac{4}{5}$ and $r^{\leftarrow}_{s_2} = \frac{5}{5}$.

6.2.2 ω-regular properties

In this section we present our approach to determine a minimal critical subsystem for ω-regular properties and PAs.

Unlike model checking these properties, we cannot compute a maximizing scheduler but we have to consider all schedulers that might induce a minimal critical subsystem. Therefore we need to know the set of accepting end components of the product PA, see Definition 33 on Page 39. As this is dependent on the chosen scheduler, this is not feasible here.

Our approach encompasses the computation of these sets by the MILP solver which is well-suited to solve NP-hard problems. We encode the state sets that *almost surely satisfy* the ω-regular property directly into the MILP where these state sets define the target states.

Let in the following $\mathscr{M} = (S, s_I, Act, \mathscr{P}, L)$ be a PA and $\mathscr{A} = (Q, q_I, \Sigma, \delta, F)$ be a DRA with the acceptance condition $F = \{(R_i, A_i) \mid i = 1, \ldots, n\}$ such that $\mathscr{L}(\mathscr{A}) = \mathscr{L}$ for an ω-regular property $\mathscr{L}$. We assume for a probability bound $\lambda \in \mathbb{R}$ that $\mathscr{M} \not\models \mathbb{P}_{\leq\lambda}(\mathscr{L})$. Let $\mathscr{M} \otimes \mathscr{A} = \mathscr{M}' = (S \times Q, (s,q)_I, Act, \mathscr{P}', L')$ be the product of $\mathscr{M}$ and $\mathscr{A}$, see Definition 31 on Page 39. We assume that $\mathscr{M}'$ has no irrelevant states, i. e., *maximal endcomponents* are reachable from all states of $S \times Q$, see Definition 24 on Page 34 for removing irrelevant states and Section 2.3.3 for (maximal) end components.

In order to simplify notation we use $U = S \times Q$, $u = (s,q)$, and $u' = (s', q')$ to abbreviate the states of the product automaton. We also refer to a state $s \in S$ as the PA-state of $u = (s,q) \in U$. Let $n_{u,\alpha,\mu} = |succ_{\mathscr{M}\otimes\mathscr{A}}(u, \alpha, \mu)|$ denote the number of successor states of u under action-distribution pair $(\alpha, \mu) \in Act \times subDistr(S \times Q)$. Recall the notations $\eta = (\alpha, \mu) \in Act \times subDistr(S)$, $\eta(s) = \mu(s)$, and $supp(\eta) = supp(\mu)$.

We now explain the background for encoding the computation of accepting end components which are used as sets of target states. We use the following lemma:

Lemma 2 *Let $(R_i, A_i) \in 2^Q \times 2^Q$ be a pair of a Rabin acceptance condition, $\sigma\colon U \to Act \times subDistr(S \times Q)$ a memoryless deterministic scheduler, and $M_i \subseteq U$ a set of states with the following properties:*

1. $\forall u \in M_i. \sum_{u' \in succ(u,\sigma(u)) \cap M_i} \sigma(u)(u') = 1$
2. $M_i \cap (S \times R_i) = \emptyset$
3. *for each state $u \in M_i$ there is a path from u to a state in $S \times A_i$.*

Then the probability of satisfying the acceptance condition F because of the pair (R_i,A_i) is 1 for all $u \in M_i$.

For an acceptance pair $(R_i,A_i) \in F$ the set $M_i \subseteq U$ is intuitively built as follows: First, it is ensured that with respect to the scheduler σ the successors of all states of M_i are also inside M_i (Condition 1). Note that the sum of the outgoing probabilities has to be 1 here. No state of $S \times R_i$ must be part of M_i (Condition 2). Finally, a state from $S \times A_i$ has to be reachable from every state of M_i (Condition 3).

A set M_i encompasses an accepting end component together with the states that reach the end component with probability 1. The probability of reaching these sets of states corresponds to the probability of satisfying the ω-regular property.

Intuition The crucial part is to define the target states. This is achieved by encoding the Conditions as in Lemma 2 directly into the MILP formulation. By implicitly assuming a set of states that meet these conditions this reduces to computing an MCS for reachability probabilities as explained in Section 6.2.1. The goal is to determine a DTMC $\mathscr{D} \sqsubseteq \mathscr{M} \otimes \mathscr{A}$ which is *a subsystem of the product*. A projection on the original PA $\mathscr{M}$ does not yield a DTMC in general, as different states $(s,q),(s,q') \in U$ are projected on the same state of $\mathscr{M}$ but might have different outgoing action-distribution pairs.

Variables

$x_s \in \{0,1\} \subseteq \mathbb{Z}$ for each state $s \in S$ is the *characteristic variable* to encode whether s belongs to the subsystem or not.

$p_u \in [0,1] \subseteq \mathbb{R}$ for each state $u \in U$ is a real-valued variable which is assigned the *probability* of each state of the product automaton.

$\sigma_{u,\eta} \in \{0,1\} \subseteq \mathbb{Z}$ for each state $u \in U$ and each action-distribution pair $\eta \in \mathscr{P}(u)$ stores the selected *scheduler* such that $\sigma_{u,\eta} = 1$ iff η is selected in state u by the scheduler that induces the critical subsystem.

$m^i_u \in \{0,1\} \subseteq \mathbb{Z}$ for each $(R_i,A_i) \in F$ and $u \in U$ is a characteristic variable, i. e., $m^i_u = 1$ iff state u is *contained* in set M_i.

$r^i_u \in [0,1] \subseteq \mathbb{R}$ for all states $u \in U$ is used to define a *partial order* on states as backward reachability of $S \times A_i$ within M_i is needed.

$t^i_{u,u'} \in \{0,1\} \subseteq \mathbb{Z}$ for all states $u,u' \in U$ with $u' \in supp(\eta)$ and $\eta \in \mathscr{P}(u)$ is used to determine the *transitions* of a path that reaches a state from $S \times A_i$ according to the partial order on states defined by the r^i_u variables.

$r^M_u \in [0,1] \subseteq \mathbb{R}$ for all states $u \in U$ is used to define a *partial order* on states for backward reachability of $M = \bigcup_{i=1}^n M_i$.

$t^M_{u,u'} \in \{0,1\} \subseteq \mathbb{Z}$ for all states $u, u' \in U$ with $u' \in supp(\eta)$ and $\eta \in \mathscr{P}(u)$ is used to determine the *transitions* of a path that reaches a state of M according to the partial order on states defined by the r^i_u variables.

Constraints Please recall the abbreviations we introduced in the beginning of this section. We augment the encoding by some explanation in order to ensure readability.

$$\text{minimize} \quad -\frac{1}{2} p_{sq_I} + \sum_{s\in S} x_s \tag{6.12a}$$

$$\text{such that}$$

- Selection of at most one action-distribution pair per state:

$$\forall u = (s,q) \in U. \quad \sum_{\eta\in\mathscr{P}(u)} \sigma_{u,\eta} \leq x_s \tag{6.12b}$$

- Definition of the set M_i for all $i = 1, \ldots, n$:

$$\forall u \in U. \forall \eta \in \mathscr{P}(u) \text{ with } \sum_{u'\in U} \eta(u') < 1. \quad m^i_u \leq 1 - \sigma_{u,\eta} \tag{6.12c}$$

$$\forall u \in U. \forall \eta \in \mathscr{P}(u). \quad n_{u,\eta} \cdot (2{-}\sigma_{u,\eta} - m^i_u) + \sum_{u'\in supp(\eta)} m^i_{u'} \geq n_{u,\eta} \tag{6.12d}$$

$$\forall u \in S \times R_i. \quad m^i_u = 0 \tag{6.12e}$$

- Backward reachability of $S \times A_i$ within M_i for all $i = 1, \ldots, n$:

$$\forall u \in U. \forall \eta \in \mathscr{P}(u). \forall u' \in supp(\eta). \quad t^i_{u,u'} \leq m^i_{u'} + (1 - \sigma_{u,\eta}) \tag{6.12f}$$

$$\forall u \in U. \forall \eta \in \mathscr{P}(u). \forall u' \in supp(\eta). \quad r^i_u < r^i_{u'} + (1 - t^i_{u,u'}) + (1 - \sigma_{u,\eta}) \tag{6.12g}$$

$$\forall u \in S \times (Q \setminus A_i). \forall \eta \in \mathscr{P}(u). \quad (1 - \sigma_{u,\eta}) + \sum_{u'\in supp(\eta)} t^i_{u,u'} \geq m^i_u \tag{6.12h}$$

- Probability computation:

$$p_{sq_I} > \lambda \tag{6.12i}$$

$$\forall i = 1, \ldots, n. \forall u \in U. \quad p_u \geq m^i_u \tag{6.12j}$$

$$\forall u \in U. \quad p_u \leq \sum_{\eta\in\mathscr{P}(u)} \sigma_{u,\eta} \tag{6.12k}$$

$$\forall u \in U. \forall \eta \in \mathscr{P}(u). \quad p_u \leq (1 - \sigma_{u,\eta}) + \sum_{i=1}^n m^i_u + \sum_{u'\in supp(\eta)} \eta(u') \cdot p_{u'} \tag{6.12l}$$

- Backward reachability of $M = \bigcup_{i=1}^{n} M_i$ within the subsystem:

$$\forall u \in U. \forall \eta \in \mathscr{P}(u). \forall u' = (s',q') \in supp(\eta). \quad t^M_{u,u'} \leq x_{s'} + (1 - \sigma_{u,\eta}) \tag{6.12m}$$

$$\forall u \in U. \forall \eta \in \mathscr{P}(u). \forall u' \in supp(\eta). \quad r^M_u < r^M_{u'} + (1 - t^M_{u,u'}) + (1 - \sigma_{u,\eta}) \tag{6.12n}$$

$$\forall u = (s,q) \in U. \forall \eta \in \mathscr{P}(u). \quad (1-\sigma_{u,\eta}) + \sum_{i=1}^{n} m^i_u + \sum_{u' \in supp(\eta)} t^M_{u,u'} \geq x_s \,. \tag{6.12o}$$

Explanation The target function is defined as before, see the encoding for reachability properties in Section 6.2.1. Constraint 6.12b defines a valid scheduler by ensuring that for each selected state at most one action-distribution pair is chosen. Contrary to reachability, it is not possible to select exactly one pair: If a subsystem is determined by the selected states $S' \subseteq S$ of the PA, we implicitly select the subsystem $S' \times Q$ of the product automaton. By this it is not guaranteed that in the DTMC induced by the scheduler of the product automaton from each state (s,q) an accepting BSCC is reachable. Since we later require that from each state in $S' \times Q$ an accepting BSCC is reachable under the selected action-distribution pair, we solve this problem by allowing not to select an action. If no action is chosen, Constraint 6.12k ensures that the probability p_u is assigned 0, as the sum will be equal to 0.

The sets M_i $(i = 1, \dots, n)$ are built according to Conditions 1– 3 of Lemma 2. Condition 1 states that for each $u \in M_i$ the probability of staying in M_i has to be 1. This is ensured in two steps: First we forbid that a state u is in M_i if under the selected action the sum of the probabilities of the outgoing edges is less than 1, see Constraint 6.12c.

Note that for each state in M_i at least one out-going action-distribution pair is selected, since the probability of states without selected action is 0, but Constraint 6.12j sets the probability of M_i-states to 1. Secondly we ensure the closure of M_i under successors by Constraint 6.12d. If state u belongs to M_i (i. e., $m^i_u = 1$) and the action-distribution pair η is chosen by the scheduler (i. e., $\sigma_{u,\eta} = 1$), all successors of u with respect to the pair η have to belong to M_i. The term $n_{u,\eta} \cdot (2 - \sigma_{u,\eta} - m^i_u)$ is 0 if and only if η is selected for u and $u \in M_i$ holds. In this case the sum over the corresponding variables $m^i_{u'}$ of the successors u' of u has to be assigned at least the number of the successors of u. Constraint 6.12e ensures that M_i does not contain any R_i state, see Condition 2.

In order to ensure backward reachability from $S \times A_i$ within M_i and thereby the satisfaction of Condition 3, we use the constraints known from the DTMC optimizations and PA reachability properties (cf. Section 6.1.2.3). The corresponding constraints are given in Constraints 6.12f–6.12h. These constraints are defined separately for all sets $(R_i,A_i) \in F$. They ensure that from each state in M_i an A_i-state is reachable with respect to the chosen scheduler, as requested in the third condition of Lemma 2. The constraints are satisfied for a set M_i that contains accepting BSCCs of the induced DTMC, which are reachable from all states in M_i. If no element of $S \times A_i$ is contained, no partial order on the states can be defined by the Constraints 6.12f–6.12h. For details of the reachability constraints we again refer to Section 6.1.2.3.

The remaining constraints are defined analogously to the MILP for reachability properties: Constraint 6.12i ensures criticality of the subsystem. Constraints 6.12k and 6.12j force the states of the sets M_i to be included in the subsystem and to have probability 1. Constraint 6.12k assigns probability 0 to all states not in the subsystem and Constraint 6.12l computes the probability of reaching a state in M_i for all remaining states. Since we do not know the target states in advance, we have to allow Constraint 6.12l to be satisfied for target states, too. This is the case due to the expression $\sum_{i=1}^{n} m_u^i$ which is at least 1 if u is a target state.

The last three constraints are again backward reachability constraints, analogous to the reachability constraints for problematic states in the case of reachability properties. They ensure that from each state with a selected action-distribution pair in the subsystem an M_i state is reachable with non-zero probability.

Formula size The number of integer variables in the MILP, its number of constraints, and the number of non-zero coefficients are in $O(n \cdot (n_{\mathcal{M}\otimes\mathcal{A}} + m_{\mathcal{M}\otimes\mathcal{A}}))$, while the number of real variables is in $O(n \cdot n_{\mathcal{M}\otimes\mathcal{A}})$, where n is the number of acceptance pairs of $\mathcal{A}$.

Theorem 10 *The MILP formulation* (6.12a)–(6.12o) *is sound and complete.*

A proof of this theorem can be found in Section 6.3.6.

Remark 29 *The optimizations for DTMCs in Section 6.1.2 can, with the exception of the SCC cuts, be directly transferred to PAs.*

6.3 Correctness proofs

In this section we give formal proofs for all encodings presented in this chapter. Let in the following *Var* be the set of variables of an SMT or MILP problem formulation, see Section 2.6 for details on *valuations* or *assignments*, respectively. Let $\mathcal{D} = (S, s_I, P, L)$ be a DTMC. When we consider reachability properties $\mathbb{P}_{\leq\lambda}(\lozenge T)$, we assume that all BSCCs of $\mathcal{D}$ contain at least one target state from $T \subseteq S$. This is a justified assumption, as by removing all irrelevant states of a DTMC, see Definition 23 on Page 33, only states remain from where a target state is reached with positive probability. For a state set $S' \subseteq S$ with $s_I \in S'$ we use $\mathcal{D}_{S'} = (S', s_I, P', L')$ to denote the restricted DTMC with $P'(s,s') = P(s,s')$ and $L'(s) = L(s)$ for all $s, s' \in S'$.

For some proofs we need additional results. These are stated as *Requirements* and used later on.

6.3.1 SMT-formulation for reachability properties of DTMCs

6.3.1.1 Requirements

First, we need to ensure that within subsystems the linear equation system for computing the reachability probability for reaching target states has a unique solution. This is in question, as subdistributions occur. For the construction of the linear equation system see Section 2.3.1. The following lemma states the uniqueness and correctness of the solution of the corresponding equation system.

Lemma 3 *Let $S' \subseteq S$ with $s_I \in S'$. Then the linear equation system*

$$\forall s \in S' \setminus T. \quad p_s = \sum_{s' \in S' \setminus T} P(s,s') \cdot p_{s'} + \sum_{s' \in T} P(s,s') \tag{6.13a}$$

has a unique satisfying assignment mapping the probability $Pr_s^{\mathcal{D}_{S'}}(\lozenge T)$ to each variable p_s.

Proof 2 Consider the assignment $\nu\colon Var \to \mathbb{R}$ that maps the probability $\nu(p_s) = Pr_s^{\mathcal{D}_{S'}}(\lozenge T)$ of reaching T from s in $\mathcal{D}_{S'}$ to each variable p_s. We observe that ν is a satisfying assignment. The proof can be found in [BK08, Theorem 10.15].

Following the proof idea of [BK08, Theorem 10.19] we show that this satisfying assignment is unique. Suppose that there are two different satisfying assignments $\nu_1, \nu_2\colon Var \to \mathbb{R}$, $\nu_1 \neq \nu_2$, and let $\nu : Var \to \mathbb{R}$ be their absolute difference, i. e., $\nu(p_s) = |\nu_1(p_s) - \nu_2(p_s)| \geq 0$ for each $s \in S'$.

Since the set of states S' is finite, there exists a state $s^* \in S' \setminus T$ such that $\nu(p_{s^*}) \geq \nu(p_s)$ for all $s \in S' \setminus T$. Let s^* be such a "maximal" state. Because $P(s^*,s) \geq 0$ for all $s \in S'$ and $\sum_{s \in S'} P(s^*,s) \leq 1$, the following (in)equations hold due to the definition of ν and the choice of s^*:

$$\begin{aligned}
\nu(p_{s^*}) &= \left|\nu_1(p_{s^*}) - \nu_2(p_{s^*})\right| \\
&= \left|\Big(\sum_{s \in S' \setminus T} P(s^*,s) \cdot \nu_1(p_s) + \sum_{s \in T} P(s^*,s)\Big) - \Big(\sum_{s \in S' \setminus T} P(s^*,s) \cdot \nu_2(p_s) + \sum_{s \in T} P(s^*,s)\Big)\right| \\
&= \left|\sum_{s \in S' \setminus T} P(s^*,s) \cdot (\nu_1(p_s) - \nu_2(p_s))\right| \\
&\leq \sum_{s \in S' \setminus T} P(s^*,s) \cdot \left|\nu_1(p_s) - \nu_2(p_s)\right| \\
&= \sum_{s \in S' \setminus T} P(s^*,s) \cdot \nu(p_s) \\
&\leq \nu(p_{s^*}) \cdot \sum_{s \in S' \setminus T} P(s^*,s) \\
&\leq \nu(p_{s^*}) .
\end{aligned}$$

We conclude that

$$\nu(p_{s^*}) = \sum_{s \in S' \setminus T} P(s^*,s) \cdot \nu(p_s) .$$

Since we have the inequality of ν_1 and ν_2 and the maximal property for s^*, we know that $\nu(p_{s^*}) > 0$. Thereby, the equations $\sum_{s \in S' \setminus T} P(s^*,s) = 1$ and $\nu(p_s) = \nu(p_{s^*})$ must hold for all $s \in S' \setminus T$. By induction it follows that $\sum_{s' \in S' \setminus T} P(s,s') = 1$ for all states $s \in S' \setminus T$ which are reachable from s^* in $\mathscr{D}$. That means, T is not reachable from s^* which contradicts our assumption that S contains no irrelevant states. □

6.3.1.2 Soundness and completeness

Having this result, we are ready to prove the soundness and completeness of the SMT-formulation for reachability properties of DTMCs, see Theorem 4 on Page 126. We start by proving the soundness, stated in the following lemma.

Lemma 4 *The SMT formulation* (6.1a)–(6.1d) *on Page 125 is sound.*

Proof 3 We have to prove that for each satisfying assignment v of the SMT formulation (6.1a)–(6.1d) the restricted DTMC $\mathscr{D}_{S'}$ with $S' = \{s \in S \mid v(x_s) = 1\}$ is an MCS for $\mathscr{D}$ and $\mathbb{P}_{\leq\lambda}(\lozenge T)$ with $v(p_{s_I}) = Pr_{s_I}^{\mathscr{D}_{S'}}(\lozenge T)$.

Let v be a satisfying assignment for Constraints 6.1a–6.1d and let $S' = \{s \in S \mid v(x_s) = 1\}$.

1. We first show that $\mathscr{D}_{S'}$ is a subsystem of $\mathscr{D}$ as in Definition 6 on Page 22. From Constraint 6.1d we can conclude that $v(p_{s_I}) > \lambda \geq 0$. By the satisfaction of Constraints 6.1b–6.1c we have that $v(x_{s_I}) = 1$, i. e., $s_I \in S'$. The remaining conditions for $\mathscr{D}_{S'}$ being a subsystem of $\mathscr{D}$ hold by the definition of $\mathscr{D}_{S'}$.

2. We show that $\mathscr{D}_{S'}$ is critical with the assigned probability $v(p_{s_I}) = Pr_{s_I}^{\mathscr{D}_{S'}}(\lozenge T)$. Constraints 6.1b–6.1c assure that (i) $v(p_s) = 0$ for all $s \in S \setminus S'$ and (ii) $v(p_s) = 1$ for all $s \in S' \cap T$. Therefore, due to the satisfaction of Constraint 6.1c, v is also a satisfying assignment to the constraints

$$\forall s \in S' \setminus T. \quad p_s = \sum_{s' \in S' \setminus T} P'(s,s') \cdot p_{s'} + \sum_{s' \in S' \cap T} P'(s,s')\,. \tag{6.14}$$

   Lemma 3 implies that this satisfying assignment is unique, assigning to each variable p_s for each state $s \in S'$ the probability $Pr_s^{\mathscr{D}_{S'}}(\lozenge T)$. From (6.1d) we conclude that this probability exceeds the probability bound for the initial state, i. e., $v(p_{s_I}) = Pr_{s_I}^{\mathscr{D}_{S'}}(\lozenge T) > \lambda$.

3. It remains to show that $\mathscr{D}_{S'}$ is minimal. Assume the opposite. Then there is some $S'' \subseteq S$ with $|S''| < |S'|$ such that $\mathscr{D}_{S''}$ is an MCS for $\mathscr{D}$ and $\mathbb{P}_{\leq\lambda}(\lozenge T)$. In Constraints (6.1a)–(6.1d) we syntactically replace x_s by 1 if $s \in S''$ and by 0 otherwise. Lemma 3 applied to S'' implies that the constraint system resulting from the above substitution has a unique satisfying assignment. However, for this satisfying assignment the number of positive x_s variables is smaller than for v, which contradicts our assumption that v is a satisfying assignment to the optimization problem. □

Lemma 5 *The SMT formulation* (6.1a)–(6.1d) *is complete.*

Proof 4 We prove that for each MCS for $\mathscr{D}$ and $\psi = \mathbb{P}_{\leq\lambda}(\lozenge T)$ with state set S' there is a satisfying assignment ν of the SMT formulation (6.1a)–(6.1d) with $S' = \{s \in S \mid \nu(x_s) = 1\}$ and $\nu(p_{s_I}) = Pr_{s_I}^{\mathscr{D}_{S'}}(\lozenge T)$.

Assume that $\mathscr{D}$ violates the property $\mathbb{P}_{\leq\lambda}(\lozenge T)$, and assume a DTMC $\mathscr{D}' = (S', I', P', L')$ with $\mathscr{D}' \sqsubseteq \mathscr{D}$ which is an MCS for $\mathscr{D}$ and ψ. Then $\mathscr{D}_{S'}$ is also an MCS for $\mathscr{D}$ and ψ. Note, that $\mathscr{D}_{S'}$ has the same states as $\mathscr{D}'$ but possibly more transitions. We define the assignment $\nu : Var \to \mathbb{R}$ by (i) $\nu(x_s) = 1$ and $\nu(p_s) = Pr_s^{\mathscr{D}_{S'}}(\lozenge T)$ for $s \in S'$ and (ii) $\nu(x_s) = \nu(p_s) = 0$ otherwise. We show that ν satisfies the SMT constraints (6.1a)–(6.1d).

1. By syntactically replacing the x_s variables by their values given by ν, the constraints (6.1b)–(6.1c) reduce to $p_s = 0$ for each $s \notin S'$, $p_s = 1$ for each $s \in S' \cap T$ and

$$\forall s \in S' \setminus T. \quad p_s = \sum_{s' \in S' \setminus T} P(s,s') \cdot p_{s'} + \sum_{s' \in S' \cap T} P(s,s'). \tag{6.15a}$$

   By Lemma 3, we know that ν is the unique satisfying assignment for this constraint system. That means that ν is also a satisfying assignment to (6.1b)–(6.1c).

2. Since $\mathscr{D}_{S'}$ is critical, it holds that $\nu(p_{s_I}) = Pr_{s_I}^{\mathscr{D}_{S'}}(\lozenge T) > \lambda$. Thereby, Constraint 6.1d also holds.

3. If the defined assignment did not minimize the number of states, then there would be another satisfying assignment that evaluates x_s to 1 for a smaller number of states. Due to soundness of the SMT formulation (Lemma 4), there would exist an MCS smaller than $\mathscr{D}_{S'}$, which contradicts the minimality assumption for $\mathscr{D}_{S'}$. □

The correctness of Theorem 4 follows then directly by the correctness of Lemma 4 and Lemma 5.

6.3.2 MILP-formulation for reachability properties of DTMCs

6.3.2.1 Requirements

Before we can prove the correctness for the dedicated MILP formulation, we need the following prerequisite. In the MILP encodings presented in Sections 6.1 and 6.2, we require that the values of the probability variables p_s are assigned *at most* the probability of moving to a direct successor state s' multiplied by the corresponding value $p_{s'}$ of the successor state. In contrast, the model checking equations require equality at this place, see Section 2.3.1. Enforcing only an upper bound is necessary, as we have to assign 0 to p_s if s is not part of the subsystem we want to compute. Therefore, we first have to show that the satisfying assignments for the inequalities are always below the actual reachability probabilities. This is then used to show the soundness and completeness of our MILP encodings.

Lemma 6 *Let $S' \subseteq S$ with $s_I \in S'$. For each satisfying assignment ν of the constraint system*

$$\forall s \in S' \setminus T. \quad p_s \leq \sum_{s' \in S' \setminus T} P(s,s') \cdot p_{s'} + \sum_{s' \in S' \cap T} P(s,s') \tag{6.16a}$$

we have that $\nu(p_s) \leq Pr_s^{\mathscr{D}_{S'}}(\Diamond T)$ for each $s \in S' \setminus T$.

Proof 5 According to Lemma 3, the assignment ν with $\nu(p_s) = Pr_s^{\mathscr{D}_{S'}}(\Diamond T)$ for each $s \in S' \setminus T$ is the unique satisfying assignment fulfilling the constraints (6.16a) with equality.

Now we show that for each satisfying assignment μ of (6.16a) we have that $\mu(p_s) \leq \nu(p_s)$ for each $s \in S' \setminus T$. Assume that the converse is true. Then there exists a satisfying assignment μ^* for Constraint 6.16a such that $\mu^*(p_{s^*}) > \nu(p_{s^*})$ for some $s^* \in S' \setminus T$. Let μ^* be such an assignment and s^* such a state, and let $\varepsilon = \mu^*(p_{s^*})$. Then

$$\text{maximize} \quad \sum_{s \in S' \setminus T} p_s \tag{6.17a}$$

such that

$$\forall s \in S' \setminus T. \quad p_s \leq \sum_{s' \in S' \setminus T} P(s,s') \cdot p_{s'} + \sum_{s' \in S' \cap T} P(s,s') \tag{6.17b}$$

$$p_{s^*} \geq \varepsilon \tag{6.17c}$$

has a satisfying assignment, since μ^* is a satisfying assignment for (6.17b)–(6.17c) and the maximum under all satisfying assignments exists because on the one hand the variable domains are all bounded by closed intervals and on the other hand all involved constraints are non-strict (linear) inequalities. Let $\mu_{\max}$ be a satisfying assignment to (6.17a)–(6.17c).

From (6.17c) we conclude that $\mu_{\max} \neq \nu$. Since ν satisfies all constraints (6.17b) with equality and $\mu_{\max} \neq \nu$, there exists at least one $s_{\max} \in S' \setminus T$ such that:

$$\mu_{\max}(p_{s_{\max}}) \quad < \quad \sum_{s' \in S' \setminus T} P(s_{\max},s') \cdot \mu_{\max}(p_{s'}) + \sum_{s' \in S' \cap T} P(s_{\max},s') \quad =: \quad d\,.$$

Let $s_{\max}$ be such a state. We define the assignment $\mu'_{\max}$ by $\mu'_{\max}(p_{s_{\max}}) = d$ and $\mu'_{\max}(p_s) = \mu_{\max}(p_s)$ for all other states $s \in S' \setminus (T \cup \{s_{\max}\})$. Note that $P(s,s') \geq 0$ and $\mu'_{\max}(p_s) \geq \mu_{\max}(p_s)$ for all $s, s' \in S' \setminus T$, therefore $\mu'_{\max}$ also satisfies (6.17b)–(6.17c):

$$\begin{aligned} \mu'_{\max}(p_s) \quad &= d \\ &= \sum_{s' \in S' \setminus T} P(s_{\max},s') \cdot \mu_{\max}(p_{s'}) + \sum_{s' \in S' \cap T} P(s_{\max},s') \\ &\leq \sum_{s' \in S' \setminus T} P(s_{\max},s') \cdot \mu'_{\max}(p_{s'}) + \sum_{s' \in S' \cap T} P(s_{\max},s') \end{aligned}$$

$$\begin{aligned}
\forall s \in S' \setminus (T \cup \{s_{\max}\}).\quad \mu'_{\max}(p_s) &= \mu_{\max}(p_s) \\
&\leq \sum_{s' \in S' \setminus T} P(s,s') \cdot \mu_{\max}(p_{s'}) + \sum_{s' \in S' \cap T} P(s,s') \\
&\leq \sum_{s' \in S' \setminus T} P(s,s') \cdot \mu'_{\max}(p_{s'}) + \sum_{s' \in S' \cap T} P(s,s')
\end{aligned}$$

$$\begin{aligned}
\mu'_{\max}(p_{s^*}) &\geq \mu_{\max}(p_{s^*}) \\
&\geq \varepsilon \quad (= \mu^*(p_{s^*}))\,.
\end{aligned}$$

However, $\mu'_{\max}$ yields a larger sum over the p_s variable values than $\mu_{\max}$, which contradicts the fact that $\mu_{\max}$ is optimal with respect to (6.17a). That means, our assumption about the existence of μ^* was wrong, which proves the statement. □

6.3.2.2 Soundness and completeness

Now we prove soundness and completeness of the MILP encoding for computing MCSs of $\mathscr{D}$ for $\mathbb{P}_{\leq\lambda}(\lozenge T)$ see Constraints 6.2a–6.2e on Page 127.

Lemma 7 *The MILP formulation* (6.2a)–(6.2e) *is sound.*

Proof 6 We show that for each satisfying assignment ν of the MILP formulation (6.2a)–(6.2e) the DTMC $\mathscr{D}_{S'}$ with $S' = \{s \in S \,|\, \nu(x_s) = 1\}$ is an MCS of $\mathscr{D}$ for $\mathbb{P}_{\leq\lambda}(\lozenge T)$ with a maximal probability $\nu(p_{s_I}) = Pr_{s_I}^{\mathscr{D}_{S'}}(\lozenge T)$ to reach T from the initial state s_I under all MCSs.

Let ν be a satisfying assignment for (6.2a)–(6.2e) and $S' = \{s \in S \mid \nu(x_s) = 1\}$.

1. We show that $\mathscr{D}_{S'}$ is a subsystem of $\mathscr{D}$. From Constraint 6.2e we conclude $0 \leq \lambda < \nu(p_{s_I})$, and therefore by the satisfaction of (6.2b)–(6.2c) we have that $\nu(x_{s_I}) = 1$, i. e., $s_I \in S'$. The remaining conditions for $\mathscr{D}_{S'}$ being a subsystem of $\mathscr{D}$ hold by the definition of $\mathscr{D}_{S'}$.

2. We show that $\nu(p_{s_I}) = Pr_{s_I}^{\mathscr{D}_{S'}}(\lozenge T)$. The constraints (6.2b)–(6.2d) assure that (i) $\nu(p_s) = 0$ for all $s \in S \setminus S'$ and (ii) $\nu(p_s) = 1$ for all $s \in S' \cap T$. Therefore, due to the satisfaction of (6.2d), ν is a satisfying assignment to

$$\forall s \in S' \setminus T.\quad p_s \leq \sum_{s' \in S' \setminus T} P'(s,s') \cdot p_{s'} + \sum_{s' \in S' \cap T} P'(s,s')\,. \tag{6.18}$$

   Lemma 6 implies $\nu(p_{s_I}) \leq Pr_{s_I}^{\mathscr{D}_{S'}}(\lozenge T)$, and Lemma 3 implies that there is also a satisfying assignment mapping $Pr_{s_I}^{\mathscr{D}_{S'}}(\lozenge T)$ to p_{s_I} and thereby satisfying the inequations with equality. Since ν is a satisfying assignment maximizing p_{s_I} in (6.2a), $\nu(p_{s_I}) = Pr_{s_I}^{\mathscr{D}_{S'}}(\lozenge T)$ has to hold.

3. We show that $\mathscr{D}_{S'}$ is critical. In (2) we have shown that $\nu(p_{s_I}) = Pr_{s_I}^{\mathscr{D}_{S'}}(\lozenge T)$. Together with $0 \leq \lambda < \nu(p_{s_I})$ ensured by Constraint 1 we get $\lambda < Pr_{s_I}^{\mathscr{D}_{S'}}(\lozenge T)$.

4. We show that $\mathscr{D}_{S'}$ is minimal. Assume the opposite. Then there is some $S'' \subseteq S$ with $|S''| < |S'|$ such that $\mathscr{D}_{S''}$ is an MCS for $\mathscr{D}$ and the property $Pr_{s_I}^{\mathscr{D}_{S'}}(\lozenge T)$. We define the assignment $\mu \colon Var \to \mathbb{R}$ by (i) $\mu(x_s) = 1$ and $\mu(p_s) = Pr_{s_I}^{\mathscr{D}_{S''}}(\lozenge T)$ for $s \in S''$ and (ii) $\mu(x_s) = \mu(p_s) = 0$ otherwise. Lemma 3 applied to S'' implies that μ satisfies (6.2b)–(6.2d) and thereby satisfying all constraints with equality. However, it holds that $\sum_{s \in S} \mu(x_s) < \sum_{s \in S} \nu(x_s)$, what contradicts our assumption that ν is optimal with respect to (6.2a).

5. It remains to show that the probability to reach T from s_I in $\mathscr{D}_{S'}$ is maximal under all MCSs. This proof is analogous to the previous item. Assume the opposite. Then there is some set of states $S'' \subseteq S$ such that $\mathscr{D}_{S''}$ is an MCS for $\mathscr{D}$ and $\mathbb{P}_{\leq \lambda}(\lozenge T)$ with $Pr_{s_I}^{\mathscr{D}_{S'}}(\lozenge T) < Pr_{s_I}^{\mathscr{D}_{S''}}(\lozenge T)$. We define μ as above. Again, Lemma 3 implies for S'' that μ satisfies (6.2b)–(6.2d), satisfying all constraints with equality. Since $\mathscr{D}_{S'}$ and $\mathscr{D}_{S''}$ are both minimal, $\sum_{s \in S} \nu(x_s) = \sum_{s \in S} \mu(x_s)$. By (2) we know that $\nu(p_{s_I}) = Pr_{s_I}^{\mathscr{D}_{S'}}(\lozenge T)$, i. e.,

$$\nu(p_{s_I}) = Pr_{s_I}^{\mathscr{D}_{S'}}(\lozenge T) < Pr_{s_I}^{\mathscr{D}_{S''}}(\lozenge T) = \mu(p_{s_I}) .$$

Thus $-\frac{1}{2}\mu(p_{s_I}) + \sum_{s \in S} \mu(x_s) < -\frac{1}{2}\nu(p_{s_I}) + \sum_{s \in S} \nu(x_s)$ holds, contradicting the optimality of ν. □

Lemma 8 *The MILP formulation* (6.2a)–(6.2e) *is complete.*

Proof 7 Assume that $\mathscr{D}$ violates the property $\psi = \mathbb{P}_{\leq \lambda}(\lozenge T)$, and assume a DTMC $\mathscr{D}' \sqsubseteq \mathscr{D}$ with state set S' that is an MCS for $\mathscr{D}$ and ψ such that the probability to reach T from s_I in $\mathscr{D}'$ is maximal under all MCSs. We show that there is a satisfying assignment ν of the MILP formulation (6.2a)–(6.2e) with $S' = \{s \in S \mid \nu(x_s) = 1\}$ and $\nu(p_{s_I}) = Pr_{s_I}^{\mathscr{D}_{S'}}(\lozenge T)$.

Note again that $\mathscr{D}_{S'}$ has the same state set but possibly more transitions than $\mathscr{D}'$, therefore $\mathscr{D}_{S'}$ is also an MCS for $\mathscr{D}$ and the given property with $Pr_{s_I}^{\mathscr{D}'}(\lozenge T) \leq Pr_{s_I}^{\mathscr{D}_{S'}}(\lozenge T)$. We define the assignment $\nu \colon Var \to \mathbb{R}$ by (i) $\nu(x_s) = 1$ and $\nu(p_s) = Pr_{s_I}^{\mathscr{D}_{S'}}(\lozenge T)$ for $s \in S'$ and (ii) $\nu(x_s) = \nu(p_s) = 0$ otherwise. We show step by step that ν satisfies the MILP constraints (6.2a)–(6.2e).

1. We show that ν satisfies (6.2b)–(6.2d). By syntactically replacing x_s by the assignment $\nu(x_s)$ for each $s \in S$, the constraints (6.2b)–(6.2d) reduce to $p_s = 0$ for each $s \in S \setminus S'$, $p_s = 1$ for each $s \in S' \cap T$, and it holds that

$$\forall s \in S' \setminus T. \quad p_s \leq \sum_{s' \in S' \setminus T} P(s,s') \cdot p_{s'} + \sum_{s' \in S' \cap T} P(s,s') . \tag{6.19a}$$

   By Lemma 3, ν is a satisfying assignment for this constraint system.

2. Since $\mathscr{D}_{S'}$ is critical, $\nu(p_{s_I}) = Pr_{s_I}^{\mathscr{D}_{S'}}(\lozenge T) > \lambda$. Therefore, (6.2e) also holds.

3. We show that ν is optimal with respect to (6.2a).

   If ν did not minimize $\sum_{s \in S} x_s$, then there would be another satisfying assignment that evaluates x_s to 1 for a lesser number of states. Due to the soundness shown in Lemma 7, there would exist an MCS that is smaller than $\mathscr{D}_{S'}$, which contradicts the minimality assumption for the MCS $\mathscr{D}_{S'}$.

   Similarly, if ν did not maximize p_{s_I}, there would be another satisfying assignment selecting the same (minimal) number of states but assigning a larger value to p_{s_I} than ν. Again, by soundness (Lemma 7), there would exist an MCS in that the probability to reach T from the initial state is larger than in $\mathscr{D}_{S'}$, which contradicts the assumption that $\mathscr{D}_{S'}$ maximizes the probability to reach a state in T from s_I under all MCSs. □

The correctness of Theorem 5 follows directly by Lemma 7 and by Lemma 8.

6.3.3 Optimizations

Next we prove that the following optimizations are optional for both the SMT and MILP formulations for reachability properties of DTMCs. All optimizations are redundant in a sense that they might be be helpful to speed up the solution process but they do not modify the set of optimal satisfying assignments. This is basically proved by showing that any satisfying assignment for either the SMT or the MILP encoding are also satisfying assignments for the optimizations.

6.3.3.1 Forward/backward constraints

Recall the forward and backwards cuts, given as an MILP encoding by Constraints 6.3a and 6.3b on Page 129.

$$\forall s \in S \setminus T. \quad -x_s + \sum_{s' \in succ(s) \setminus \{s\}} x_{s'} \geq 0 \tag{6.20a}$$

$$\forall s \in S \setminus \{s_I\}. \quad -x_s + \sum_{s' \in pred(s) \setminus \{s\}} x_{s'} \geq 0\ . \tag{6.20b}$$

Lemma 9 *The forward and backward cuts are satisfied by any satisfying assignment of either the SMT formulation* (6.1a)–(6.1d) *on Page 125 or the MILP formulation* (6.2a)–(6.2e) *on Page 127.*

Proof 8 Let ν be an arbitrary optimal satisfying assignment of either the SMT or the MILP formulation and let $S' = \{s \in S \mid \nu(x_s) = 1\}$. For states $s \in S \setminus S'$ it holds that $\nu(x_s) = 0$; for states $s \in S$ we have $\nu(x_s) \geq 0$.

Assume that Constraint 6.20a is violated by ν for a state $s \in S' \setminus T$, i. e., $\nu(x_s) = 1$, but $\nu(x_{s'}) = 0$ for all $s' \in succ_{\mathscr{D}}(s) \setminus \{s\}$. Then we have

$$\nu(p_s) \le \sum_{s'\in succ_{\mathscr{D}}(s)} P(s,s')\cdot\nu(p_{s'}) = \left(\sum_{s'\in succ_{\mathscr{D}}(s')\setminus\{s\}} P(s,s')\cdot 0\right) + P(s,s)\cdot\nu(p_s)$$

since $\nu(p_{s'}) = 0$ for all $s' \in S$ with $\nu(x_{s'}) = 0$. As we assumed the DTMC $\mathscr{D}$ to have only states that are relevant for the set of target states T, which implies that $P(s,s) < 1$, the only solution is $\nu(p_s) = 0$. Therefore state s can be removed from the MCS $\mathscr{D}_{S'}$ without altering the reachability probability of the initial state. This contradicts the minimality of $\mathscr{D}_{S'}$.

Assume now that Constraint 6.20b is violated for a state $s \in S' \setminus \{s_I\}$, i. e., $\nu(x_s) = 1$, but $\nu(x_{s'}) = 0$ for all $s' \in pred_{\mathscr{D}}(s) \setminus \{s\}$. Then s is not reachable from s_I in $\mathscr{D}_{S'}$, therefore $\mathscr{D}_{S'\setminus\{s\}}$ is a subsystem which is also critical and has the same probability to reach T from s_I as $\mathscr{D}_{S'}$. In contrast, it has less states, which is a contraction to $\mathscr{D}_{S'}$ being an MCS. □

6.3.3.2 SCC constraints

Recall the SCC cuts given by Constraints 6.4a and 6.4b on Page 130. Let again $\mathscr{S}$ denote the set of all SCCs of the underlying graph of $\mathscr{D}$.

$$\forall S' \in \{S'' \in \mathscr{S} \mid S'' \cap \{s_I\} = \emptyset\}.\, \forall s \in S' \setminus Inp(S').\quad x_s \le \sum_{s'\in Inp(S')} x_{s'} \tag{6.21a}$$

$$\forall S' \in \{S'' \in \mathscr{S} \mid S'' \cap T = \emptyset\}.\, \forall s \in S'.\quad x_s \le \sum_{s'\in Out(S')} x_{s'}\ . \tag{6.21b}$$

Lemma 10 *The input and output SCC cuts are satisfied by any optimal satisfying assignment of either the SMT formulation* (6.1a)–(6.1d) *or the MILP formulation* (6.2a)–(6.2e).

Proof 9 Let ν be an arbitrary optimal satisfying assignment of either the SMT or the MILP formulation and let $S' = \{s \in S \mid \nu(x_s) = 1\}$.

Assume an SCC $C \subseteq S \setminus \{s_I\}$ which violates Constraint 6.21a. All paths in $\mathscr{D}$ from s_I to T going through C contain a state from $Inp(C)$. If $S' \cap Inp(C) = \emptyset$ then there is no path in $\mathscr{D}_{S'}$ from s_I to T containing a state from C. Therefore all states in $C \cap S' \neq \emptyset$ can be removed from $\mathscr{D}_{S'}$ without alternating the probability of s_I, which contradicts the minimality of $\mathscr{D}_{S'}$.

Now assume that (6.21b) is violated. With the same argument as before we can show that all states in $C \cap S' \neq \emptyset$ are irrelevant in $\mathscr{D}_{S'}$, contradicting the minimality assumption. □

6.3.3.3 Forward reachability constraints

Recall the forward reachability cuts originally given in Constraints 6.6a–6.6c. Consider the forward reachability constraints with $x_s \in \{0,1\} \subseteq \mathbb{Z}$, $t^{\rightarrow}_{s,s'} \in \{0,1\} \subseteq \mathbb{Z}$ and $r^{\rightarrow}_s \in [0,1] \subseteq \mathbb{R}$ for all $s,s' \in S$:

$$\forall s' \in S \setminus \{s_I\}.\, \forall s \in pred(s').\quad t^{\rightarrow}_{s,s'} \le x_s \tag{6.22a}$$

$$\forall s' \in S \setminus \{s_I\}.\ \forall s \in pred(s').\quad r^{\rightarrow}_{s} < r^{\rightarrow}_{s'} + (1 - t^{\rightarrow}_{s,s'}) \tag{6.22b}$$

$$\forall s' \in S \setminus \{s_I\}.\quad \sum_{s \in pred(s')} t^{\rightarrow}_{s,s'} = x_{s'}\ . \tag{6.22c}$$

The following lemma states the redundancy of these constraints.

Lemma 11 *For each optimal satisfying assignment ν of either the SMT formulation* (6.1a)–(6.1d) *or the MILP formulation* (6.2a)–(6.2e) *there exists an extending satisfying assignment μ of Constraints 6.22a–6.22c with $\nu(v) = \mu(v)$ for all $v \in \{x_s, p_s \,|\, s \in S\}$.*

Proof 10 Let ν be an arbitrary optimal satisfying assignment of either the SMT or the MILP formulation and let $S' = \{s \in S \mid \nu(x_s) = 1\}$. By the soundness of the SMT and MILP formulations (Theorems 4 and 7) we know that $\mathscr{D}_{S'}$ is an MCS for $\mathscr{D}$ and $\mathbb{P}_{\leq\lambda}(\Diamond T)$.

We consider a tree which contains for each state $s \in S'$ one shortest path (in terms of the number of states) from s_I to s. (Such a tree exists, since minimality of the MCS $\mathscr{D}_{S'}$ implies that all states in S' are reachable from s_I in $\mathscr{D}_{S'}$.) We define a function $f : S' \setminus \{s_I\} \rightarrow S'$ by assigning to each state $s \in S' \setminus \{s_I\}$ the predecessor state of s in this tree. We fix the assignment μ by

- $\nu(v) = \mu(v)$ for all $v \in \{x_s, p_s \mid s \in S\}$,
- for all $s, s' \in S$, $\mu(t^{\rightarrow}_{s,s'}) = 1$ if $s' \in S' \setminus \{s_I\}$ and $s = f(s')$, and $\mu(t^{\rightarrow}_{s,s'}) = 0$ otherwise, and
- for all $s \in S$ we have that $\mu(r^{\rightarrow}_{s}) = n/d$ where n is the length of a shortest path from s_I to s in $\mathscr{D}_{S'}$, and d is the maximum of the lengths of all shortest paths from s_I to any state in $\mathscr{D}_{S'}$.

It directly follows that μ satisfies the constraint system (6.22a)–(6.22c). □

To show that these constraints fulfill their purpose, we consider them in isolation. Then, for all solutions for (6.22a)–(6.22c), all states in the selected subsystem are reachable from the initial state.

Lemma 12 *Let ν be a satisfying assignment of the forward reachability constraints* (6.22a)–(6.22c). *Then for all $s' \in S$, if $\nu(x_{s'}) = 1$ then there is a path $s_0 s_1 \ldots s_n = s'$ from $s_0 = s_I$ to s' with $\nu(x_{s_i}) = 1$ for all $0 \leq i \leq n$.*

Proof 11 Constraint 6.22c enforces that each state $s' \in S \setminus \{s_I\}$ with $\nu(x_{s'}) = 1$ has a predecessor state $s \in pred(s')$ with $\nu(t^{\rightarrow}_{s,s'}) = 1$. Constraint 6.22a ensures that for this predecessor state $\nu(x_s) = 1$ holds. Constraint 6.22b finally ensures that $\nu(r^{\rightarrow}_{s}) < \nu(r^{\rightarrow}_{s'})$.

Assume there is a state $u_0 \in S \setminus \{s_I\}$ such that the statement of the lemma is false. Then we can construct an infinite sequence $u_0 u_1 u_2 \ldots$ such that $u_{i+1} \in pred(u_i)$, $\nu(x_{u_i}) = 1$, $\nu(t^{\rightarrow}_{u_{i+1},u_i}) = 1$, and $\nu(r^{\leftarrow}_{u_{i+1}}) < \nu(r^{\leftarrow}_{u_i})$ for all $i \geq 0$.

Since S is finite there are $i < k$ with $u_i = u_k$. However $\nu(r^{\leftarrow}_{u_k}) < \nu(r^{\leftarrow}_{u_i})$ holds, which is a contradiction. Therefore our assumption was wrong and the lemma is valid. □

6.3.3.4 Backward reachability constraints

Consider the backward reachability constraints with $x_s \in \{0,1\} \subseteq \mathbb{Z}$, $t^{\leftarrow}_{s,s'} \in \{0,1\} \subseteq \mathbb{Z}$ and $r^{\leftarrow}_s \in [0,1] \subseteq \mathbb{R}$ for all $s, s' \in S$:

$$\forall s \in S \setminus T.\, \forall s' \in succ(s). \quad t^{\leftarrow}_{s,s'} \leq x_{s'} \tag{6.23a}$$

$$\forall s \in S \setminus T.\, \forall s' \in succ(s). \quad r^{\leftarrow}_s < r^{\leftarrow}_{s'} + (1 - t^{\leftarrow}_{s,s'}) \tag{6.23b}$$

$$\forall s \in S \setminus T. \quad \sum_{s' \in succ(s)} t^{\leftarrow}_{s,s'} = x_s \ . \tag{6.23c}$$

Lemma 13 *For each optimal satisfying assignment ν of either the SMT formulation* (6.1a)–(6.1d) *or the MILP formulation* (6.2a)–(6.2e) *there exists an extending satisfying assignment μ of* (6.8a)–(6.8c) *with $\nu(v) = \mu(v)$ for all $v \in \{x_s, p_s \,|\, s \in S\}$.*

Proof 12 Let ν be an arbitrary optimal satisfying assignment of either the SMT or the MILP formulation and let $S' = \{s \in S \,|\, \nu(x_s) = 1\}$. By the soundness of the SMT and MILP formulations (Theorems 4 and 7) we now that $\mathscr{D}_{S'}$ is an MCS for $\mathscr{D}$ and $\mathbb{P}_{\leq\lambda}(\Diamond T)$.

We consider a set Π of paths of $\mathscr{D}_{S'}$ such that Π is postfix-closed and it contains for each state $s \in S'$ exactly one shortest path from s to T in $\mathscr{D}_{S'}$. (Such a set exists, since minimality of the MCS $\mathscr{D}_{S'}$ implies that T can be reached from all states in $\mathscr{D}_{S'}$.) We define the function $f : S' \setminus T \rightarrow S'$ by assigning to each state $s \in S' \setminus T$ the unique successor state of s on the shortest path from s to T in Π. We fix the assignment μ by

- $\nu(v) = \mu(v)$ for all $v \in \{x_s, p_s \,|\, s \in S\}$,
- for all $s, s' \in S$, $\mu(t^{\leftarrow}_{s,s'}) = 1$ if $s \in S' \setminus T$ and $s' = f(s)$, and $\mu(t^{\leftarrow}_{s,s'}) = 0$ otherwise, and
- for all $s \in S$, $\mu(r^{\leftarrow}_s) = 1 - n/d$ where n is the length of a shortest path from s to T in $\mathscr{D}_{S'}$, and d is the maximum of the lengths of all shortest paths from any state to T in $\mathscr{D}_{S'}$.

It directly follows that μ satisfies the constraint system (6.23a)–(6.23c). □

We show that for all solutions for (6.23a)–(6.23c) T is reachable from all states in the selected subsystem.

Lemma 14 *Let ν be a satisfying assignment of the backward reachability constraints* (6.23a)–(6.23c). *Then for all $s \in S$, $\nu(x_s) = 1$ implies that there is a path $s = s_0 s_1 \ldots s_n$ from s to a state $s_n \in T$ with $\nu(x_{s_i}) = 1$ for all $0 \leq i \leq n$.*

Proof 13 Constraint 6.23c enforces that each state $s \in S \setminus T$ with $\nu(x_s) = 1$ has a successor state $s' \in succ(s)$ with $t^{\leftarrow}_{s,s'} = 1$. Constraint 6.23a ensures that for this successor state $\nu(x_{s'}) = 1$ holds. Constraint 6.23b finally ensures that $\nu(r^{\leftarrow}_s) < \nu(r^{\leftarrow}_{s'})$.

Assume now that there is a state $u_0 \in S \setminus T$ such that the statement of the lemma is false. Then we can construct an infinite path $u_0 u_1 u_2 \dots$ such that $u_{i+1} \in succ(u_i)$, $\nu(x_{u_i}) = 1$, $\nu(t^{\leftarrow}_{u_i,u_{i+1}}) = 1$, and $\nu(r^{\leftarrow}_{u_i}) < \nu(r^{\leftarrow}_{u_{i+1}})$ for all $i \geq 0$.

Since S is finite there are $i < k$ with $u_i = u_k$. However, $\nu(r^{\leftarrow}_{u_i}) < \nu(r^{\leftarrow}_{u_k})$ leads to a contradiction. Therefore our assumption was wrong and the lemma is valid. □

Recall now finally Theorem 6 on Page 134, whose proof we give below.

Proof 14 Since each satisfying assignment for the SMT or MILP formulation *with* optimization constraints is also a satisfying assignment for the SMT or MILP formulation *without* optimization constraints, soundness follows directly from the soundness of the SMT and MILP formulations (Theorems 4 and 7).

For completeness assume an MCS. By the completeness results for the SMT and MILP formulations (Theorems 5 and 8) we know that they have a satisfying assignment inducing the given MCS. Above we have shown that this satisfying assignment also satisfies the optimization constraints. □

6.3.4 MILP-formulation for ω-regular properties of DTMCs

6.3.4.1 Requirements

Let in this section $\mathscr{D} = (S, s_I, P, L)$ be a DTMC, $\psi = \mathbb{P}_{\leq\lambda}(\mathscr{L})$ an ω-regular property, which is violated by $\mathscr{D}$, and $\mathscr{A} = (Q, q_I, 2^{AP}, \delta, F)$ a DRA with $\mathscr{L}(\mathscr{A}) = \mathscr{L}$. We consider the product $\mathscr{D} \otimes \mathscr{A}$ of the DTMC $\mathscr{D}$ and the DRA $\mathscr{A}$ as in Definition 29 on Page 38 with distribution function P', and assume that all irrelevant states have been removed, see Definition 23 on Page 33. To simplify notation we use $U = S \times Q$ and $u = (s, q)$, $u' = (s', q')$, etc. as typical elements from U. Let $\mathscr{B}$ be the set of accepting BSCCs of $\mathscr{D} \otimes \mathscr{A}$ and $T = \bigcup_{B \in \mathscr{B}} B$.

we make use of the following fact regarding accepting BSCCs for restricted DTMCs.

Lemma 15 *Let $S' \subseteq S$ with $s_I \in S'$ and let $\mathscr{B}_{S'}$ be the set of all accepting BSCCs of $\mathscr{D}_{S'} \otimes \mathscr{A}$. Then $\mathscr{B}_{S'} = \{B \in \mathscr{B} \mid B \subseteq S'\}$.*

Proof 15 Let S' and $\mathscr{B}_{S'}$ be as above. It is easy to see that each accepting BSCC $B \in \mathscr{B}$ of $\mathscr{D}_S \otimes \mathscr{A}$ with $B \subseteq S'$ is also an accepting BSCC of $\mathscr{D}_{S'} \otimes \mathscr{A}$.

For the other direction fix some $B \in \mathscr{B}_{S'}$. By definition, B is an accepting BSCC in $\mathscr{D}_{S'} \otimes \mathscr{A}$, i. e., B is strongly connected, maximal, bottom and accepting in $\mathscr{D}_{S'} \otimes \mathscr{A}$. We show that B is an accepting BSCC also in $\mathscr{D}_S \otimes \mathscr{A}$, i. e., B is strongly connected, maximal, bottom and accepting in $\mathscr{D}_S \otimes \mathscr{A}$.

- *Strongly connected:* Since $\mathscr{D}_{S'}$ is a subsystem of $\mathscr{D}$, also $\mathscr{D}_{S'} \otimes \mathscr{A}$ is a subsystem of $\mathscr{D}_S \otimes \mathscr{A}$, therefore B is also a strongly connected state set in $\mathscr{D}_S \otimes \mathscr{A}$.

- *Maximal:* Since B is a bottom SCC in $\mathscr{D}_{S'} \otimes \mathscr{A}$, we have that $\sum_{u' \in B} P'(u,u') = 1$ for all $u \in B$. Thus B cannot be extended to any larger strongly connected state set in $\mathscr{D}_S \otimes \mathscr{A}$, i. e., B is maximal in $\mathscr{D}_S \otimes \mathscr{A}$.
- *Bottom:* The bottom property of B in $\mathscr{D}_{S'} \otimes \mathscr{A}$ directly implies its bottom property in $\mathscr{D}_S \otimes \mathscr{A}$.
- *Accepting:* As B is accepting in $\mathscr{D}_{S'} \otimes \mathscr{A}$, it is also accepting in $\mathscr{D}_S \otimes \mathscr{A}$.

Thus, B is an accepting BSCC in $\mathscr{D}_S \otimes \mathscr{A}$. □

6.3.4.2 Soundness and completeness

Recall the MILP-formulation for an MCS of $\mathscr{D}$ for $\psi = \mathbb{P}_{\leq\lambda}(\mathscr{L})$, see Section 6.1.3, which reads as follows:

$$\text{minimize} \quad -\frac{1}{2} p_{sq_I} + \sum_{s \in S} x_s \tag{6.24a}$$

$$\text{such that}$$

$$p_{sq_I} > \lambda \tag{6.24b}$$

$$\forall B \in \mathscr{B}\ \forall (s,q) \in B. \quad p_{sq} = x_B \tag{6.24c}$$

$$\forall B \in \mathscr{B}\ \forall (s,q) \in B. \quad x_s \geq x_B \tag{6.24d}$$

$$\forall (s,q) \in S_{\mathscr{D}\otimes\mathscr{A}} \setminus T. \quad p_{sq} \leq x_s \tag{6.24e}$$

$$\forall (s,q) \in S_{\mathscr{D}\otimes\mathscr{A}} \setminus T. \quad p_{sq} \leq \sum_{(s',q') \in succ_{\mathscr{D}\otimes\mathscr{A}}((s,q))} P\big((s,q),(s',q')\big) \cdot p_{s'q'} \tag{6.24f}$$

Lemma 16 *The MILP formulation* (6.24a)–(6.24f) *is sound.*

Proof 16 We show that for each satisfying assignment ν of the MILP constraints (6.24a)–(6.24f) there is a corresponding MCS $\mathscr{D}' \subseteq \mathscr{D}$ for ψ with state space $S' = \{s \in S \mid \nu(x_s) = 1\}$ and a maximal probability $\nu(p_{sq_I}) = Pr_{s_I}^{\mathscr{D}_{S'}}(\mathscr{L})$ to satisfy $\mathscr{L}$ under all MCSs.
Let ν be a satisfying assignment of the MILP constraints (6.24a)–(6.24f) and $S' = \{s \in S \mid \nu(x_s) = 1\}$. Let furthermore again $\mathscr{B}'$ be the set of all accepting BSCCs of $\mathscr{D}_{S'} \otimes \mathscr{A}$. Lemma 15 states that each accepting BSCC $B' \in \mathscr{B}'$ is also an accepting BSCC of $\mathscr{D}_S \otimes \mathscr{A}$, i. e., $B' \in \mathscr{B}$. For simplicity, in the following we also write x_B to denote $x_{B'}$ with $B' = B \in \mathscr{B}$, and define $B_\nu = \{u \in B \mid B' \in \mathscr{B}' \wedge \nu(x_{B'}) = 1\} \subseteq T$ to be the set of all states in selected accepting BSCCs of $\mathscr{D}_{S'} \otimes \mathscr{A}$.

1. We show that $\mathscr{D}_{S'}$ is a subsystem of $\mathscr{D}$. From Constraint 6.24b we imply $\nu(p_{sq_I}) > \lambda \geq 0$. Using (6.24c)–(6.24e) we get that $\nu(x_{s_I}) = 1$. The other conditions for $\mathscr{D}_{S'}$ being a subsystem of $\mathscr{D}$ are straightforward by the definition of $\mathscr{D}_{S'}$.

2. Now we show that no state from $T \setminus B_\nu$ is reachable from sq_I in $\mathscr{D}_{S'} \otimes \mathscr{A}$. Assume the opposite. Then there is an accepting BSCC B' of $\mathscr{D}_{S'} \otimes \mathscr{A}$ that is reachable from sq_I in $\mathscr{D}_{S'} \otimes \mathscr{A}$ such that $\nu(x_{B'}) = 0$. Assume a shortest path $\pi = u_0 \dots u_m$ from $sq_I = u_0$ to $B' \ni u_m$ in $\mathscr{D}_{S'} \otimes \mathscr{A}$, and define $c_i = \Pi_{l=i}^{m-1} P'(u_l, u_{l+1}) > 0$ for $i = 0, \dots, m$ (with $c_m = 1$) to be the probabilities of the postfixes of π starting at position i. We define an assignment μ by

   - $\mu(x_s) = \nu(x_s)$ for all $s \in S$,
   - $\mu(x_{B'}) = 1$ and $\mu(x_B) = \nu(x_B)$ for all $B \in \mathscr{B}$ with $B \neq B'$, and
   - $\mu(p_u) = \begin{cases} 1, & \text{for } u \in B', \\ \nu(p_{u_i}) + c_i, & \text{for } u = u_i,\ i = 0, \dots, m-1, \\ \nu(p_u), & \text{otherwise.} \end{cases}$

   Note that $\mu(p_u) \geq \nu(p_u)$ for all $u \in U$. We show that μ is a satisfying assignment to (6.24b)–(6.24e). The only interesting case is (6.24f). For those states from $U \setminus T$ that are not on the path π, the left-hand-side evaluates equal under μ and ν, and the right-hand-side evaluates under μ to a value that is at least as large as under ν. For states u_i, $i = 0, \dots, m-1$, on the path π we have the following relations:

$$\begin{array}{lll} & \mu(p_{u_i}) & \\ = & c_i + \nu(p_{u_i}) & \\ = & P'(u_i, u_{i+1}) \cdot c_{i+1} + \nu(p_{u_i}) & \\ \leq & P'(u_i, u_{i+1}) \cdot c_{i+1} + \sum_{u' \in U} P'(u_i, u') \cdot \nu(p_{u'}) & \\ = & P'(u_i, u_{i+1}) \cdot c_{i+1} + P'(u_i, u_{i+1}) \cdot \nu(p_{u_{i+1}}) & + \ \sum_{u_{i+1} \neq u' \in U} P'(u_i, u') \cdot \nu(p_{u'}) \\ = & P'(u_i, u_{i+1}) \cdot (c_{i+1} + \nu(p_{u_{i+1}})) & + \ \sum_{u_{i+1} \neq u' \in U} P'(u_i, u') \cdot \nu(p_{u'}) \\ = & P'(u_i, u_{i+1}) \cdot \mu(p_{u_{i+1}}) & + \ \sum_{u_{i+1} \neq u' \in U} P'(u_i, u') \cdot \nu(p_{u'}) \\ \leq & P'(u_i, u_{i+1}) \cdot \mu(p_{u_{i+1}}) & + \ \sum_{u_{i+1} \neq u' \in U} P'(u_i, u') \cdot \mu(p_{u'}) \\ = & \sum_{u' \in U} P'(u_i, u') \cdot \mu(p_{u'}) & \end{array}$$

   Thus (6.24f) is satisfied for all states from $U \setminus T$. However, having $\mu(p_{sq_I}) = \nu(p_{sq_I}) + c_0 > \nu(p_{sq_I})$ and $\mu(x_s) = \nu(x_s)$ for all $s \in S$, the objective function would have a smaller value for μ than for ν, which contradicts the optimality of ν.

3. Now we are able to show that $\nu(p_{sq_I}) = Pr_{s_I}^{\mathscr{D}_{S'}}(\mathscr{L})$ holds. Let $\mathscr{D}'$ be $\mathscr{D}_{S'} \otimes \mathscr{A}$ without the unreachable states $T \setminus B_\nu$ and their connected transitions and let $A = (S' \times Q) \setminus (T \setminus B_\nu)$ denote its state space. By Theorem 1 and the above item (2) we have that

$$Pr_{s_I}^{\mathscr{D}_{S'}}(\mathscr{L}) = Pr_{sq_I}^{\mathscr{D}_{S'} \otimes \mathscr{A}}(\Diamond T) = Pr_{sq_I}^{\mathscr{D}'}(\Diamond B_\nu) .$$

   By (6.24c)–(6.24e) it holds that $\nu(p_u) = 0$ for all $u \in U \setminus A$. Since ν satisfies (6.24f), we have for all states $u \in A$ of $\mathscr{D}'$:

$$\begin{aligned} \nu(p_u) \leq & \sum_{u' \in U} P'(u,u') \cdot \nu(p_{u'}) \\ = & \sum_{u' \in A} P'(u,u') \cdot \nu(p_{u'}) \\ = & \sum_{u' \in A \setminus B_\nu} P'(u,u') \cdot \nu(p_{u'}) + \sum_{u' \in A \cap B_\nu} P'(u,u') \, . \end{aligned}$$

Using Lemma 6 we get that $\nu(p_u) \leq Pr_u^{\mathscr{D}'}(\Diamond B_\nu)$ for each $u \in A$. Lemma 3 furthermore states that there is a satisfying assignment μ with $\mu(p_u) = Pr_u^{\mathscr{D}'}(\Diamond B_\nu)$ for each $u \in A$. Since ν minimizes the objective function (6.24a), it maximizes the value of p_{sq_I}, therefore $\nu(p_{sq_I}) = Pr_{sq_I}^{\mathscr{D}'}(\Diamond B_\nu)$.

4. Now it is easy to see that $\mathscr{D}_{S'}$ is critical: Item (1) above showed that $\nu(p_{sq_I}) > \lambda$ and item (3) showed that $\nu(p_{sq_I}) = Pr_{s_I}^{\mathscr{D}_{S'}}(\mathscr{L})$, together implying $Pr_{s_I}^{\mathscr{D}_{S'}}(\mathscr{L}) > \lambda$.

5. We show that $\mathscr{D}_{S'}$ is also minimal. Assume the opposite. Then there is some $S'' \subseteq S$ with $|S''| < |S'|$ such that $\mathscr{D}_{S''}$ is an MCS for $\mathscr{D}$ and $\psi = \mathbb{P}_{\leq\lambda}\mathscr{L}$. In (6.24a)–(6.24f) we syntactically replace x_s by 1 if $s \in S''$ and by 0 otherwise, and x_B by 1 if $B \subseteq S''$ and T is reachable from sq_I in $\mathscr{D}_{S''} \otimes \mathscr{A}$ and by 0 otherwise. Lemma 3 applied to S'' implies that the constraint system resulting from (6.24c)–(6.24f) by the above substitution has a satisfying assignment; following the argumentation in item (3) above we have that this assignment maps $Pr_{s_I}^{\mathscr{D}_{S''}}(\mathscr{L}) > \lambda$ to p_{sq_I}, thus also satisfying (6.24b). However, for this satisfying assignment the number of positive x_s variables is smaller than for ν, which contradicts our assumption that ν minimizes the objective function.

6. It remains to show that the probability to satisfy $\mathscr{L}$ from s_I in $\mathscr{D}_{S'}$ is maximal under all MCSs. This proof is analogous to the previous item. Assume the opposite. Then there is some $S'' \subseteq S$ such that $\mathscr{D}_{S''}$ is an MCS for $\mathscr{D}$ and $\mathbb{P}_{\leq\lambda}\mathscr{L}$ with a higher probability to satisfy $\mathscr{L}$ in the initial state.

   We apply the same replacement as above to (6.24a)–(6.24f) to get a satisfying assignment μ inducing $\mathscr{D}_{S''}$. Since $\mathscr{D}_{S'}$ and $\mathscr{D}_{S''}$ are both minimal, $\sum_{s \in S} \nu(x_s) = \sum_{s \in S} \mu(x_s)$, however $\nu(p_{sq_I}) < \mu(p_{sq_I})$, contradicting the optimality of ν. □

Lemma 17 *The MILP formulation* (6.24a)–(6.24f) *is complete.*

Proof 17 Let $\mathscr{D}' \sqsubseteq \mathscr{D}$ with state space $S' \subseteq S$ be an MCS of $\mathscr{D}$ for $\psi = \mathbb{P}_{\leq\lambda}(\mathscr{L})$ with a maximal probability to satisfy $\mathscr{L}$ under all MCSs. We show that there is a satisfying assignment ν of the MILP constraints (6.24a)–(6.24f) such that $\nu(x_s) = 1$ iff $s \in S'$, and $\nu(p_{sq_I}) = Pr_{s_I}^{\mathscr{D}_{S'}}(\mathscr{L})$.

Note that $\mathscr{D}_{S'}$ is also an MCS for $\mathscr{D}$ and ψ with the same state space and a maximal probability to satisfy $\mathscr{L}$ under all MCSs. Let $\Pi = \{\pi \in \mathit{Paths}_{\mathit{inf}}^{\mathscr{D}_{S'}}(s_I) \mid \pi \vDash \mathscr{L}\}$ denote the set of infinite paths within the subsystem that satisfy $\mathscr{L}$. Since $\mathscr{D}_{S'}$ is a critical subsystem, $\mathit{Pr}(\Pi) > \lambda$ holds.

For $\pi = s_0, s_1, \ldots \in \Pi$ let $\pi^* = (s_0, q_0)(s_1, q_1)\ldots$ with $q_0 = \delta(q_I, L(s_I))$ and $q_{i+1} = \delta(q_i, L(s_{i+1}))$ be the unique extension of π to the product automaton $\mathscr{D} \otimes \mathscr{A}$. Let $\Pi^* = \{\pi^* \mid \pi \in \Pi\}$ and $\inf(\pi)$ be the set of states which occur infinitely often on π. Since all stepwise probabilities are preserved by the extension, we have that $\mathit{Pr}^{\mathscr{D}}(\Pi) = \mathit{Pr}^{\mathscr{D}\otimes\mathscr{A}}(\Pi^*) > \lambda$.

We now consider the subsystem $S' \times Q$ of $\mathscr{D} \otimes \mathscr{A}$. Π^* contains only paths in $S' \times Q$. $\mathscr{B}$ denotes the set of bottom SCCs of $\mathscr{D} \otimes \mathscr{A}$. Then $\mathit{Pr}(\{\pi \in \mathit{Paths}_{\mathit{inf}}^{\mathscr{D}\otimes\mathscr{A}}(sq_I) \mid \inf(\pi) \in \mathscr{B}\}) = 1$ [BK08, Theorem 10.27]. Contrarily, $\mathit{Pr}(\{\pi \in \mathit{Paths}_{\mathit{inf}}^{\mathscr{D}\otimes\mathscr{A}}(sq_I) \mid \inf(\pi) \notin \mathscr{B}\}) = 0$. We can conclude:

$$\begin{aligned} 0 &\le \mathit{Pr}(\{\pi^* \in \Pi^* \mid \inf(\pi^*) \notin \mathscr{B}\}) \\ &\le \mathit{Pr}(\{\pi^* \in \mathit{Paths}_{\mathit{inf}}^{\mathscr{D}\otimes\mathscr{A}}(sq_I) \mid \inf(\pi^*) \notin \mathscr{B}\}) \\ &= 0 \end{aligned}$$

and

$$\lambda < \mathit{Pr}(\Pi^*) = \mathit{Pr}(\{\pi^* \in \Pi^* \mid \inf(\pi^*) \in \mathscr{B}\}) \ .$$

We now set $C := \{\inf(\pi^*) \mid \pi^* \in \Pi^*\} \cap \mathscr{B}$. We make the following observations:

- all elements of C are BSCCs, and
- $\forall c \in C. \exists i \in \{1, \ldots, n\}. (\forall u \in c. R_i \notin L'(u)) \wedge (\exists u \in c. A_i \in L'(u))$, i. e., C contains only accepting BSCCs. Otherwise the paths in Π^* were not accepted.

We define the following variable assignment ν for the decision variables: $\nu(x_s) = 1$ iff $s \in S'$ and $\nu(x_B) = 1$ iff $B \in C$. These assignments trigger the following implications in the MILP constraints above:

$$p_u = \begin{cases} 0, & \text{if } u = (s,q) \text{ and } s \notin S', \\ 1, & \text{if } u \in B \in C, \\ \sum\limits_{u' \in U} P'(u,u') \cdot p_{u'}, & \text{otherwise.} \end{cases}$$

Using Lemma 3, we can show that this linear equation system has a satisfying assignment ν which describes the probability of reaching a target state within the subsystem $S' \times Q$. Therefore $\nu(p_{sq_I}) = \mathit{Pr}_{sq_I}^{\mathscr{D}\otimes\mathscr{A}}(\Diamond \mathit{accept}) \ge \mathit{Pr}^{\mathscr{D}\otimes\mathscr{A}}(\Pi^*) = \mathit{Pr}^{\mathscr{D}}(\Pi) > \lambda$.

We show that ν is optimal with respect to Constraint 6.24a.

- If ν did not minimize the number of states, then there would be another satisfying assignment that evaluates x_s to 1 for a fewer number of states. Due to soundness of the MILP formulation (Lemma 16), there would exist an MCS smaller than $\mathscr{D}_{S'}$, which contradicts the minimality assumption for $\mathscr{D}_{S'}$.

- Similarly, if ν did not maximize p_{s_I}, then there would be another satisfying assignment selecting the same (minimal) number of states but mapping a larger value to p_{s_I} than ν does. By soundness (Lemma 16), there would exist an MCS in which the probability to satisfy $\mathscr{L}$ in the initial state is larger than in $\mathscr{D}_{S'}$, which contradicts the assumption that $\mathscr{D}_{S'}$ maximizes this probability under all MCSs. □

Theorem 11 *7 The MILP formulation* (6.24a)–(6.24f) *is sound and complete.*

Proof 18 The MILP formulation is sound by Lemma 16 and complete by Lemma 17.

6.3.5 MILP-formulation for reachability properties of PAs

Let in the following $\mathscr{M} = (S, s_I, Act, \mathscr{P}, L)$ be a PA, $\psi = \mathbb{P}_{\leq\lambda}(\Diamond T)$ a reachability property violated by $\mathscr{M}$, and $T \subseteq S$ be a set of target states. The MILP formulation for ψ and $\mathscr{M}$ reads as follows:

$$\text{minimize} \quad -\frac{1}{2} p_{s_I} + \sum_{s \in S} x_s \tag{6.25a}$$

$$\text{such that}$$

$$p_{s_I} > \lambda \tag{6.25b}$$

$$\forall s \in T.\quad p_s = x_s \tag{6.25c}$$

$$\forall s \in S \setminus T.\quad p_s \leq x_s \tag{6.25d}$$

$$\forall s \in S \setminus T.\quad \sum_{\eta \in \mathscr{P}(s)} \sigma_{s,\eta} = x_s \tag{6.25e}$$

$$\forall s \in S \setminus T.\ \forall (\alpha,\mu) \in \mathscr{P}(s).\quad p_s \leq (1 - \sigma_{s,\alpha,\mu}) + \sum_{s' \in succ_{\mathscr{M}}(s,\alpha,\mu)} \mu(s') \cdot p_{s'} \tag{6.25f}$$

$$\forall s \in S_{\text{probl}(T)}.\, \forall \eta \in H_{\text{probl}(T)}(s).\quad \sum_{s' \in succ_{\mathscr{M}}(s,\eta)} t^{\leftarrow}_{s,s'} \leq x_{s'} \tag{6.25g}$$

$$\forall s \in S_{\text{probl}(T)}.\, \forall \eta \in H_{\text{probl}(T)}(s).\, \forall s' \in supp(\eta).\quad r^{\leftarrow}_s < r^{\leftarrow}_{s'} + (1 - t^{\leftarrow}_{s,s'}) \tag{6.25h}$$

$$\forall s \in S_{\text{probl}(T)}.\, \forall \eta \in H_{\text{probl}(T)}(s).\quad (1 - \sigma_{s,\eta}) + \sum_{s' \in succ_{\mathscr{M}}(s,\eta)} t^{\leftarrow}_{s,s'} \geq x_s\ . \tag{6.25i}$$

Lemma 18 *The MILP formulation* (6.25a)–(6.25i) *is sound.*

Proof 19 Let ν be a satisfying assignment of the MILP constraints (6.25a)–(6.25i) and let $S' = \{s \in S \mid \nu(x_s) = 1\}$ as before. We define the (partial) memoryless deterministic scheduler $\sigma\colon S' \setminus T \to Act \times subDistr(S)$ by $\sigma(s) = \eta$ iff $\nu(\sigma_{s,\eta}) = 1$ for $\eta \in Act \times subDistr(S)$. The scheduler

σ is well-defined, since Constraint 6.25e ensures that for each $s \in S' \setminus T$ there is exactly one action $\alpha \in \mathit{Act}$ with $\sigma_{s,\eta} = 1$.

We show that the DTMC $\mathcal{D}_{S'} = (S', s_I, P', L')$ with $P'(s,s') = \eta(s')$ for $\eta \in \mathcal{P}(s)$ and $\sigma(s) = \eta$ and $L'(s') = L(s')$ for all $s \in S' \setminus T$ and $s' \in S'$ is an MCS of $\mathcal{M}$ for ψ with maximal probability to reach T from s_I under all MCSs.

1. We show that $\mathcal{D}_{S'}$ is a subsystem of $\mathcal{M}$. From (6.25b) we conclude $0 \le \lambda < \nu(p_{s_I})$, and therefore by the satisfaction of (6.25c)–(6.25d) we have that $\nu(x_{s_I}) = 1$, i. e., $s_I \in S'$. The remaining conditions for $\mathcal{D}_{S'}$ being a subsystem of $\mathcal{M}$ hold by the definition of $\mathcal{D}_{S'}$.

2. We show that all states of $\mathcal{D}_{S'}$ are relevant for T, see Definition 24 on Page 34. By definition, from all unproblematic states $s \in S' \setminus S^{\mathcal{M}}_{\text{probl}(T)}$ there is a path in $\mathcal{M}_{\sigma'}$ to a target state for all schedulers σ'. This holds also for each extension of the (partial) scheduler σ. Due to the backward reachability given by Constraints 6.25g–6.25i, from all states that are problematic in $\mathcal{M}$ an unproblematic state and therefore also a target state is reachable in $\mathcal{D}_{S'}$.

3. We show that $\nu(p_{s_I}) = Pr^{\mathcal{D}_{S'}}_{s_I}(\Diamond T)$. The constraints (6.25c)–(6.25d) assure that (i) $\nu(p_s) = 0$ for all $s \in S \setminus S'$ and (ii) $\nu(p_s) = 1$ for all $s \in S' \cap T$. Therefore, due to the satisfaction of (6.25f) for the action-distribution pairs selected by σ, the assignment ν satisfies the following constraint system:

$$\begin{aligned} \forall s \in S' \setminus T. \quad p_s \le & \quad (1 - \sigma_{s,\sigma(s)}) + \sum_{s' \in succ_{\mathcal{M}}(s,\sigma(s))} \sigma(s)(s') \cdot p_{s'} \\ = & \quad \sum_{s' \in S' \setminus T} P'(s,s') \cdot p_{s'} + \sum_{s' \in S' \cap T} P'(s,s') . \end{aligned}$$

   As shown in (2), all states of $\mathcal{D}_{S'}$ are relevant for a. Lemma 6 implies $\nu(p_{s_I}) \le Pr^{\mathcal{D}_{S'}}_{s_I}(\Diamond T)$, and Lemma 3 implies that there is also a satisfying assignment mapping $Pr^{\mathcal{D}_{S'}}_{s_I}(\Diamond T)$ to p_{s_I} (satisfying the inequations with equalities). Since ν is a satisfying assignment maximizing p_{s_I} in (6.25a), $\nu(p_{s_I}) = Pr^{\mathcal{D}_{S'}}_{s_I}(\Diamond T)$ must hold.

4. We show that $\mathcal{D}_{S'}$ is critical. In (3) we have shown that $\nu(p_{s_I}) = Pr^{\mathcal{D}_{S'}}_{s_I}(\Diamond T)$. Combined with $0 \le \lambda < \nu(p_{s_I})$ from (1) we get $\lambda < Pr^{\mathcal{D}_{S'}}_{s_I}(\Diamond T)$.

5. We show that $\mathcal{D}_{S'}$ is minimal. Assume the opposite. Then there is a scheduler σ'' and a state set $S'' \subseteq S$ with $|S''| < |S'|$ such that the DTMC $\mathcal{D}_{S''} = (S'', s_I, P'', L'')$ with $P''(s,s') = \eta(s')$ for $\sigma''(s) = \eta \in \mathit{Act} \times \mathit{subDistr}(S)$ and $L''(s) = L(s)$ for all $s, s' \in S''$ is an MCS of $\mathcal{M}$ for $\mathbb{P}_{\le\lambda}(\Diamond T)$.

   Note that, since $\mathcal{D}_{S''}$ is minimal, it has only states that are relevant for T, i. e., T is reachable from all of its states. We consider a set Π of paths of $\mathcal{D}_{S''}$ such that Π is postfix-closed and it contains for each state $s \in S''$ exactly one shortest path from s to T in $\mathscr{D}_{S''}$. We define the function $f : S'' \setminus T \to S''$ by assigning to each state $s \in S'' \setminus T$ the unique successor state of s on the shortest path from s to T in Π.

Now we define an assignment $\mu\colon Var \to \mathbb{R}$ by

- $\mu(x_s) = 1$ and $\mu(p_s) = Pr_{s_I}^{\mathscr{D}_{S''}}(\Diamond T)$ for $s \in S''$, and $\mu(x_s) = \mu(p_s) = 0$ otherwise
- $\mu(\sigma_{s,\eta}) = 1$ if $s \in S''$ and $\sigma''(s) = \eta$, and $\mu(\sigma_{s,\eta}) = 0$ otherwise
- for all $s, s' \in S$ we have that $\mu(t^{\leftarrow}_{s,s'}) = 1$ if $s \in S'' \setminus T$ and $s' = f(s)$, and $\mu(t^{\leftarrow}_{s,s'}) = 0$ otherwise
- for all $s \in S$ we have that $\mu(r^{\leftarrow}_s) = 1 - n/d$ where n is the length of a shortest path from s to T in $\mathscr{D}_{S''}$, and d is the maximum of the lengths of all shortest paths from any state to T in $\mathscr{D}_{S''}$.

Lemma 3 applied to the induced DTMC $\mathscr{M}_{\sigma''}$ and S'' implies that μ satisfies the constraints (6.25f) with equality. The satisfaction of the other constraints is easy to see.

However, $\sum_{s\in S} \mu(x_s) < \sum_{s\in S} \nu(x_s)$, what contradicts our assumption that ν is optimal with respect to Condition (6.25a).

6. It remains to show that the probability to reach T from s_I in $\mathscr{D}_{S'}$ is maximal under all MCSs. This proof is analogous to the previous item (5). Assume the opposite. Then there is a scheduler σ'' and some $S'' \subseteq S$ such that $\mathscr{D}_{S''}$ defined as above is an MCS for $\mathscr{M}$ and $\mathbb{P}_{\le\lambda}(\Diamond T)$ with $Pr_{s_I}^{\mathscr{D}_{S'}}(\Diamond T) < Pr_{s_I}^{\mathscr{D}_{S''}}(\Diamond T)$. We define μ as above. Again, with the help of Lemma 3 we can show that μ satisfies all constraints. Since $\mathscr{D}_{S'}$ and $\mathscr{D}_{S''}$ are both minimal, $\sum_{s\in S} \nu(x_s) = \sum_{s\in S} \mu(x_s)$. From Condition (3) we know that $\nu(p_{s_I}) = Pr_{s_I}^{\mathscr{D}_{S'}}(\Diamond T)$, i. e.,

$$\nu(p_{s_I}) = Pr_{s_I}^{\mathscr{D}_{S'}}(\Diamond T) < Pr_{s_I}^{\mathscr{D}_{S''}}(\Diamond T) = \mu(p_{s_I}).$$

Thus $-\frac{1}{2}\mu(p_{s_I}) + \sum_{s\in S} \mu(x_s) < -\frac{1}{2}\nu(p_{s_I}) + \sum_{s\in S} \nu(x_s)$, contradicting the optimality of ν. □

Lemma 19 *The MILP formulation* (6.25a)–(6.25i) *is complete.*

Proof 20 Since $\mathscr{M} \not\models \mathbb{P}_{\le\lambda}(\Diamond T)$, there is a scheduler σ and a state set S' such that the DTMC $\mathscr{D}_{S'} = (S', s_I, P', L')$ with $P'(s,s') = P(s,\sigma(s),s')$ and $L'(s) = L(s)$ for all $s, s' \in S'$ is an MCS of $\mathscr{M}$ for $\mathbb{P}_{\le\lambda}(\Diamond T)$ having a maximal probability to reach T under all MCSs. We show that there is a satisfying assignment ν of the MILP formulation (6.25a)–(6.25i) with $S' = \{s \in S \mid \nu(x_s) = 1\}$ and $\nu(\sigma_{s,\alpha}) = 1$ iff $\sigma(s) = \alpha$ such that $\nu(p_{s_I}) = Pr_{s_I}^{\mathscr{D}_{S'}}(\Diamond T)$.

As in the proof of Lemma 13, we consider a set Π of paths of $\mathscr{D}_{S'}$ such that Π is postfix-closed and it contains for each state $s \in S'$ exactly one shortest path from s to T in $\mathscr{D}_{S'}$. We know that such a set exists, since minimality of the MCS $\mathscr{D}_{S'}$ implies that T can be reached from all states in $\mathscr{D}_{S'}$. We define the function $f : S' \setminus T \to S'$ by assigning to each state $s \in S' \setminus T$ the unique successor state of s on the shortest path from s to T in Π.

We define the assignment $\nu : \mathit{Var} \to \mathbb{R}$ by

- $\nu(x_s) = 1$ and $\nu(p_s) = Pr_{s_I}^{\mathscr{D}_{S'}}(\Diamond T)$ for $s \in S'$, and $\nu(x_s) = \nu(p_s) = 0$ otherwise,
- $\nu(\sigma_{s,\eta}) = 1$ if $s \in S'$ and $\sigma(s) = \eta$, and $\nu(\sigma_{s,\eta}) = 1$ otherwise,
- for all $s, s' \in S$, $\nu(t^{\leftarrow}_{s,s'}) = 1$ if $s \in S' \setminus T$ and $s' = f(s)$, and $\nu(t^{\leftarrow}_{s,s'}) = 0$ otherwise, and
- for all $s \in S$, $\nu(r^{\leftarrow}_s) = 1 - n/d$ where n is the length of a shortest path from s to T in $\mathscr{D}_{S'}$, and d is the maximum of the lengths of all shortest paths from any state to T in $\mathscr{D}_{S'}$.

Lemma 3 implies that μ satisfies the constraints (6.25f) with equality. The satisfaction of the other constraints is easy to show by replacing the variables by their values under ν (see also the proof of Lemma 8). □

Theorem 12 *9 The MILP formulation* (6.25a)–(6.25i) *is sound and complete.*

Proof 21 The MILP formulation is sound by Lemma 18 and complete by Lemma 19. □

6.3.6 MILP-formulation for ω-regular properties of PAs

6.3.6.1 Requirements

Let $\mathscr{M} = (S, s_I, Act, \mathscr{P}, L)$ be a PA, $\mathbb{P}_{\leq\lambda}(\mathscr{L})$ an ω-regular property which is violated by $\mathscr{M}$, and $\mathscr{A} = (Q, q_I, 2^{AP}, \delta, F)$ a DRA with $F = \{(R_i, A_i) \mid i = 1, \ldots, n\}$ and $\mathscr{L}(\mathscr{A}) = \mathscr{L}$.

We consider the product $\mathscr{M} \otimes \mathscr{A}$ of the PA $\mathscr{M}$ and the DRA $\mathscr{A}$ as in Definition 31 with the probabilistic transition relation $\mathscr{P}'$, and assume that all irrelevant states have been removed, see Definition 24. To simplify notation we use $U = S \times Q$ and $u = (s, q)$, $u' = (s', q')$, etc. as typical elements from U.

Lemma 20 *2 Let* $(R_i, A_i) \in 2^Q \times 2^Q$ *be a pair of a Rabin acceptance condition,* $\sigma : U \to Act \times subDistr(U)$ *a scheduler, and* $M_i \subseteq U$ *a set of states with the following properties:*

1. $\forall u \in M_i. \sum_{u' \in succ(u,\eta) \cap M_i} \eta(u') = 1$, *where* $\sigma(u) = \eta$,
2. $M_i \cap (S \times R_i) = \emptyset$, *and*
3. *for each state* $u \in M_i$ *there is a path from* u *to a state in* $S \times A_i$.

Then the probability of satisfying the acceptance condition F *in* $\mathscr{M}$ *because of the pair* (R_i, A_i) *is 1 for all* $u \in M_i$.

Proof 22 Since M_i is closed under successors w. r. t. scheduler σ, this set forms a sub-PA of $\mathscr{M}$. The probability to reach a BSCC under scheduler σ is 1 for every state of M_i. Let $M_i' \subseteq M_i$ be such a BSCC. As M_i' is strongly connected, it forms an end component of $\mathscr{M}$. As a state out of $S \times A_i$ is reachable from every state of M_i, at least one state of $S \times A_i$ has to be included in M_i'. Hence, M_i' is an accepting end component of $\mathscr{M}$. As this holds for every BSCC included in M_i, the probability to reach an accepting end component inside M_i is one. □

6.3.6.2 Soundness and completeness

Recall the MILP-formulation for an MCS of $\mathscr{M}$ for $\psi = \mathbb{P}_{\leq\lambda}(\mathscr{L})$ as given in Section 6.2.2.

$$\text{minimize} \quad -\frac{1}{2} p_{sq_I} + \sum_{s\in S} x_s \tag{6.26a}$$

such that

- Selection of at most one action-distribution pair per state:

$$\forall u = (s,q) \in U. \quad \sum_{\eta\in\mathscr{P}(u)} \sigma_{u,\eta} \leq x_s \tag{6.26b}$$

- Definition of the set M_i for all $i = 1,\dots,n$:

$$\forall u \in U. \forall \eta \in \mathscr{P}(u) \text{ with } \sum_{u'\in U} \eta(u') < 1. \quad m^i_u \leq 1 - \sigma_{u,\eta} \tag{6.26c}$$

$$\forall u \in U. \forall \eta \in \mathscr{P}(u). \quad n_{u,\eta} \cdot (2 - \sigma_{u,\eta} - m^i_u) + \sum_{u'\in supp(\eta)} m^i_{u'} \geq n_{u,\eta} \tag{6.26d}$$

$$\forall u \in S \times R_i. \quad m^i_u = 0 \tag{6.26e}$$

- Backward reachability of $S \times A_i$ within M_i for all $i = 1,\dots,n$:

$$\forall u \in U. \forall \eta \in \mathscr{P}(u). \forall u' \in supp(\eta). \quad t^i_{u,u'} \leq m^i_{u'} + (1 - \sigma_{u,\eta}) \tag{6.26f}$$

$$\forall u \in U. \forall \eta \in \mathscr{P}(u). \forall u' \in supp(\eta). \quad r^i_u < r^i_{u'} + (1 - t^i_{u,u'}) + (1 - \sigma_{u,\eta}) \tag{6.26g}$$

$$\forall u \in S \times (Q \setminus A_i). \forall \eta \in \mathscr{P}(u). \quad (1 - \sigma_{u,\eta}) + \sum_{u'\in supp(\eta)} t^i_{u,u'} \geq m^i_u \tag{6.26h}$$

- Probability computation:

$$p_{sq_I} > \lambda \tag{6.26i}$$

$$\forall i = 1,\dots,n. \forall u \in U. \quad p_u \geq m^i_u \tag{6.26j}$$

$$\forall u \in U. \quad p_u \leq \sum_{\eta\in\mathscr{P}(u)} \sigma_{u,\eta} \tag{6.26k}$$

$$\forall u \in U.\, \forall \eta \in \mathscr{P}(u). \quad p_u \le (1-\sigma_{u,\eta}) + \sum_{i=1}^{n} m_u^i + \sum_{u' \in supp(\eta)} \eta(u') \cdot p_{u'} \tag{6.26l}$$

- Backward reachability of $M = \bigcup_{i=1}^{n} M_i$ within the subsystem:

$$\forall u \in U.\, \forall \eta \in \mathscr{P}(u).\, \forall u' = (s',q') \in supp(\eta). \quad t^M_{u,u'} \le x_{s'} + (1-\sigma_{u,\eta}) \tag{6.26m}$$

$$\forall u \in U.\, \forall \eta \in \mathscr{P}(u).\, \forall u' \in supp(\eta). \quad r^M_u < r^M_{u'} + (1-t^M_{u,u'}) + (1-\sigma_{u,\eta}) \tag{6.26n}$$

$$\forall u = (s,q) \in U.\, \forall \eta \in \mathscr{P}(u). \quad (1-\sigma_{u,\eta}) + \sum_{i=1}^{n} m_u^i + \sum_{u' \in supp(\eta)} t^M_{u,u'} \ge x_s\,. \tag{6.26o}$$

Lemma 21 *The MILP formulation* (6.26a)–(6.26o) *is sound.*

Proof 23 Assume a satisfying assignment ν of the MILP (6.26a)–(6.26o). Analogously to DTMCs, we define the restricted PA $\mathscr{M}' = (S', s_I, Act, \mathscr{P}', L')$ with $S' = \{s \in S \mid \nu(x_s) = 1\}$,

$$\mathscr{P}'(s) = \{\eta \in \mathscr{P}(s) \mid \exists q \in Q.\, \nu(\sigma_{(s,q),\eta}) = 1\}$$

and $L'(s) = L(s)$ for all $s, s' \in S'$. We show that $\mathscr{M}'$ is an MCS of $\mathscr{M}$ for $\mathbb{P}_{\le\lambda}(\mathscr{L})$, with a maximal probability to satisfy $\mathscr{L}$ under all MCSs.

1. We show that $\mathscr{M}'$ is a subsystem of $\mathscr{M}$. From Constraint 6.26i we imply $\nu(p_{sq_I}) > \lambda \ge 0$. Using Constraints 6.26b and 6.26k we get that $\nu(x_{s_I}) = 1$, i. e., $s_I \in S'$. The other conditions for $\mathscr{M}'$ being a subsystem of $\mathscr{M}$ are straightforward by the definition of $\mathscr{M}'$.

2. For Constraint 6.26b we observe that $\big|\{\eta \in \mathscr{P}(u) \mid \nu(\sigma_{u,\eta}) = 1\}\big| \le 1$ for all $u \in U$. Therefore the deterministic memoryless scheduler σ for $\mathscr{M}' \otimes \mathscr{A}$ with $\sigma(u)(\eta) = \nu(\sigma_{u,\eta})$ for all $u \in S' \times Q$ is well-defined, and it induces a DTMC $\mathscr{D}' = (\mathscr{M}' \otimes \mathscr{A})_\sigma$ with state set $U' = S' \times Q$. In the following we use the notation $\mathrm{def}(\sigma) = \{u \in U' \mid \exists \eta \in \mathscr{P}(u) \colon \sigma(u)(\eta) = 1\}$ as the set of states where an action-distribution pair is chosen by the scheduler. We show that $Pr_u^{\mathscr{D}'}(\Diamond accept) = \nu(p_u)$ for all states in $u \in U'$ (where all states in all accepting end components of $\mathscr{D}'$ are labeled with *accept*).

   For $i = 1, \dots, n$ let $M_i = \{u \in U \mid \nu(m_u^i) = 1\}$ and $M = \bigcup_{i=1}^{n} M_i$. Using

$$\nu(m_u^i) \overset{(6.26j)}{\le} \nu(p_u) \overset{(6.26k)}{\le} \sum_{\eta \in \mathscr{P}(u)} \nu(\sigma_{u,\eta}) \overset{(6.26b)}{\le} \nu(x_s) \tag{6.27}$$

   for all $u = (s,q) \in U$ and $i = 1, \dots, n$, we have that $M \subseteq U'$.

   - We start by showing that $Pr_u^{\mathscr{D}'}(\Diamond accept) = \nu(p_u) = 1$ holds for all $u \in M$. First, all prerequisites of Lemma 2 on Page 143 are satisfied for each M_i as state set and the A_i-states in M_i as target states. The Constraints 6.26j–6.26k assure that $M_i \subseteq \mathrm{def}(\sigma)$, i. e., that, the scheduler σ selects an action for all M_i-states, for which by Constraint 6.26c it holds that

$$\sum_{u' \in succ_{\mathscr{M} \otimes \mathscr{A}}(u,\sigma(u))} \eta(u') = 1$$

where $\sigma(u) = \eta$.

M_i is closed under successors with respect to the action-distribution pairs selected by σ because of Constraint 6.26d. Furthermore, M_i does not contain any R_i-states according to Constraint 6.26e. Given the assignment of $\sigma_{u,\eta}$, Constraints 6.26f–6.26h are backward reachability constraints with the A_i-states as the target states. According to Lemma 14 on Page 157, an assignment ν is satisfying these constraints iff from all states in M_i an A_i-state state is reachable inside M_i. Therefore by Lemma 2 it follows that $Pr_u^{\mathscr{D}'}(\lozenge accept) = 1$ for all states $u \in \bigcup_{i=1}^{n} M_i$, which coincides with $\nu(p_u)$ because of Constraint 6.26j. It follows that $Pr_u^{\mathscr{D}'}(\lozenge accept) = \nu(p_u) = 1$ for all $u \in M$.

- Now we show that $Pr_u^{\mathscr{D}'}(\lozenge accept) = \nu(p_u)$ for all $u \in U' \setminus M$.

  Constraints 6.26m–6.26o ensure that for all states $u \in U' \setminus M$ either $u \notin \text{def}(\sigma)$ or a state in M is reachable from u (cf. Lemma 14).

  For states $u \in U \setminus \text{def}(\sigma)$ without any action selected by σ, Constraint 6.26k implies $\nu(p_u) = 0 = Pr_u^{\mathscr{D}'}(\lozenge accept)$. Assume that these states and their connected (incoming) transitions are removed from $\mathscr{D}'$.

  Note that for non-selected states $u \in U \setminus U'$, Constraints 6.26b and 6.26k enforce $\nu(p_u) = 0$. Recall furthermore that $\nu(p_u) = 1$ for each $u \in M$. Therefore, for each $u \in (U' \setminus M) \cap \text{def}(\sigma)$ and $\eta \in \mathscr{P}(u)$ with $\sigma(u) = \eta$, according to Constraint 6.26l it holds that

$$\nu(p_u) \leq \sum_{u' \in (U' \cap \text{def}(\sigma)) \setminus M} \eta(u') \cdot \nu(p_{u'}) + \sum_{u' \in M} \eta(u') .$$

  Lemma 6 applied to the state set $U' \cap \text{def}(\sigma)$ and target set M gives us $\nu(p_u) \leq Pr_u^{\mathscr{D}'}(\lozenge accept)$. According to the objective function in Constraint 6.26a, ν maximizes the probability p_{sq_I}. Lemma 3 states that the maximal solution satisfies $Pr_u^{\mathscr{D}'}(\lozenge accept) = \nu(p_u)$ for all $u \in (U' \cap \text{def}(\sigma)) \setminus M$. We conclude that $\nu(p_u) = Pr_u^{\mathscr{D}'}(\lozenge accept)$ for all $u \in U'$.

3. Above we have shown that $\nu(p_{sq_I}) = Pr_{sq_I}^{\mathscr{D}'}(\lozenge accept)$, and by Constraint 6.26i we have that $\nu(p_{sq_I}) > \lambda$. Thus $Pr_{sq_I}^{\mathscr{D}'}(\lozenge accept) > \lambda$. Using Theorem 2 we get that $Pr_{s_I}^{\mathscr{M}^\sigma}(\mathscr{L}) = Pr_{sq_I}^{\mathscr{D}'}(\lozenge accept) > \lambda$, i. e., $\mathscr{M}'$ is critical.

4. We show that $\mathscr{M}'$ is minimal. Assume the opposite. Then there is an MCS $\mathscr{M}''$ for $\mathscr{M}$ and $\mathbb{P}_{\leq\lambda}(\mathscr{L})$ with state set $S'' \subseteq S$ such that $|S''| < |S'|$. Since $\mathscr{M}''$ is an MCS, there is a

deterministic memoryless scheduler σ for $\mathscr{M}'' \otimes \mathscr{A}$ such that $Pr_{sq_I}^{(\mathscr{M}''\otimes\mathscr{A})^\sigma}(\lozenge accept) > \lambda$.

In the Constraints 6.26a–6.26o we syntactically replace (i) x_s by 1 if $s \in S''$ and by 0 otherwise, (ii) m_u^i by 1 if $u \in S'' \times Q$ is in an end component of $\mathscr{M}'' \otimes \mathscr{A}$ accepting for the ith accepting condition and by 0 otherwise, and $\sigma_{u,\eta}$ by $\sigma(u)(\eta) \in \{0,1\}$.

Lemma 3 applied to S'' implies that the constraint system resulting from the above substitution has a satisfying assignment; following the argumentation in Item (2) above we get that this assignment maps $Pr_{s_I}^{\mathscr{D}_{S''}}(\mathscr{L}) > \lambda$ to p_{sq_I}, thus also satisfying Constraint 6.26i. However, for this satisfying assignment the number of positive x_s variables is smaller than for ν, which contradicts our assumption that ν minimizes the objective function.

5. It remains to show that the probability to satisfy $\mathscr{L}$ from s_I in $\mathscr{M}'$ is maximal among all MCSs. This proof is analogous to the previous item. Assume the opposite. Then there is some MCS $\mathscr{M}''$ of $\mathscr{D}$ for $\mathbb{P}_{\leq\lambda}(\mathscr{L})$ with state set $S'' \subseteq S$ such that the probability to satisfy $\mathscr{L}$ in the initial state is higher in $\mathscr{M}''$ as in $\mathscr{M}'$.

   We apply the same replacement as above to the constraint system (6.26a)–(6.26o) to get a satisfying assignment μ inducing $\mathscr{M}''$. Since $\mathscr{M}'$ and $\mathscr{M}''$ are both minimal, $\sum_{s\in S} \nu(x_s) = \sum_{s\in S} \mu(x_s)$, however $\nu(p_{sq_I}) < \mu(p_{sq_I})$, contradicting the optimality of ν. □

Lemma 22 *The MILP formulation* (6.26a)–(6.26o) *is complete.*

Proof 24 Let $\mathscr{M}' = (S', s_I, Act, \mathscr{P}', L')$ be an MCS of $\mathscr{M}$ for $\mathbb{P}_{\leq\lambda}(\mathscr{L})$, in which the probability to satisfy $\mathscr{L}$ in the initial state is the highest among all MCSs.

Since the subsystem $\mathscr{M}'$ is critical, there is a memoryless deterministic scheduler σ for $\mathscr{M}' \times \mathscr{A}$ such that $Pr_{sq_I}^{(\mathscr{M}'\otimes\mathscr{A})^\sigma}(\lozenge accept) > \lambda$. Let $\mathscr{B}$ be the set of accepting BSCCs of the induced DTMC $(\mathscr{M} \otimes \mathscr{A})_\sigma$ and $M_i = \bigcup\{C \in \mathscr{B} \mid C \cap R_i = \emptyset \wedge C \cap A_i \neq \emptyset\}$ for $i = 1, \ldots, m$.
We define the following partial assignment ν:

- $\nu(x_s) = 1$ iff $s \in S'$
- $\nu(\sigma_{u,\eta}) = 1$ iff $u = (s,q) \wedge s \in S' \wedge \sigma(u)(\eta) = 1$, and
- $\nu(p_u) = Pr_u^{(\mathscr{M}'\otimes\mathscr{A})_\sigma}(\lozenge accept)$,
- $\nu(m_u^i) = 1$ iff $u \in M_i$.

Now we show that there is a total extension of this assignment that satisfies all constraints:

(6.26a) The defined assignment minimizes this function, since the MCS is minimal with maximal probability to satisfy $\mathscr{L}$ under all MCSs.

(6.26b) This constraint is satisfied since we do not select any action for states $u = (s,q)$ with $s \notin S'$ and σ selects exactly one action-distribution pair for each state $u = (s,q)$ with $s \in S'$.

(6.26c) Since all states of M_i are contained in a BSCC, and—for all states in a BSCC—the probability that a successor state is also in a BSCC is 1, this constraint is satisfied.

(6.26d) For states u outside M_i and for action-distribution pairs not chosen by σ, the constraint is satisfied because in these cases $(2 - m^i_u - \sigma_{u,\eta}) \geq 1$. For states $u = (s,q)$ with $s \in S'$ and $\eta = \sigma(u)$, $\nu(m^i_{u'}) = 1$ is required for all successor states u' of u. This is the case since M_i is a union of BSCCs.

(6.26e) In the definition of M_i we have required that $M_i \cap R_i = \emptyset$. Therefore this constraint is satisfied.

(6.26f)–(6.26h) Each accepting BSCC in M_i contains by construction a state from A_i. Since in a BSCC each state is reachable from each state, we can apply Lemma 13 to obtain a satisfying assignment for these backward reachability constraints.

(6.26i) $\nu(p_{sq_I}) = Pr^{(\mathcal{M}' \otimes \mathcal{A})_\sigma}_{sq_I}(\Diamond accept) > \lambda$ holds since the subsystem is critical.

(6.26j) For target states, which are the states in the accepting BSCCs, the reachability probability is one.

(6.26k) For each deadlock state u without outgoing transitions it holds that $\nu(p_u) = Pr^{(\mathcal{M}' \otimes \mathcal{A})_\sigma}_{u}(\Diamond accept) = 0$. Therefore, the inequality is trivially satisfied. For non-deadlocking states, this inequation puts no constraints on the probability values, thus it holds also in that case.

(6.26l) For states from an M_i this constraint is trivially satisfied since the right-hand side evaluates at least to one. The case for $\sigma(u) \neq \eta$ is similarly straightforward. The reachability probabilities for the remaining states which can reach the accepting BSCCs satisfy the equality

$$p_u = \sum_{u' \in succ_{\mathcal{M}' \otimes \mathcal{A}}(u,\eta)} \eta(u') \cdot p_{u'}$$

and therefore satisfy also this constraint. For the remaining states $\nu(p_u) = 0$ holds, also satisfying the constraint.

(6.26m)–(6.26o) These are the backward reachability constraints ensuring reachability of the accepting BSCCs. We distinguish different cases:

- $s \notin S'$: Set $\nu(t^M_{u,u'}) = 0$ for all $q \in Q$, $u = (s,q)$, $u' \in succ_{\mathcal{M}' \otimes \mathcal{A}}(u)$ and $\nu(r^M_u) = 0$. Then all three constraints are satisfied.
- $s \in S'$, but from u no accepting BSCC can be reached. Choose $\nu(t^M_{u,u'}) = 0$ and $\nu(r^M_u) = 0$ as in the previous case. Since $\nu(\sigma_{u,\eta}) = 0$ for all $\eta \in \mathcal{P}(u)$, the three constraints are satisfied.

- $s \in S'$ and from u a BSCC can be reached. According to Lemma 13 we can find a satisfying assignment for these backward reachability constraints.

We have shown that the constructed assignment ν satisfies all constraints of the MILP. □

Theorem 13 *10 The MILP formulation* (6.26a)–(6.26o) *is sound and complete.*

Proof 25 The MILP formulation is sound by Lemma 21 and complete by Lemma 22. □

CHAPTER 7

High-level counterexamples for probabilistic automata

Summary In this chapter we present a new approach to the computation and representation of counterexamples. All previous approaches, be it the ones described in Chapter 3 concerning related work or the ones described in Chapters 5 and 6, explicitly or symbolically present the *state-space* of a counterexample. Although these methods are often able to generate small counterexamples in a compact representation, they might still be too large to be understandable for human users.

It is therefore only natural to present counterexamples at the *modeling level* of the system at hand. Following our previous approaches, we want to determine a *minimal critical model description*, i. e., a high-level description of a counterexample that is as small as possible. In detail, we assume the general class of PAs to be described by the PRISM [KNP11] input language, which is a stochastic version of Alur and Henzinger's reactive modules [AH99]. As many models are inherently concurrent, a model for a PA can be specified as the parallel composition of single modules, where modules communicate by shared variables or synchronization on common actions. Here, we determine a subset of guarded commands that together induce a critical subsystem as presented in Section 5.1. Furthermore, as an extension we are able to simplify the commands by removing command branches which are not necessary to obtain a counterexample.

We present this as a special case of a method where the number of different transition labels for a probabilistic automaton is minimized, again by developing dedicated MILP encodings.

Background This chapter does not need extensive background, as the PRISM language is introduced here. Apart from this, we need again the explicit model checking of DTMCs and PAs based on solving linear equation systems, see Sections 2.3.1.1 and 2.3.1.2. Moreover, recall the introduction to solving techniques in Sections 2.6.

7.1 PRISM's guarded command language

First, we give an overview to the input language PRISM used to define PAs. This language has also semantics fit to describe DTMCs and CTMCs, however, this is out of the scope of this thesis.

Let in the following for a set *Var* of Boolean variables $\mathcal{V}_{Var}$ denote the set of all variable assignments. We first introduce the notions of PRISM-models, that consist of *modules* which in turn consist of *commands* that are defined over a finite set of variables.

Definition 57 (Model, module, command) *A* model *is a tuple* $(Var, s_I, \{M_1, \ldots, M_k\})$ *with*

- *a finite set of Boolean variables Var*
- *an initial assignment* $s_I \in \mathcal{V}_{Var}$
- *a finite set of* modules $\{M_1, \ldots, M_k\}$.

A module *is a tuple* $M_i = (Var_i, Act_i, C_i)$ *with*

- *a subset of variables* $Var_i \subseteq Var$ *such that* $Var_i \cap Var_j = \emptyset$ *for* $i \neq j$
- *a finite set of synchronizing actions* Act_i
- *a finite set of* commands C_i.

The action τ *with* $\tau \notin \bigcup_{i=1}^{k} Act_i$ *denotes the internal non-synchronizing action. A* command $c \in C_i$ *has the form*

$$c \;=\; [\alpha]\; g \rightarrow\; p_1 : f_1 + \ldots + p_n : f_n$$

with $\alpha \in Act_i \uplus \{\tau\}$, g *a Boolean predicate ("guard") over the variables in Var,* $p_j \in [0,1]$ *a rational number with* $\sum_{j=1}^{n} p_j = 1$*, and* $f_j : \mathcal{V}_{Var} \rightarrow \mathcal{V}_{Var_i}$ *a variable update function.*

We denote the action α of command c by act(c). Intuitively, each model contains a set of modules that might interact via the commands that have common actions. A command has a guard which is a Boolean expression over the set of variables. If this is true for a variable evaluation—a *state*—the guard can be executed.

Note that each module may only change the values of its own variables while the updated values may depend on variables of other modules. To flatten a model, each one is equivalent to a model with a single module. This is possible by defining a suitable parallel composition of these modules. We only give a short intuition on how this composition works and refer to the documentation of PRISM for more detail.

7.1.1 Parallel composition

Assume two modules $M_1 = (Var_1, Act_1, C_1)$ and $M_2 = (Var_2, Act_2, C_2)$ with $Var_1 \cap Var_2 = \emptyset$. We first define the composition $c \otimes c'$ of two commands c and c', given by $c = [\alpha]\ g \rightarrow\ p_1 \colon f_1 + \ldots + p_n \colon f_n \in C_1$ and $c' = [\alpha]\ g' \rightarrow\ p'_1 \colon f'_1 + \ldots + p'_m \colon f'_m \in C_2$. We define the operation $\otimes$ for commands, which basically builds the conjunction of the guards and the component-wise multiplication of the probabilities a well as the component-wise update of the variable update functions. In detail, for two functions $f_i \colon \mathcal{V}_{Var} \rightarrow \mathcal{V}_{Var_1}$ and $f'_j \colon \mathcal{V}_{Var} \rightarrow \mathcal{V}_{Var_2}$ we define $f_i \otimes f'_j \colon \mathcal{V}_{Var} \rightarrow \mathcal{V}_{Var_1 \cup Var_2}$ such that for all $v \in \mathcal{V}_{Var}$ we have that $(f_i \otimes f'_j)(v)(\xi)$ equals $f_i(v)(\xi)$ for each $\xi \in Var_1$ and $f'_j(v)(\xi)$ for each $\xi \in Var_2$. Altogether, we have:

$$c \otimes c' \;=\; [\alpha]\ g \wedge g' \quad\rightarrow\quad \sum_{i=1}^{n}\sum_{j=1}^{m} p_i \cdot p'_j \colon f_i \otimes f'_j\ .$$

Using this, the *parallel composition* $M = M_1 \parallel M_2 = (Var, Act, C)$ is given by $Var = Var_1 \cup Var_2$, $Act = Act_1 \cup Act_2$, and

$$\begin{aligned} C \;=\; & \{\quad c \quad | c \in C_1 \cup C_2 \wedge \mathrm{act}(c) \in \{\tau\} \cup (Act_1 \setminus Act_2) \cup (Act_2 \setminus Act_1) \quad \} \\ \cup\; & \{\ c \otimes c' | c \in C_1 \wedge c' \in C_2 \wedge \mathrm{act}(c) = \mathrm{act}(c') \in Act_1 \cap Act_2 \quad \}\ . \end{aligned}$$

Intuitively, commands labeled with non-synchronizing actions are executed individually while for synchronizing actions one command from each synchronizing module is executed simultaneously together with the others. Note that if a module has an action in its synchronizing action set but no commands labeled with this action, this module will block the execution of commands with this action in the composition. This is considered to be a modeling error and the corresponding commands are ignored.

7.1.2 PA semantics of PRISM models

The operational semantics of a model $M = (Var, Act, C)$ as defined above is a PA. As we do not consider compositional verification, we assume a model $(Var = \{\xi_1, \ldots, \xi_m\}, s_I, \{M\})$ with a single module $M = (Var, Act, C)$ which will not be subject to parallel composition any more.

The *state space* S of the corresponding PA $\mathcal{M} = (S, s_I, Act, \mathcal{P}, L)$ is given by the set of all possible variable assignments $\mathcal{V}_{Var}$, i. e., a state s is a vector $(v_1, \ldots, v_m)$ with v_i being a value of the variable $\xi_i \in Var$. To construct the transitions, we observe that the guard g of each command

$$c \;=\; [\alpha]\ g \rightarrow\ p_1 \colon f_1 + \ldots + p_n \colon f_n\ \in C$$

defines a subset of the state space $S_c \subseteq \mathcal{V}_{Var}$ with $s \in S_c$ iff s satisfies g. For each state $s \in S_c$ we

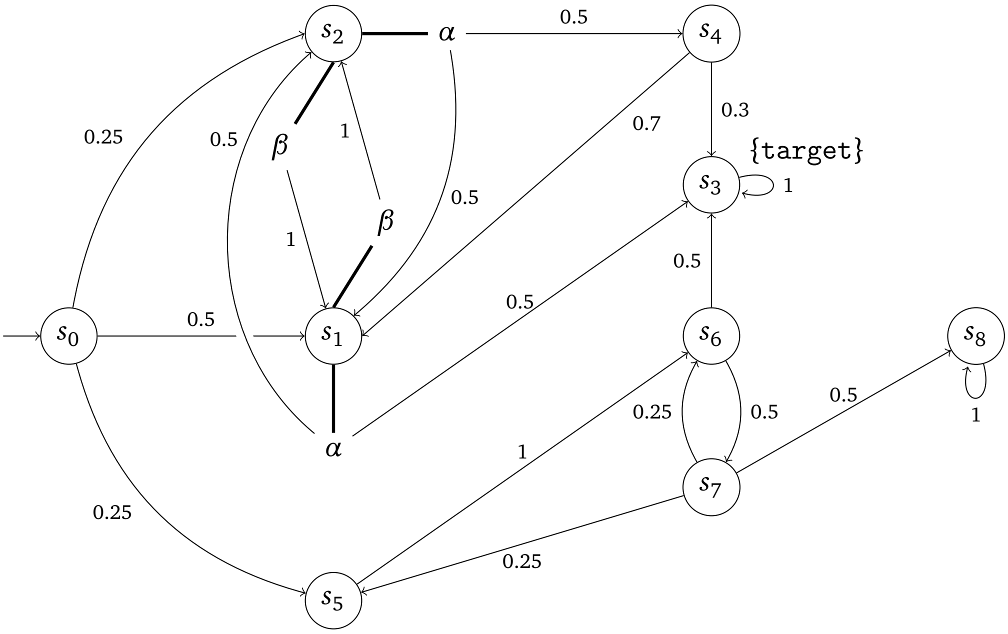


Figure 7.1: PA $\mathscr{M}$

define a probability distribution $\mu_{c,s}\colon \mathcal{V}_{Var} \to [0,1]$ with

$$\mu_{c,s}(s') = \sum_{\{1\le i\le n \mid f_i(s)=s'\}} p_i$$

for each $s' \in \mathcal{V}_{Var}$. The probabilistic transition relation $P\colon \mathcal{V}_{Var} \to 2^{Act\times Distr(\mathcal{V}_{Var})}$ is given by $P(s) = \{(\alpha,\mu_{c,s}) \mid c \in C \land \mathrm{act}(c) = \alpha \land s \in S_c\}$ for all $s \in \mathcal{V}_{Var}$.

Example 30 *We first make a connection to the running example of this thesis. Reconsider the PA (MDP) in Figure 7.1 as in the previous PA examples. We define a model $M = (Var, Act, C)$ such that $Var = \{x_1, x_2, x_3\}$ and $Act = \{\alpha, \beta\}$. As in Example 6 on Page 42 concerning the symbolic representation of a DTMC, we implicitly use the following (binary) encoding for the state space and ignore the absorbing state s_8.*

	x_1	x_2	x_3
s_0	0	0	0
s_1	0	0	1
s_2	0	1	0
s_3	0	1	1
s_4	1	1	1
s_5	1	1	0
s_6	1	0	0
s_7	1	0	1

Let in the following the updated value of variable v be denoted by v'. Accordingly, if we have a command of the form $c = [\tau]\ \neg x_1 \land x_2 \to\ 0.4\colon x_1' + 0.6\colon \neg x_1'$, this means that if the guard is true, i. e., $x_1 = \mathtt{false}$ and $x_2 = \mathtt{true}$, we set x_1 to $\mathtt{true}$ with probability 0.4 and with probability 0.6 it is set to $\mathtt{false}$, while x_2 is not changed. Consider the following set of commands:

$$
\begin{array}{ll}
[\tau]\ \neg x_1 \land \neg x_2 \land \neg x_3 & \to 0.5\colon x_3' + 0.25\colon x_2' + 0.25\colon x_1' \land x_2' \\
[\alpha]\ \neg x_1 \land \neg x_2 \land x_3 & \to 0.5\colon x_2' \land \neg x_3' + 0.5\colon x_2' \\
[\beta]\ \neg x_1 \land \neg x_2 \land x_3 & \to 1\colon x_2' \land \neg x_3' \\
[\alpha]\ \neg x_1 \land x_2 \land \neg x_3 & \to 0.5\colon \neg x_2' \land x_3' + 0.5\colon x_1' \land x_2' \land x_3' \\
[\beta]\ \neg x_1 \land x_2 \land \neg x_3 & \to 1\colon \neg x_2' \land x_3 \\
[\tau]\ \neg x_1 \land x_2 \land x_3 & \to 1\colon \mathtt{true} \\
[\tau]\ x_1 \land x_2 \land x_3 & \to 0.7\colon \neg x_1' \land \neg x_2' + 0.3\colon \neg x_1' \\
[\tau]\ x_1 \land x_2 \land \neg x_3 & \to 1\colon \neg x_2' \\
[\tau]\ x_1 \land \neg x_2 \land \neg x_3 & \to 0.5\colon \neg x_1' \land x_2' \land x_3' + 0.5\colon x_3' \\
[\tau]\ x_1 \land \neg x_2 \land x_3 & \to 0.25\colon x_2' \land \neg x_3' + 0.25\colon \neg x_3'
\end{array}
$$

This set of commands defines exactly the PA as in Figure 7.1, if $s_0 = (0,0,0)$ is the initial variable assignment. Note that we deliberately named the actions of the nondeterministic transitions α and β. If these were also τ-labeled actions, the semantics would be the same. If a command has as update only the value $1\colon \mathtt{true}$, this corresponds to a self-loop.

We also want to provide a more praxis-oriented example, taken from the PRISM benchmark suite [KNP12].

Example 31 *We consider the shared coin protocol of a randomized consensus algorithm [AH90]. The protocol returns a preference between two choices with a certain probability. A shared integer variable c is incremented or decremented by each process depending on the internal result of a coin flipping. If the value of c becomes lower than a threshold left or higher than a threshold right, the result is heads or tails, respectively.*

The protocol, which is the same for each participating process, has the following local variables: coin which is either 0 or 1, flip which is true iff the coin shall be flipped, flipped which is true iff the coin has already been flipped, check which is true iff the value of c shall be checked. Initially, c has a value between left and right, flip is true, and flipped and check are false. Consider a simplified version of the original PRISM code:

$$
\begin{array}{ll}
[\tau]\ \mathit{flip} & \to 0.5\colon \mathit{coin{=}0} \,\&\, \mathit{flip{=}false} \,\&\, \mathit{flipped{=}true} \\
 & \quad + 0.5\colon \mathit{coin{=}1} \,\&\, \mathit{flip{=}false} \,\&\, \mathit{flipped{=}true}
\end{array}
\tag{7.1}
$$

$$[\tau]\ \mathit{flipped}\,\&\,\mathit{coin}{=}0\,\&\,c{\leq}\mathit{right} \rightarrow 1:\ c{=}c{-}1\,\&\,\mathit{flipped}{=}\mathit{false}\,\&\,\mathit{check}{=}\mathit{true} \tag{7.2}$$

$$[\tau]\ \mathit{flipped}\,\&\,\mathit{coin}{=}1\,\&\,\mathit{left}{\leq}c \rightarrow 1:\ c{=}c{+}1\,\&\,\mathit{flipped}{=}\mathit{false}\,\&\,\mathit{check}{=}\mathit{true} \tag{7.3}$$

$$[\tau]\ c{<}\mathit{left} \rightarrow 1:\ \mathit{heads}{=}\mathit{true} \tag{7.4}$$

$$[\tau]\ c{>}\mathit{right} \rightarrow 1:\ \mathit{tails}{=}\mathit{true} \tag{7.5}$$

$$[\tau]\ \mathit{check}\,\&\,c{\leq}\mathit{right}\,\&\,c{\geq}\mathit{left} \rightarrow 1:\ \mathit{check}{=}\mathit{false}\,\&\,\mathit{flip}{=}\mathit{true} \tag{7.6}$$

Command 7.1 sets `coin` to 0 *or* 1*, each with probability* 0.5*. Commands 7.2 and 7.3 increment or decrement the shared counter `c` depending on the value of `coin`. Commands 7.4 and 7.5 check whether the value of `c` is above or below the boundaries `left` and `right` and return `heads` or `tails`, respectively. If no boundary is violated, Command 7.6 sets `flip` to `true` which enables Command 7.1 again.*

7.2 Computing high-level counterexamples

We are now ready to introduce our concepts of computing *smallest critical command sets*. For this, we introduce a generalization of this problem, namely *smallest critical label sets*, state the complexity, and specify an MILP formulation to solve this problem.

7.2.1 Smallest critical labelings

Let in the following $\mathcal{M} = (S, s_I, Act, \mathcal{P}, L)$ be a PA, $T \subseteq S$ a set of target states, and Lab a finite set of labels. Assume furthermore a partial labeling function $\mathcal{L} : S \times Act \times subDistr(S) \times S \rightarrow 2^{\mathrm{Lab}}$ such that $\mathcal{L}(s, \eta, s')$ is defined iff $\eta \in \mathcal{P}(s)$ and $s' \in supp(\eta)$. Recall that we abbreviate $\eta = (\alpha, \mu) \in Act \times subDistr(S)$ and write $supp(\eta) = supp(\mu)$.

We need to restrict a given PA with respect to a set of labels.

Definition 58 (Restricted PA) *Given a PA $\mathcal{M}$ and a subset of labels $\mathrm{Lab}' \subseteq \mathrm{Lab}$, the* restricted PA *induced by Lab' is given by $\mathcal{M}_{|\mathrm{Lab}'} = (S, s_I, Act, \mathcal{P}_{|\mathrm{Lab}'}, L)$ such that for all $s \in S$:*

$$\mathcal{P}_{|\mathrm{Lab}'}(s) = \{(\alpha, \mu_{|\mathrm{Lab}'}) \in Act \times subDistr(S) \mid (\alpha, \mu) \in \mathcal{P}(s) \wedge \exists s' \in S.\, L(s, \alpha, \mu, s') \subseteq \mathrm{Lab}'\}$$

$$\textit{with } \forall s' \in S.\, \mu_{|\mathrm{Lab}'}(s') = \begin{cases} \mu(s') & \textit{if } \mathcal{L}(s, \alpha, \mu, s') \subseteq \mathrm{Lab}' \\ 0 & \textit{otherwise.} \end{cases}$$

Intuitively, in the restricted PA $\mathcal{M}_{|\mathrm{Lab}'}$ all branches have been removed where the labeling is not a subset of Lab'. We now use this definition to state our central problem, where we want to have a labeling that is minimal with respect to a cost function.

Definition 59 (Smallest critical label set (SCL) problem) *Let $\mathscr{M}$, T,* Lab*, and $\mathscr{L}$ be defined as above and $\psi = \mathbb{P}_{\leq\lambda}(\lozenge T)$ be a reachability property with $\mathscr{M} \not\models \psi$. A set of labels* $\mathrm{Lab}' \subseteq \mathrm{Lab}$ *and the restricted PA $\mathscr{M}_{|\mathrm{Lab}'} = (S, s_I, Act, \mathscr{P}')$ are called* critical *if $Pr_{s_I}^{\mathscr{M}_{|\mathrm{Lab}'}}(\lozenge T) > \lambda$.*

Given a weight function $w\colon \mathrm{Lab} \to \mathbb{R}^{\geq 0}$, the smallest critical label set *(*SCL*)* problem *is to determine a critical subset* $\mathrm{Lab}' \subseteq \mathrm{Lab}$ *such that $w(\mathrm{Lab}') = \sum_{\ell \in \mathrm{Lab}'} w(\ell)$ is minimal among all critical subsets of* Lab.

Theorem 14 *To decide whether there is a critical label set* $\mathrm{Lab}' \subseteq \mathrm{Lab}$ *with $w(\mathrm{Lab}') \leq k$ for a given integer $k \geq 0$ is NP-complete.*

The proof of this theorem is a reduction from exact 3-cover (X3C) [GJ79], similar to a proof in [CV10] for Theorem 8 on Page 137 concerning the complexity of finding minimal critical subsystems. The proof is given in [1].

7.2.2 Applications of smallest critical labelings

As mentioned before, we utilize the concept of smallest critical label sets to compute different kinds of counterexamples that are minimal with respect to different quantities. We now list six instances of the SCL problem that we deem helpful in debugging a probabilistic system.

Commands Our foremost motivation was to minimize the number of involved commands of a PRISM program that together induce a critical system. We call such a set a *critical command set*. Let $\mathscr{M} = (S, s_I, Act, \mathscr{P}, L)$ be a PA generated by the modules $M_i = (\mathrm{Var}_i, Act_i, C_i)$ for $i = 1, \ldots, k$. For each module M_i and each command $c \in C_i$ we introduce a unique label $\ell_{c,i}$ having the weight $w(\ell_{c,i}) = 1$. In the following we abbreviate ℓ_c if the index i is clear from the context. We define the labeling function $\mathscr{L}\colon S \times Act \times Distr(S) \times S \rightarrow 2^{\mathrm{Lab}}$ such that the labels in $\mathscr{L}(s, \eta, s')$ correspond to the set of commands which together generate this transition $\eta \in \mathscr{P}(s)$. Note that if several command sets generate the same transition, we introduce copies of the transition. Note furthermore that synchronizing commands together may create one certain transition, i. e., a transition may be labeled by multiple commands. An SCL then corresponds to a *smallest critical command set*.

Modules A more coarse notion of a counterexample would be to minimize the number of modules involved to form a critical system. This can simply be done by using the same label for all commands in a module. This is a justified concept as many systems, e. g., in the PRISM benchmark suite [KNP12], consist partly of many copies of the same module. These modules contain exactly the same commands, only the local variables are renamed. Consider for instance a network consisting of a certain number of nodes that want to transmit messages using an access protocol, see, e. g., a typical wireless protocol [KNS02], which is ran by all nodes of the network. Additionally, there may be a module describing the communication channel. When fixing an erroneous system, one wants to preserve the identical structure of the different nodes. Therefore the selected commands should contain the same subset of commands from all identical modules. Let again $\mathscr{M} = (S, s_I, Act, \mathscr{P}, L)$ be a PA generated by the modules $M_1, \ldots, M_k$ while there exists an index $1 \leq k' \leq k$ such that $M_i = M_j$ for $1 \leq i, j \leq k'$, i. e., they are equal up to variable renaming, and $M_i \neq M_j$ for $k' < i, j \leq k$. For each module $M_i = (\mathrm{Var}_i, Act_i, C_i)$ with $i \leq k'$ and each command $c \in C_i$ we use the same label ℓ having the weight $w(\ell_{c,i}) = k'$. Contrary, for each module $M_i = (\mathrm{Var}_i, Act_i, C_i)$ with $i > k'$ we introduce labels as in the previous paragraph. We basically assign the same label to all corresponding commands from the symmetric modules and use the number of symmetric modules as its weight.

Deletion of unnecessary branches In addition to minimize the number of commands one might want to simplify commands in the sense that certain branches of a probability distribution are not necessary to render the subsystem critical. We utilize the SCL problem to delete these sort of branches. For this we identify a smallest set of command branches that need to be preserved in order for the induced sub-PA still to violate the reachability property. The resulting command branches can be removed, still yielding an erroneous system. Assume a single module $M = (Var, Act, C)$ with $C = \{c_1, \ldots, c_m\}$. Given a command c_i of the form

$$[\alpha]\ g \to p_1 \colon f_1 + p_2 \colon f_2 + \cdots + p_n \colon f_n,$$

we assign a unique label $\ell_{i,j}$ with weight $w(\ell_{i,j}) = 1$ for each command branch $p_j \colon f_j$ for $1 \leq i \leq m$ and $1 \leq j \leq n$. Let Lab be the set of all such labels. When the parallel composition of modules is computed, see Section 7.1, we build the union of the labelings of the synchronizing command branches being executed together. When computing the corresponding PA $\mathscr{M}$, we transfer this labeling to the transition branches of $\mathscr{M}$: We define the labeling function $\mathscr{L}$ such that $\mathscr{L}(s, \eta, s')$ contains the labels of all command branches that are involved in generating the branch from s to s' via the transition η.

Variable domains In PRISM-programs one can define arbitrary large but finite variable domains. This might results in unnecessary large state spaces. In particular, the part of the system inducing critical behavior can be formed by a small subset of variable evaluations. We therefore define a labeling suited to use an SCL to reduce the domains of variables in the PRISM program. Let Var be the set of variables of a model and $\mathscr{M} = (S, s_I, Act, \mathscr{P}, L)$ the corresponding PA. Note that each state $s \in S$ corresponds to an assignment of the variables in Var. For a variable $\xi \in \mathrm{Var}$ we denote the value of ξ in state s by $s(\xi)$. Let $\mathrm{Lab} = \{\ell_{\xi,v} \mid \xi \in \mathrm{Var} \wedge v \in \mathrm{dom}(\xi)\}$ be the set of labels, each with weight 1. We define the labeling of transition branches by corresponding variable values as $\mathscr{L}(s, \eta, s') = \{\ell_{\xi,v} \mid \xi \in \mathrm{Var} \wedge s'(\xi) = v\}$. A smallest critical labeling then induces a critical subsystem with a minimum number of variable values.

Variable intervals The previous reduction technique removes a maximum number of values from the variables' domains. Originally the domains are intervals in $\mathbb{Z}$. Minimization, however, yields sets that are in general not intervals anymore. We can also minimize the size of the intervals instead. To do so we need to impose further constraints on the valid label sets Lab'. Details are presented in Section 7.2.5.

States Finally, the state-minimal critical subsystems as introduced in Chapter 6 can be obtained as a special case of smallest critical label sets. We introduce a label ℓ_s with weight 1 for each state. Then we set set $\mathscr{L}(s,\eta,s') = \{\ell_{s'}\}$ for all $s \in S$, $\eta \in \mathscr{P}(s)$ and $s' \in supp(\eta)$. This corresponds to labeling each transition of the PA with the label belonging to its source state. Thereby, an SCL $\mathrm{Lab}' \subseteq \mathrm{Lab} = \{\ell_s \mid s \in S\}$ induces a state-minimal critical subsystem.

Moreover, a combination of the aforementioned applications is possible. For instance, one could first *remove all commands* which are not necessary for a violation of the property, then *remove all unnecessary branches* of the remaining commands, and finally *reduce the domains* of the variables as much as possible.

7.2.3 An MILP encoding for smallest critical labelings

In this section we give an MILP encoding that is dedicated to the SCL problem as introduced above. With the exception of the reduction of variable intervals this offers a blackbox algorithm for all concepts of high-level counterexamples listed in the previous section.

In the following, we assume an PA $\mathscr{M} = (S, s_I, Act, \mathscr{P}, L)$ where all irrelevant states and incident edges have been removed, see Definition 24 on Page 34. Moreover, recall the notions of problematic states and problematic action-distribution pairs as in Definition 56 on Page 138. Let again $S_{\mathrm{probl}(T)}$ denote the problematic states for a set of target states $T \subseteq S$. Let for all problematic states $s \in S_{\mathrm{probl}(T)}$ the set of problematic action-distribution pairs be denoted by $H_{\mathrm{probl}(T)}(s)$. Furthermore, we assume all labels that do not occur in the relevant part of the PA to be removed. We are now ready to present the encoding.

Intuition The basic idea is to compute a minimal labeling $\mathrm{Lab}' \subseteq \mathrm{Lab}$ such that the PA that is restricted to this labeling is critical. As already explained in Section 6.2 for the computation of minimal critical subsystems for PAs, the key problem is to encode the nondeterministic choices to be made into the MILP formula, which shall be resolved by a deterministic memoryless scheduler, see Definition 21 on Page 30. Again we introduce binary variables that indicate the choice of an action-distribution pair. For the particular scenario of the SCL problem, we need to explicitly "activate" or "deactivate" the probability contribution of branches $(s,\eta,s') \in S \times Act \times subDistr(S) \times S$ with respect to the inclusion of their individual set of labels. As always for PAs, the backward reachability of target states needs to be ensured for problematic states.

Variables We use the following variables. For the sake of the autonomy of this chapter we explain all variables in detail in spite of redundancy regarding the previous chapter.

$x_\ell \in \{0,1\} \subseteq \mathbb{Z}$ for each label $\ell \in \mathrm{Lab}$ is a *characteristic variable* which is assigned 1 iff ℓ shall be part of the critical label set.

$\sigma_{s,\eta} \in \{0,1\} \subseteq \mathbb{Z}$ is a binary variable for each state $s \in S \setminus T$ and each pair of action and distribution $\eta \in \mathscr{P}(s)$ that is available at s such that $\sigma_{s,\eta} = 1$ iff η is selected in state s by the scheduler that induces by the critical subsystem. Note that it is possible for the scheduler not to choose a pair η for s.

$p_{s,\eta,s'} \in [0,1] \subseteq \mathbb{R}$ for each branch (s,η,s') with $s \in S$, $\eta \in \mathscr{P}(s)$ and $s' \in supp(\eta)$ is a variable which is assigned 0 if not all labels in $\mathscr{L}(s,\eta,s')$ are contained in Lab', and at most $\eta(s')$ otherwise, i. e., the *actual probability* to move from s to s' under the condition that $\eta \in \mathscr{P}(s)$ is chosen by the scheduler.

$p_s \in [0,1] \subseteq \mathbb{R}$ for each state $s \in S$ is a variable whose value is assigned at most the *probability* to reach a target out of T starting at s under the selected scheduler within the subsystem induced by the selected label set.

$r_s^{\leftarrow} \in [0,1] \subseteq \mathbb{R}$ for all problematic states $s \in S_{\mathrm{probl}(T)}$ are used to encode the *backward reachability* of non-problematic states.

$t_{s,s'}^{\leftarrow} \in \{0,1\} \subseteq \mathbb{Z}$ are used for the reachability of non-problematic states to determine the existence of a *transition* in the MCS between problematic states $s,s' \in S_{\mathrm{probl}(T)}^{\mathscr{M}}$ where the action-distribution pair $\eta \in H_{\mathrm{probl}(T)}(s)$ is problematic.

Encoding Let in the following $w_{\min} = \min\{w(\ell) \mid \ell \in \mathrm{Lab} \wedge w(\ell) > 0\}$ be the smallest positive weight that is assigned to any label.

$$\text{minimize} \quad -\frac{1}{2}w_{\min} \cdot p_{s_I} + \sum_{\ell \in \mathrm{Lab}} w(\ell) \cdot x_\ell \tag{7.7a}$$

such that

$$p_{s_I} > \lambda \tag{7.7b}$$

$$\forall s \in T. \quad p_s = 1 \tag{7.7c}$$

$$\forall s \in S \setminus T. \quad \sum_{\eta \in \mathscr{P}(s)} \sigma_{s,\eta} \leq 1 \tag{7.7d}$$

$$\forall s \in S \setminus T. \quad p_s \leq \sum_{\eta \in \mathscr{P}(s)} \sigma_{s,\eta} \tag{7.7e}$$

$$\forall s \in S \setminus T. \, \forall \eta \in \mathscr{P}(s). \, \forall s' \in supp(\eta). \, \forall \ell \in \mathscr{L}(s,\eta,s').$$

$$p_{s,\eta,s'} \leq x_\ell \tag{7.7f}$$

$$\forall s \in S \setminus T.\, \forall \eta \in \mathscr{P}(s).\, \forall s' \in supp(\eta). \quad p_{s,\eta,s'} \leq \eta(s') \cdot p_{s'} \tag{7.7g}$$

$$\forall s \in S \setminus T.\, \forall \eta \in \mathscr{P}(s). \quad p_s \leq (1 - \sigma_{s,\eta}) + \sum_{s' \in supp(\eta)} p_{s,\eta,s'} \tag{7.7h}$$

$$\forall s \in S_{\text{probl}(T)}.\, \forall \eta \in H_{\text{probl}(T)}(s). \quad \sigma_{s,\eta} = \sum_{s' \in supp(\eta)} t^{\leftarrow}_{s,s'} \tag{7.7i}$$

$$\forall s \in S_{\text{probl}(T)}.\, \forall \eta \in H_{\text{probl}(T)}(s).\, \forall s' \in supp(\eta). \quad r^{\leftarrow}_s < r^{\leftarrow}_{s'} + (1 - t^{\leftarrow}_{s,s'}) \tag{7.7j}$$

Explanation Constraints 7.7b–7.7j describe a critical label set. First, Constraint 7.7b ensures that the probability of the initial state s_I exceeds the probability bound λ. The probability of target states is forced to be assigned 1 (Constraint 7.7c). For each state $s \in S \setminus T$ the scheduler selects at most one action-distribution pair $\eta \in \mathscr{P}(s)$. This is enforced as at most one scheduler variable $\sigma_{s,\eta} \in \mathscr{P}(s)$ can be assigned 1 (Constraint 7.7d). Note that there may be states where no transition is chosen. In this case the probability of a state is explicitly set to 0 (Constraint 7.7e).

The next two constraints enforce the correct computation of the probability contribution of the branch for $\eta \in \mathscr{P}(s)$ and s to s': If a label $\ell \in \mathscr{L}(s, \eta, s')$ is not contained in the selected subset of labels, the probability of the branch is assigned 0 (Constraint 7.7f). Otherwise this constraint is satisfied for all possible values of $p_{s,\eta,s'} \in [0,1] \subseteq \mathbb{R}$. Then Constraint 7.7g imposes an upper bound on the contribution of this branch which is defined to be the correct probability $\eta(s')$ multiplied by the probability of the successor state s'. Constraint 7.7h is trivially satisfied if $\sigma_{s,\eta} = 0$, i. e., if the scheduler does not select the current transition. Otherwise the probability p_s of state s is assigned at most the sum of the probabilities of its outgoing branches.

The reachability of at least one deadlocking or a non-problematic state is ensured by Constraints 7.7i and 7.7j. First, if a problematic action-distribution pair $\eta \in H_{\text{probl}(T)}(s)$ of a state s is selected by the scheduler then exactly one transition branch flag must be activated. Second, for all paths along activated branches of problematic transitions, an increasing order on the problematic states is enforced. Because of this order, no problematic states can be revisited on an increasing path which enforces the final reachability of a non-problematic or a deadlocking state. For more explanation to the reachability constraints see Section 6.1.2.3.

These constraints assure that each satisfying assignment of the label variables x_ℓ corresponds to a critical label set. By now minimizing the total weight of the selected labels we obtain a smallest critical label set with respect to the weight function. By the additional term $-\frac{1}{2} w_{\min} \cdot p_{s_I}$ we obtain not only a smallest critical label set but one with maximal probability as already explained in Section 6.1 for the computation of minimal critical subsystems for DTMCs,.

Formula size Let $l = |\text{Lab}|$ denote the number of labels, and—as usual—$n_{\mathscr{M}} = |S|$ the number of states of the PA $\mathscr{M}$. Let here $m_{\mathscr{M}}$ denote the number of branches $(s, \eta, s) \in S \times Act \times subDistr(S) \times S$ of PA $\mathscr{M}$. The number of integer variables in this MILP is in $O(l + m_{\mathscr{M}})$, the number of real variables in $O(n_{\mathscr{M}} + m_{\mathscr{M}})$, and the number of constraints in $O(n_{\mathscr{M}} + l \cdot m_{\mathscr{M}})$.

7.2.4 Optimizations

As for the computation of minimal critical subsystem we present a number of optimizations in the form of *redundant constraints* for the MILP, see Section 6.1.2.

7.2.4.1 Scheduler constraints

Intuition We want to exclude solutions of the constraint set for which a non-deadlocking state s has only deadlocking successors with respect to the selected scheduler. Note that such solutions would define $p_s = 0$, i. e., s does not contribute to the probability of reaching target states and will not be included in a minimal solution. Analogously, we require for each non-initial state s with a selected action-distribution pair $\eta \in \mathscr{P}(s)$ that there is a selected action-distribution pair leading to s. We call these optimizations *scheduler forward cuts* and *scheduler backward cuts*, respectively.

Remark 30 *Note that these sort of redundant constraints can also be used for the computation of minimal critical subsystems for PAs, see Section 6.2.*

Constraints

$$\forall s \in S \setminus T.\, \forall \eta \in \mathscr{P}(s) \text{ with } supp(\eta) \cap T = \emptyset. \quad \sigma_{s,\eta} \leq \sum_{s' \in supp(\eta)\setminus\{s\}} \; \sum_{\eta' \in \mathscr{P}(s')} \sigma_{s',\eta'} \tag{7.8}$$

$$\forall s \in S \setminus \{s_I\}. \quad \sum_{\eta \in \mathscr{P}(s)} \sigma_{s,\eta} \leq \sum_{s' \in \{s'' \in S\setminus\{s\} \mid \exists \eta \in \mathscr{P}(s'').\, s \in supp(\eta)\}} \; \sum_{\eta' \in \{\eta'' \in \mathscr{P}(s') \mid s \in supp(\eta'')\}} \sigma_{s',\eta'} \tag{7.9}$$

Explanation Consider Constraint 7.8 for the scheduler forwards cuts. For all states $s \in S \setminus T$ that are not target states and action-distribution pairs $\eta \in \mathscr{P}(s)$ that do not induce transitions leading to target states, it is enforced that if the corresponding scheduler variable $\sigma_{s,\eta}$ is assigned 1, for one successor s' apart from s itself there is another scheduler $\sigma_{s',\eta}$ variable selected for one action-distribution pair $\eta \in \mathscr{P}(s')$. Otherwise, the constraint is trivially true.

The other way around, see Constraint 7.9 for the scheduler backward cuts. For all states $s \in S \setminus \{s_I\}$, the sum of all action-distribution pairs $\eta \in \mathscr{P}(s)$ available at s is considered. If this sum is greater than or equal to 1, i. e., one η is selected, for states s' apart from s that are predecessors of s, a scheduler variable leading to s is selected.

As special cases of these cuts, we can simply enforce that the initial state has at least one activated outgoing transition and that at least one of the target states has a selected incoming transition. This induces that the initial state is not temporarily selected as a deadlock state and that at least one predecessor of the target states is no deadlock state. These special cuts come with very few additional constraints and often have a great impact on the solving times.

7.2.4.2 Label cuts

Intuition Intermediately, the solver might select inconsistent pairs of *label variables* and *scheduler variables*. In order to prevent this, we enforce for every selected label ℓ at least one scheduler variable $\sigma_{s,\eta}$ to be selected.

Encoding

$$\forall \ell \in \mathrm{Lab}. \quad x_\ell \leq \sum_{s \in S} \sum_{\eta \in \{\eta' \in \mathscr{P}(s) \mid \exists s' \in supp(\eta').\, \ell \in \mathscr{L}(s,\eta',s')\}} \sigma_{s,\eta} \tag{7.10}$$

Explanation For all labels $\ell \in \mathrm{Lab}$ it holds for the corresponding label variable x_ℓ that—if selected—for each state s and each action-distribution pair $\eta \in \mathscr{P}(s)$ available at s that at least one scheduler variable $\sigma_{s,\eta}$ is selected where $\ell \in \mathscr{L}(s,\eta,s')$.

7.2.4.3 Synchronization cuts

The previously introduced scheduler and label cuts are generally applicable to the smallest critical label set problem. We introduce the so-called *synchronization cuts* that are dedicated to the computation of minimal critical command sets, see Section 7.2.2.

Intuition Basically, for each synchronizing commands it is ensured that a command is selected in all other modules where there is a corresponding command.

Encoding We present the encoding on the example of two modules M_1, M_1 which synchronize on action α. Let c be a command of M_1 with this action α, and $C_{2,\alpha}$ the set of commands that also have action α in module M_2. We have the constraint

$$x_{\ell_c} \leq \sum_{d \in C_{2,\alpha}} x_{\ell_d} \,. \tag{7.11}$$

Explanation If c is selected by assigning 1 to the variable x_{l_c}, then at least one command $d \in C_{2,\alpha}$ is selected, too. Similar constraints can be formulated for minimization of command branches.

7.2.5 Reduction of variable intervals

As mentioned before, for reducing variable intervals without just reducing the domains, additional constraints are in order. For each variable $\xi \in \mathrm{Var}$, we want to minimize the interval

$$[l_\xi, u_\xi] \subseteq \mathrm{dom}(\xi) \subseteq \mathbb{Z} \,.$$

To encode these upper and lower bounds, we need to introduce for each variable $\xi \in \text{Var}$ and possible variable values $v \in \text{dom}(\xi)$ two additional variables $h^u_{\xi,v}, h^l_{\xi,v} \in \{0,1\} \subseteq \mathbb{Z}$ defining the interval.

Intuition Intuitively, these Boolean variables are assigned 1 if and only if they induce an actual upper or lower bound of the interval, not including the maximal or minimal values that are assigned. More formally, we have that $h^u_{\xi,v} = 1$ iff $v > u_\xi$ and $h^l_{\xi,v} = 1$ iff $v < l_\xi$. The remaining values $v \in \text{dom}(\xi)$ with $h^u_{\xi,v} = 0$ and $h^l_{\xi,v} = 0$ induce the interval.

Encoding The following constraints are added to the encoding for computing a minimal critical labeling, see Constraints 7.7a–7.7j.

$$\forall \xi \in \text{Var}.\, \forall v \in \text{dom}(\xi).\, v \neq \min\big(\text{dom}(\xi)\big). \quad h^l_{\xi,v} \leq h^l_{\xi,v-1} \tag{7.12a}$$

$$\forall \xi \in \text{Var}.\, \forall v \in \text{dom}(\xi).\, v \neq \max\big(\text{dom}(\xi)\big). \quad h^u_{\xi,v} \leq h^u_{\xi,v+1} \tag{7.12b}$$

$$\forall \xi \in \text{Var}.\, \forall v \in \text{dom}(\xi). \quad h^l_{\xi,v} + h^u_{\xi,v} + x_{\ell_{\xi,v}} = 1 \tag{7.12c}$$

Explanation Constraint 7.12a ensures that each for each lower bound defined by $h^l_{\xi,v} = 1$, also the lower value the value $v - 1$ is neglected, i. e., $h^l_{\xi,v-1} = 1$. The same holds for $h^u_{\xi,v}$ and the successor value $v + 1$ (Constraint 7.12b). Finally, Constraint 7.12c connects the decision variables $x_{\ell_{\xi,v}}$ for the labeling with the auxiliary variables $h^l_{\xi,v}$ and $h^u_{\xi,v}$: Exactly one of these three variables has to be set to 1—either v is below the lower bound ($h^l_{\xi,v} = 1$) or above the upper bound ($h^u_{\xi,v} = 1$), or the label $\ell_{\xi,v}$ is contained in the computed label set.

7.3 Correctness proof

For the correctness of the MILP formulation given by Constraints 7.7a–7.7j we proceed as in Section 6.3 and show that from each satisfying assignment of the MILP one can construct a minimal critical label set (soundness) and that for each critical label set there is a satisfying assignment of the MILP (completeness).

As setting we have a PA $\mathscr{M}$, a set of target states T, a labeling function L, a set of labels Lab and a PA $\mathscr{M}_{|\text{Lab}'}$ that is induced by $\text{Lab}' \subseteq \text{Lab}$. Recall that we abbreviate $\eta = (\alpha, \mu) \in \mathit{Act} \times \mathit{subDistr}(S)$.

Lemma 23 *The MILP formulation* (7.7a)–(7.7j) *is sound.*

Proof 26 Let ν be a satisfying assignment of the MILP and $\text{Lab}' = \{\ell \in \text{Lab} \mid \nu(x_\ell) = 1\}$ the induced label set. We define the scheduler $\sigma\colon S \twoheadrightarrow \mathit{Act} \times \mathit{subDistr}(S)$ by $\text{dom}(\sigma) = \{s \in S \mid \exists \eta \in \mathscr{P}(s).\, \nu(\sigma_{s,\eta}) = 1 \wedge \exists s' \in \mathit{supp}(\eta).\ L(s, \eta, s') \subseteq \text{Lab}'\}$ with respect to the assignment, i. e., for each $s \in \text{dom}(\sigma)$ we set $\sigma(s) = \eta$ with $\nu(\sigma_{s,\eta}) = 1$. Due to Constraint 7.7d there is at most one transition $\eta \in \mathscr{P}(s)$ for $\sigma_{s,\eta} = 1$ which induces a well-defined scheduler. If $s \notin \operatorname{dom}(\sigma)$, s is a deadlock state w. r. t.σ with no outgoing transition.

Let $\sigma_{|\mathrm{Lab}'} \colon S \nrightarrow \mathit{Act} \times \mathit{subDistr}(S)$ be the scheduler resulting from σ by removing branches whose labels are not included in Lab', i. e., $\operatorname{dom}(\sigma_{|\mathrm{Lab}'}) = \operatorname{dom}(\sigma)$ and for each $s \in \operatorname{dom}(\sigma_{|\mathrm{Lab}'})$ and $s' \in S$ we have

$$\sigma_{|\mathrm{Lab}'}(s)(s') = \begin{cases} \sigma(s)(s') & \text{if } L(s,\sigma(s),s') \subseteq \mathrm{Lab}' \\ 0 & \text{otherwise .} \end{cases}$$

To reduce notation, let in the following $\mathscr{D} = \mathscr{M}_\sigma$ denote the DTMC that is induced by $\mathscr{M}$ and σ, see Definition 20 on Page 30. Let U be the set of states from which T is not reachable in $\mathscr{D}$[1], D the deadlock states in U, and R the states in U whose scheduled transitions were reduced by removing some branches due to the selected label set:

$$\begin{aligned} U &= \{s \in S \mid T \text{ is unreachable from } s \text{ in } \mathscr{D}\} \\ D &= U \setminus \operatorname{dom}(\sigma_{|\mathrm{Lab}'}) \\ R &= \{s \in U \cap \operatorname{dom}(\sigma_{|\mathrm{Lab}'}) \mid \sigma(s) \neq \sigma_{|\mathrm{Lab}'}(s)\} \,. \end{aligned}$$

The reachability probabilities $q_s = Pr_s^{\mathscr{D}}(\Diamond T)$ are now computed by the linear equation system for computing reachability probabilities, see Section 2.3.1.1), using the predefined sets:

$$q_s = \begin{cases} 1 & \text{for } s \in T, \\ 0 & \text{for } s \in U, \\ \sum_{s' \in S} \sigma_{|\mathrm{Lab}'}(s)(s') \cdot q_{s'} & \text{otherwise.} \end{cases} \tag{7.13}$$

This equation system is well defined, since, if $\sigma_{|\mathrm{Lab}'}(s)$ is undefined, either s is a target state or the target states are unreachable from s. In the following we have to prove the correct probability computations:

$$\nu(p_s) = 1 \qquad \text{for } s \in T, \tag{7.14}$$

$$\nu(p_s) = 0 \qquad \text{for } s \in U, \tag{7.15}$$

$$\nu(p_s) \le q_s \qquad \text{otherwise}. \tag{7.16}$$

Thus $\nu(p_s) \le q_s$ for each $s \in S$. With (7.7b) we get $q_{s_I} > \lambda$, i. e., Lab' is critical.

It remains to show that (7.14)–(7.16) hold.

(7.14) is straightforward for target states $s \in T$ due to (7.7c).

[1] Note that the order of operations is not arbitrary here. We have $(\mathscr{M}_\sigma)_{|\mathrm{Lab}'} = (\mathscr{M}_{|\mathrm{Lab}'})_{\sigma_{|\mathrm{Lab}'}}$.

(7.15) First we observe that from all states $s \in U$ a state in $D \cup R$ is reachable: Constraints 7.7i and 7.7j ensure that from each problematic state in U we can reach a state from $D \cup R$, see Section 6.3.3.4. From the non-problematic states in U the target states T are reachable in $\mathscr{M}$ under each scheduler. Therefore, the unreachability of T from those states in $\mathscr{D}$ is due to the selected label set, where certain branches on each path leading to T are not available any more. Thus also from each non-problematic state in U we can reach a state in $D \cup R$.

Now we show that $\nu(p_s) = 0$ for all $s \in U$. Assume the opposite and let $s \in U$ with $\nu(p_s) = \xi^{\max} = \max\{\nu(p_{s'}) \,|\, s' \in U\} > 0$. Then $s \in \mathrm{dom}(\sigma_{|\mathrm{Lab}'})$ by Constraints 7.7e–7.7h, and for $\sigma_{|\mathrm{Lab}'}(s) = \eta$ we get:

$$\begin{aligned} \xi^{\max} \;=\; \nu(p_s) \;\leq\; & \sum_{s' \in supp(\eta)} \eta(s') \cdot \nu(p_{s'}) \;\leq\; \sum_{s' \in supp(\eta)} \eta(s') \cdot \xi^{\max} \\ =\; & \xi^{\max} \cdot \sum_{s' \in supp(\eta)} \eta(s') \;\leq\; \xi^{\max} \,. \end{aligned} \tag{7.17}$$

Therefore all inequalities have to hold with equality. Since $\xi^{\max}$ is assumed to be positive, this is possible only if $\sum_{s' \in supp(\eta)} \eta(s') = 1$, i. e., $s \in U \setminus R$, and $\nu(p_{s'}) = \xi^{\max}$ for all $s' \in supp(\eta)$. By induction we conclude that $\nu(p_{s'}) = \xi^{\max}$ and $s' \in U \setminus R$ for all states s' that are reachable from s under $\sigma_{|\mathrm{Lab}'}$. We know that from each $s \in U$ either a state $s' \in D$ or a state $s' \in R$ is reachable. For the former case $s' \in D$, from (7.7e)–(7.7h) we imply $\nu(p_{s'}) = 0$, contradicting to $\nu(p_{s'}) = \xi^{\max} > 0$. In the latter case $s' \in R$, the definition of R implies $\sum_{s'' \in supp(\sigma_{|\mathrm{Lab}'}(s'))} \sigma_{|\mathrm{Lab}'}(s')(s'') < 1$, contradicting to $\sum_{s'' \in supp(\sigma_{|\mathrm{Lab}'}(s'))} \sigma_{|\mathrm{Lab}'}(s')(s'') = 1$. Therefore our assumption was wrong and we have proven $\nu(p_s) = 0$ for each $s \in U$.

(7.16) Finally we show that $\nu(p_s) \leq q_s$ for each $s \in S \setminus (T \cup U)$. Constraints 7.7f–7.7h can be simplified for the chosen action $\sigma_{|\mathrm{Lab}'}(s) = \eta$ to:

$$p_s \leq \sum_{s' \in supp(\eta)} \eta(s') \cdot p_{s'} \tag{7.18}$$

Let ν_{opt} be a satisfying assignment such that $\nu_{\mathrm{opt}}(p_s)$ is maximal among all satisfying assignments (this maximum exists, since the set of satisfying assignments is compact). We claim that for all $s' \in S \setminus (T \cup U)$ reachable from s in the induced DTMC $\mathscr{D}$, Constraint 7.18 is satisfied by ν_{opt} with equality. Assume the converse is true, i. e., there is a state $s' \in S \setminus (T \cup U)$ that is reachable from s in $\mathscr{D}$ such that $\sigma_{|\mathrm{Lab}'}(s) = \eta$ and

$$0 < \varepsilon = \left(\sum_{s'' \in supp(\eta)} \eta(s'') \cdot \nu_{\mathrm{opt}}(p_{s''}) \right) - \nu_{\mathrm{opt}}(p_{s'}) \,.$$

Let $s = s_0 \eta_0 s_1 \eta_1 \ldots s_n = s'$ be an acyclic path in $\mathscr{D}$ from s to s'. We could increase the value

$v_{\mathrm{opt}}(p_{s_n})$ by at least $\varepsilon_n = \varepsilon$ (moreover, if p_{s_n} also appears on the right-hand side; note that $0 \le \eta_i(s_i) < 1$ holds for all $i = 0, \ldots, n$). This would not violate any inequality, since in the inequalities for the other states p_{s_n} appears only in upper bounds on the right-hand sides with a non-negative coefficient. Assume that, for some $i \le n$, we have increased the value of s_i by ε_i. Then the right-hand side of the inequality for s_{i-1} increases by at least $\eta_{i-1}(s_i) \cdot \varepsilon_i > 0$. Therefore we could also increase the value of $p_{s_{i-1}}$ by $\eta_{i-1}(s_i) \cdot \varepsilon_i$. This could be continued along the path back to $s = s_0$, whose value could be increased by $\varepsilon_0 = \varepsilon \cdot \prod_{i=0}^{n-1} \eta_i(s_{i+1}) > 0$. But then $v_{\mathrm{opt}}(p_s)$ would not be optimal, contradicting our assumption $\varepsilon > 0$.

This means, the inequalities for all states that are reachable from s are satisfied with equality for v_{opt}, with other words, v_{opt} encodes the solution $v_{\mathrm{opt}}(p_s) = q_s$ to (7.13). Since v_{opt} is maximal for s, all other assignments satisfy $v(p_s) \le q_s$.

It remains to show the minimality of the induced critical labeling. With $\mathrm{Lab}' = \{\ell \in \mathrm{Lab} \mid v(x_\ell) = 1\}$ and $w(\mathrm{Lab}') = \sum_{\ell \in \mathrm{Lab}'} w(\ell) = \sum_{\ell \in \mathrm{Lab}} w(\ell) \cdot v(x_\ell)$, for the objective function it holds that

$$w(\mathrm{Lab}') - w_{\min} < -\frac{1}{2} w_{\min} \cdot v(p_{s_I}) + w(\mathrm{Lab}') < w(\mathrm{Lab}') \, .$$

By minimizing the objective function, we obtain a smallest critical label set.

Lemma 24 *The MILP formulation* (7.7b)–(7.7j) *on Page 182 is complete.*

Proof 27 Let $\mathrm{Lab}' \subseteq \mathrm{Lab}$ be a critical label set. Then $Pr_{s_I}^{\mathscr{M}_{|\mathrm{Lab}'}}(\Diamond T) > \lambda$ and there is a deterministic memoryless scheduler σ for $\mathscr{M}_{|\mathrm{Lab}'}$ with $Pr^{\mathscr{D}_{|\mathrm{Lab}'}}(s_I, \Diamond T) = Pr^{\mathscr{M}_{|\mathrm{Lab}'}}(s_I, \Diamond T) > \lambda$ and $s \in \mathrm{dom}(\sigma)$ iff $Pr_s^{\mathscr{M}_{|\mathrm{Lab}'}}(\Diamond T) > 0$, for all $s \in S \setminus T$, where again $\mathscr{D}$ denotes the induced DTMC $\mathscr{M}^\sigma$ that is induced by $\mathscr{M}$ and σ.

Let $G = (V, E)$ be a directed graph with $V = \mathrm{dom}(\sigma) \cup T$ and $E = \{(s, s') \in V \times V \mid s' \in supp(\sigma(s))\}$. Now consider a smallest (edge-minimal) subgraph $G' = (V, E')$ of G containing for each state $s \in V$ a path from s to T. Due to minimality, G' is loop-free and contains for each state $s \in V \setminus T$ exactly one outgoing edge. We build the assignment:

$$v(x_\ell) = \begin{cases} 1 & \text{if } \ell \in \mathrm{Lab}', \\ 0 & \text{otherwise;} \end{cases} \qquad v(\sigma_{s,\eta}) = \begin{cases} 1 & \text{if } s \in \mathrm{dom}(\sigma) \text{ and } \sigma(s) = \eta, \\ 0 & \text{otherwise;} \end{cases}$$

$$v(p_s) = Pr_s^{\mathscr{D}_{|\mathrm{Lab}'}}(\Diamond T); \qquad v(p_{s,\eta,s'}) = \begin{cases} \eta(s') \cdot v(p_{s'}) & \text{if } \mathscr{L}(s,\eta,s') \subseteq \mathrm{Lab}', \\ 0 & \text{otherwise;} \end{cases}$$

$$\nu(t_{s,\eta,s'}) \quad = \quad \begin{cases} 1 & \text{if } s \in V \cap S_{\text{probl}(T)} \text{ and } (s,s') \in E' \text{ and } \sigma(s) = \eta, \\ 0 & \text{otherwise;} \end{cases}$$

$$\nu(r_s) \quad = \quad \begin{cases} \frac{1}{2}\nu(r_{s'}) & \text{if } s \in V \cap S_{\text{probl}(T)} \text{ and}(s,s') \in E', \\ 1 & \text{otherwise.} \end{cases}$$

We now systematically check Constraints 7.7b–7.7j:

(7.7b) is satisfied by ν because $\nu(p_{s_I}) = Pr_{s_I}^{\mathscr{D}^{\sigma}_{|\text{Lab}'}}(\Diamond T) > \lambda$.

(7.7c) holds because $Pr_{s_I}^{\mathscr{D}^{\sigma}_{|\text{Lab}'}}(\Diamond T) = 1$ for all target states $s \in T$.

(7.7d) holds since the deterministic memoryless scheduler σ selects at most one transition in each state.

(7.7e) is trivially satisfied if $\nu(\sigma_{s,\eta}) = 1$ for some $\eta \in \mathscr{P}(s)$. Otherwise, if no action is chosen, s is a deadlock state and the probability $\nu(p_s)$ to reach a target state is 0.

(7.7f) is satisfied as $\nu(p_{s,\eta,s'})$ is defined to be 0 if $\ell \in \mathscr{L}(s,\eta,s')$ for some $\ell \notin \text{Lab}'$.

(7.7g) holds by the definition of $\nu(p_{s,\eta,s'})$.

(7.7h) is trivially satisfied if $\nu(\sigma_{s,\eta}) = 0$. In case $\nu(\sigma_{s,\eta}) = 1$, the constraint reduces to $p_s \le \sum_{s' \in supp(\eta)} p_{s,\eta,s'} \le \sum_{s' \in supp(\eta)} \eta(s') \cdot p_{s'}$ with $\eta = \sigma(s)$. It is satisfied if $\nu(p_s) = 0$. Otherwise, since $\nu(p_s)$ is the reachability probability of T in $\mathscr{M}^{\sigma}_{|\text{Lab}'}$, it satisfies the following equation, see Lemma 6 on Page 151.

$$\nu(p_s) = \sum_{s' \in supp(\eta)} \eta(s') \cdot \nu(p_{s'}) = \sum_{s' \in supp(\eta)} \nu(p_{s,\eta,s'}).$$

Note that $\mathscr{M}_{|\text{Lab}'}$ contains exactly those branches (s,η,s') of $\mathscr{D}$ for which $\mathscr{L}(s,\eta,s') \subseteq \text{Lab}'$ and therefore $\nu(p_{s,\eta,s'}) = \eta(s') \cdot \nu(p_{s'})$. For all other branches (s,η,s') in $\mathscr{M}^{\sigma}$, but not in $\mathscr{D}_{|\text{Lab}'}$, $\nu(p_{s,\eta,s'}) = 0$ holds. Hence we have $\nu(p_s) = \sum_{s' \in supp(\eta)} \nu(p_{s,\eta,s'})$ and (7.7h) is satisfied.

(7.7i) holds if $\nu(\sigma_{s,\eta}) = 0$, since in this case by definition either $s \notin \text{dom}(\sigma)$ or $\eta \neq \sigma(s)$ and therefore $\nu(t_{s,\eta,s'}) = 0$ for all $s' \in S$. Otherwise $\nu(\sigma_{s,\eta}) = 1$, i. e., $\sigma(s) = \eta$. By the construction of G' there is exactly one $s' \in supp(\eta)$ with $\nu(t_{s,\eta,s'}) = 1$.

(7.7j) is straightforward if $\nu(t_{s,\eta,s'}) = 0$. Otherwise by definition $r_s = \frac{1}{2} r_{s'}$, and since $\nu(r_s), \nu(r_{s'}) > 0$, the inequation holds.

Theorem 15 *The MILP encoding* (7.7a)–(7.7j) *is sound and complete.*

This theorem follows directly by the correctness of Lemma 23 and Lemma 24.

CHAPTER 8

Implementation and experiments

To demonstrate the feasibility of the approaches presented in this thesis, we now report on our implementations and experimental results. We first describe the open source tool `COMICS` and the set of benchmarks. Sectioned according to the previous chapters describing our theoretical approaches we then show the results for model checking or computing counterexamples, respectively. We also describe each implementation that is not part of `COMICS` in the corresponding sections. We use this chapter to demonstrate the general feasibility of our methods; for more extensive experimental results, we refer to the corresponding publications.

8.1 The COMICS Tool – Computing Minimal Counterexamples for DTMCs

Basically, `COMICS` provides an implementation of the *hierarchical counterexample generation* as schematically depicted in Figure 5.3 on Page 90. As a special case, also the *incremental generation of critical subsystems* as in Figure 5.4 on Page 93 is integrated. The tool was published in [6, 14]. Note that the implementation of the *symbolic counterexample generation* as described in Section 5.7 is not part of this publication but was developed as an extension.

The tool can be used either as a command-line tool or with a GUI, the latter allowing the user to actively influence the process of finding hierarchical counterexamples. The user may select *exact* or *floating point* arithmetics for the computations. The program consists of approximately 20 000 lines of code in five main components whose interaction is depicted in Fig. 8.1. The GUI is implemented in Java, all other components in C++.

Let us now list the basic functionalities. The `GUI` is presented afterwards.

`SCCModelCheck` performs *SCC-based model checking* for an input DTMC and returns an abstract

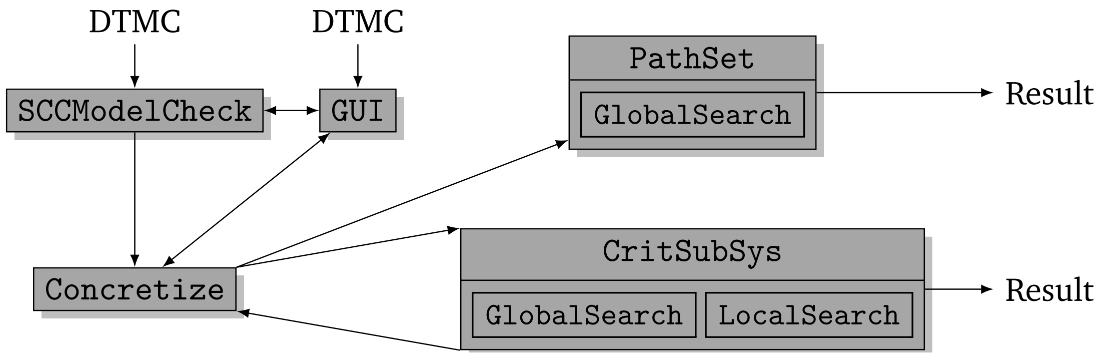


Figure 8.1: Architecture of COMICS

DTMC to Concretize or to GUI (see Chapter 4).

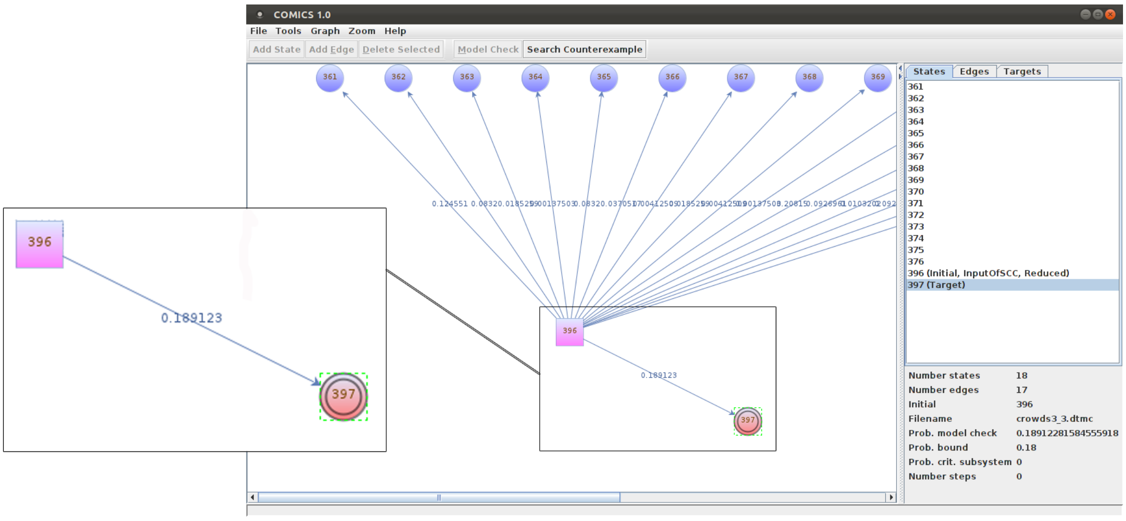
Concretize selects and concretizes some states, either automatically or user-guided via the GUI (see Section 5.2.1). Heuristics for the number of states to concretize in a single step as well as for the choice of states are offered. It is also possible to predefine the number of concretization steps.

CritSubSys can be invoked on the modified system to compute a critical subsystem (see Definition 50 on Page 82) using the global search approach (see Section 5.5.1) or the local/fragment search (see Section 5.5.2). The result is given back to Concretize for further refinement or returned as the result.

PathSet invokes the global search and yields a set of paths representing a *minimal counterexample* [HKD09] (see Section 3.1.1).

The GUI provides a *graph editor* for specifying and modifying DTMCs. A large number of *layout algorithms* increase the usability even for large graphs. Both concrete and abstract graphs can be *stored*, *loaded*, *abstracted*, and *concretized* by the user. As the most important feature, the user is able to *control the hierarchical concretization* of a counterexample. If an input graph seems too large to display, the tool offers to operate without the graphical representation. In this case the abstract graph can be computed and refined in order to reduce the size. Figure 8.2 shows one abstracted instance of the CROWDS *protocol* benchmark [RR98], where the probability of reaching the unique target state is displayed in the information panel on the right as well as on the transition leading from the initial state to the target state. The initial state is abstract and can therefore be expanded.

For performance tests, several predefined *benchmarking options* are provided for the command-line version.



361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
396 (Initial, InputOfSCC, Reduced)
397 (Target)

Number states	18
Number edges	17
Initial	396
Filename	crowds3_3.dtmc
Prob. model check	0.18912281584555918
Prob. bound	0.18
Prob. crit. subsystem	0
Number steps	0

Figure 8.2: Screenshot of COMICS's GUI with an instance of the CROWDS protocol

8.2 Experiments

First, we present the different case studies that are used in this thesis. In general, the properties we are investigating all reduce to reachability properties, see Section 2.3.1, of the form $\Pr_{\leq\lambda}(\Diamond T)$. We therefore always assume a set of target states that shall be reached with at most probability $\lambda \in \mathbb{Q}$. For the approaches on counterexample generation our goal is to *exceed* this λ. If we use a benchmark only for testing model checking or parametric model checking, no λ is listed. All benchmarks are available at the PRISM-webpage [PRI10] or as part of the PRISM benchmark suite [KNP12]. We also used PRISM to generate the explicit state space of each instance. For some approaches, we utilize the MILP solver Gurobi version 5.6 [Gur13], see Section 2.6.3.

Unless stated otherwise, we defined a timeout of 3600 seconds denoted by TO and a memory out of 4 GB denoted by MO. The experiments were run on an Intel Core i7® CPU with 4 GHz and 16GB of main memory under Ubuntu 12.04. For the high-level counterexamples we used a different setting which is explained later.

8.2.1 Discrete-time Markov chains

The crowds protocol is dedicated to anonymous web browsing [RR98]. Each node sends a packet with probability p_f directly to the target node and with probability $1 - p_f$ to a randomly chosen node in the crowd. With probability p_{bad} a member is corrupt and tries to identify the sender of a packet. For the non-parametric methods, we have the instantiations $p_f = 0.8$ and $p_{bad} = 0.02$. The parameter R denotes the number of rounds in which packets are sent. N is the number of non-corrupt crowd members. The model statistics for each instance CROWDS-N-K are given in Table 8.1. Our property states that a sender is identified by a corrupt member at least once. If this property has a high probability $\Pr(\Diamond T)$ for T being a set of target states labeled with

Model-N-K	$\lvert S\rvert$	$\lvert E\rvert$	$\Pr(\Diamond T)$	λ
crowds-5-4	3 515	6 035	0.235	0.23
crowds-5-6	18 817	32 677	0.427	0.4
crowds-5-8	68 740	120 220	0.591	0.59
crowds-5-17	2 888 761	5 127 151	0.933	–
crowds-12-6	829 669	2 166 277	0.332	0.1
crowds-10-20	4 163 510 716	10 172 513 716	0.931	0.4
crowds-20-20	10 173 177 100 089 080	38 403 575 234 221 120	??	0.2
sleader-4-4	782	1 037	1.0	0.5
sleader-4-6	3 902	5 197	1.0	0.5
sleader-4-8	12 302	16 397	1.0	0.5
sleader-8-4	458 847	524 382	1.0	0.5
nand-5-2	1 728	2 505	0.611	0.2
nand-5-3	2 526	3 639	0.611	0.2
nand-5-4	3 324	4 773	0.611	0.2
nand-25-2	347 828	541 775	0.565	0.2
brp-32-2	1 349	1 731	$2.61{\cdot}10^{-5}$	–
brp-128-2	5 381	6 915	$2.64{\cdot}10^{-5}$	–
brp-512-2	21 509	27 651	$2.64{\cdot}10^{-5}$	–
brp-1024-2	43 013	55 299	$2.58{\cdot}10^{-5}$	–
brp-2048-4	139 271	184 323	$2.35{\cdot}10^{-5}$	–
brp-8192-8	983 051	1 327 107	$1.58{\cdot}10^{-5}$	–
cont-5-2	33 790	34 813	0.52	0.5
cont-5-8	156 670	157 693	0.52	0.5
cont-7-2	737 278	753 661	0.50	0.48
cont-7-4	1 654 782	1 671 165	0.50	0.48

Table 8.1: Model statistics for the DTMC benchmarks

the atomic proposition `identified`, the protocol is considered to be faulty. Note that we were not able to determine the model checking probability of crowds-20-20 using `PRISM` due to the system size, this is indicated by "??" in the corresponding table.

Synchronous leader election is a *leader election protocol* [IR90]. Its purpose is to identify a leader node in a symmetric *synchronous* network ring of N participants. Each node randomly chooses a value from $\{1,\ldots,K\}$ and sends it around the ring. The node with the highest unique number becomes the leader. If there is no unique number, a new round starts. We measure the probability of the property which describes if a leader is finally elected, denoted by $\Pr(\Diamond T)$ with a set of target states T that are labeled with the atomic proposition `elected`. Each instance `sleader`-N-K is parametrized by N and K. The instances we use are listed in Table 8.1 together with the number of states $|S|$, the number of transitions $|E|$, and the probability to reach target states. Although some of our approaches seem to be capable of handling larger instances of this protocol, we were not able to build these models due to an internal error of `PRISM` (e. g., SLEADER-8-5 or SLEADER-4-9).

NAND multiplexing describes a redundancy technique. Basically, it concerns constructing reliable computation from unreliable components [vN56, NPKS05]. The model operates in stages, each of which contains N NAND gates that are all doing the same job. Probabilities model the faultiness of the units ($p_{err} = 0.02$) and if an input is erroneous ($p_{in} = 0.9$). For parametric models, these probabilities are left unspecified. K is the number of stages. We check the property that never a reliable state is reached for a dedicated set of target states T. NAND instances are denoted by NAND-N-K, see Table 8.1.

The bounded retransmission protocol is dedicated to transferring files over unreliable networks [DJJL01, DKRT97]. A file consists of N chunks. During the transfer to a target node, chunks might get lost. Therefore each chunk is re-transferred up to K times until the target node has received it properly and the sender node has obtained an acknowledgment thereof. The unreliability of those channels is defined by probability $p_{loss} = 0.02$ which is left unspecified for parametric models. We check the property that the sender is unsure whether the target node has successfully received the file, which is again modeled by a reachability property. Instances of this protocol are denoted by BRP-N-K (Table 8.1).

Probabilistic Contract Signing is a network protocol targeting the *fair* exchange of critical information between two parties A and B. In particular, whenever B has obtained A's commitment to a *contract*, B should not be able to prevent A from getting B's commitment. The reachability property we are investigating describes an unfair situation where A knows B's secrets while B doesn't know A's secrets. Each instance of CONT-N-K is given by the number N of data pieces to exchange and by the size K of each data piece (Table 8.1).

8.2.2 Markov decision processes

The consensus shared coin protocol shall establish agreement between N asynchronous processes [AH90]. The processes access a global counter which is increased or decreased in dependence of a coin flipping which is performed when a process enters the protocol. Depending on the current value of the counter and the values of N and a given constant K, the process decides whether it agrees or not. The protocol proceeds in rounds as long as no agreement is achieved. As different processes may try to access the protocol at the same time, it is nondeterministically decided which process may flip a coin. The property to investigate is that all processes have flipped their coin and made their decision, model-wise given by reaching a set of dedicated target states. Instances are denoted by CONSENSUS-N-K, see Table 8.2.

The CSMA communication protocol concerns the IEEE 802.3 CSMA/CD network protocol which aims at minimizing data collision, i. e., the simultaneous use of a common channel [NSY92,

Model-N-K	$\lvert S\rvert$	$\lvert E\rvert$	$\Pr(\lozenge T)$	λ
CONSENSUS-2-1	144	208	1.0	0.1
CONSENSUS-2-2	272	400	1.0	0.1
CONSENSUS-2-4	528	784	1.0	0.1
CSMA-2-2	1038	1054	1.0	0.1
CSMA-2-4	7958	7988	1.0	0.1
CSMA-2-6	66718	66788	1.0	0.1
WLAN-0-1	3123	4186	1.0	0.5
WLAN-0-2	6063	8129	0.184	0.1
WLAN-0-5	14883	19958	0.00114	0.0005
WLAN-1-1	8742	11493	1.0	0.5
WLAN-2-1	28597	37119	1.0	0.5
WLAN-2-2	28598	37120	0.184	0.1

Table 8.2: Model statistics for the MDP benchmarks

KNS03]. N is the number of processes that want to access the common channel, K is the maximal value of a backoff limit for each process, i. e., the waiting time for retransmission. We check a property expressing all stations successfully sending their messages before a collision with maximal backoff occurs for each instance CSMA-N-K, see Table 8.2.

Firewire models the Tree Identify Protocol of the IEEE 1394 High Performance Serial Bus (called "FireWire") [Sto03]. This is again a leader election protocol which is executed each time a node enters or leaves the network. The parameter N denotes the delay of the wire as multiples of 10 ns. We check the probability of finally electing a leader. The instances of the protocol are only parametrized by N, i. e., we have FW-N, see Table 8.2.

8.2.3 SCC-based model checking

In this section, the model checking performance of `COMICS` is tested for non-parametric and parametric models.

8.2.3.1 Model checking of DTMCs

We first measure the performance of the SCC-based model checking [11] as implemented in `COMICS` [10] for non-parametric benchmarks. For details we refer to Section 4. We compare the running times and the memory consumption to that of `PRISM 4.0` [KNP11]. The timeout is set to 3600 seconds and a memory out of 4 GB. The results for a selection of DTMC benchmarks are depicted in Table 8.3. In each row the best running times are printed boldfaced. Note that we rounded the values such that the number of decimal places is two. Although this model checking approach was developed as a preprocessing step for the hierarchical counterexample generation, see Section 5.2, it proofs competitive for benchmarks with up to 100 000 states. For benchmarks where the model checking probability is very small, as in BRP, it even performs better. This is due to the fact, that on an abstract level of the SCC decomposition all paths leading to absorbing non-target states are ignored.

8.2.3.2 Model checking of PDTMCs

	COMICS		PRISM	
Model	Time	Mem.	Time	Mem.
BRP-32-2	**0.00**	1.93	0.07	66.23
BRP-128-2	**0.03**	5.94	0.34	74.10
BRP-512-2	**0.48**	19.07	2.13	77.18
BRP-1024-2	**1.87**	36.93	6.49	82.66
BRP-2048-4	73.88	117.62	**33.67**	90.16
BRP-8192-8	TO	–	**1291.77**	128.24
CROWDS-5-4	0.05	3.58	**0.04**	66.15
CROWDS-5-6	1.20	15.69	**0.09**	67.53
CROWDS-5-8	19.72	49.88	**0.28**	70.73
CROWDS-5-17	TO	–	**15.65**	141.55
CROWDS-12-6	TO	–	**2.69**	94.64
SLEADER-4-4	**0.00**	1.43	0.02	67.79
SLEADER-4-6	**0.01**	3.78	0.12	79.13
SLEADER-4-8	**0.12**	10.10	0.46	96.42
SLEADER-8-4	193.79	341.56	**15.97**	411.04
NAND-5-2	**0.01**	2.30	0.03	64.56
NAND-5-3	**0.01**	2.93	0.05	64.75
NAND-5-4	**0.02**	3.50	0.06	64.85
NAND-25-2	1594.02	316.76	**5.07**	87.22

Table 8.3: SCC-based model checking in comparison to PRISM

By means of an extension of COMICS, we implemented our approaches to parametric probabilistic model checking [2], see Section 4.3. The implementation is a C++ prototype using the arithmetic library GiNaC [BFK02]. Moreover, we implemented the state-elimination approach used by PARAM [HHWZ10] using our optimized factorization approach to provide a more distinct comparison.

The experimental setting includes our SCC-based approach (COMICS–SCC MC) using an optimized factorization of polynomials, see again [2], the state elimination as in PARAM but also using the factorization of polynomials, and the PARAM tool itself. Note that no bisimulation reduction was applied to any of the input models, which would improve the feasibility of all approaches likewise. For all instances and each tool we list the running time in seconds and the memory consumption in MB; the best time is **boldfaced**. Moreover, for our approaches we list the number of polynomials which are intermediately stored.

For BRP, COMICS–STATE ELIM always outperforms PARAM and COMICS–SCC MC by up to two orders of magnitude. On larger instances, COMICS–SCC MC is faster than PARAM while on smaller ones PARAM is faster and has a smaller memory consumption.

In contrast, the crowds protocol always induces a nested SCC structure, which is very hard for PARAM since many divisions of polynomials have to be carried out. On larger benchmarks, it is therefore outperformed by more than three orders of magnitude while COMICS–SCC MC performs best. This is actually measured by the timeout; using PARAM we could not retrieve results for larger instances.

To give an example where PARAM performs mostly better than our approaches, we consider NAND. Its graph is acyclic consisting mainly of single paths leading to states that have a high number of outgoing edges, i. e.many paths join at these states and diverge again. Together with a large number of different probabilities, this involves the addition of many polynomials, whose factorizations are completely stored. The SCC approach performs better here, as for acyclic graphs just the linear equation system is solved, as described in Section 4.3. This seems to be superior to the state elimination as implemented in our tool. We don't know about PARAM's interior for these special cases. As a solution, our implementation offers the possibility to limit the number of stored polynomials, which decreases the memory consumption at the price of losing information about the factorizations. However, an efficient strategy to manage this bounded pool of polynomials is not yet implemented. Therefore, we refrain from presenting experimental results for this scenario.

	COMICS–SCC MC			COMICS–STATE ELIM			PARAM	
Model	Time	Poly	Mem.	Time	Poly	Mem.	Time	Mem.
BRP-32-2	**0.10**	1163	7.92	0.16	3287	15.08	1.83	7.22
BRP-128-2	**4.81**	4619	47.03	5.34	13367	163.13	166.58	63.19
BRP-512-2	565.92	18443	776.48	**303.25**	53687	3091.43	TO	–
CROWDS-5-4	0.51	894	7.22	**0.30**	1530	7.20	12.80	6.45
CROWDS-5-6	**2.25**	2446	17.41	2.90	4977	16.84	266.82	20.67
CROWDS-5-8	**8.78**	6139	50.04	19.74	12633	48.33	2695.12	66.76
CROWDS-12-6	TO	–	–	**1722.83**	14271	643.07	TO	–
CROWDS-5-17	**312.16**	110078	2463.87	TO	–	–	TO	—
NAND-5-2	0.57	2975	18.59	0.49	16042	29.25	**0.17**	4.70
NAND-5-3	0.69	4088	30.78	0.94	25337	48.92	**0.29**	5.48
NAND-5-4	1.35	5201	50.62	1.73	34583	77.94	**0.47**	5.84
NAND-25-2	–	–	MO	–	–	MO	TO	–

Table 8.4: Model checking PDTMCs comparing COMICS and PARAM

8.2.4 Counterexample generation using DiPro

We tested several algorithms of DiPro [ALFLS11] for our benchmark set, namely the extended best-first search (XBF), the Eppstein algorithm and the K^* algorithm. For the generated counterexamples, we present in Table 8.5 for each algorithm the size of the critical subsystem in terms of the number of states ($|S_{min}|$), its probability (Prob.), the computation time and the memory. The best results in terms of the size and the time are printed boldface.

	XBF				Eppstein				K*			
Model	$\lvert S_{min}\rvert$	Prob.	Time	Mem.	$\lvert S_{min}\rvert$	Prob.	Time	Mem.	$\lvert S_{min}\rvert$	Prob.	Time	Mem.
CONT-5-2	**13311**	0.5	1182.79	641.34	13515	0.5	**60.40**	827.91	14029	0.516	145.01	621.96
CONT-5-8	–	–	–	MO	**75075**	0.5	**266.34**	989.39	–	–	–	MO
CONT-7-2	–	–	TO	–	–	–	TO	–	–	–	TO	–
CONT-7-4	–	–	TO	–	–	–	TO	–	–	–	TO	–
CROWDS-5-4	**2669**	0.233	**12.01**	130.41	–	–	TO	–	2722	0.230	522.65	699.73
CROWDS-5-6	**10793**	0.401	**93.30**	204.14	–	–	TO	–	–	–	TO	–
CROWDS-5-8	**63334**	0.590	**3376.39**	655.80	–		TO	–	–		TO	–
SLEADER-4-4	1308	0.667	3.83	129.08	**920**	0.5	**1.57**	124.71	1025	0.559	4.88	137.89
SLEADER-4-6	8256	0.900	14.20	261.94	**4560**	0.500	**6.52**	283.41	**4560**	0.500	17.79	283.95
SLEADER-4-8	14512	0.506	33.82	454.00	**14360**	0.5	**29.75**	438.62	**14360**	0.5	86.72	507.86
SLEADER-8-4	–	–	TO	–	–	–	TO	–	–	–	TO	–
NAND-5-2	**203**	0.210	1.38	86.70	**203**	0.210	1.58	1111.68	**203**	0.210	**1.13**	87.24
NAND-5-3	450	0.351	2.46	98.48	**375**	0.203	**2.06**	131.00	399	0.310	4.10	106.80
NAND-5-4	507	0.250	**2.18**	103.79	**479**	0.203	2.61	154.66	**479**	0.253	4.44	115.38
NAND-25-2	**7420**	0.102	**81.17**	203.24	–	–	TO	–	–	–	TO	–

Table 8.5: Counterexample generation using DiPro

Summarizing, for CONT Eppstein performs best while DiPro is only able to generate critical subsystems for the smaller instances. On CROWDS, only XBF is able to generate counterexamples. SLEADER and NAND are handled best by Eppstein, while all instances can be solved within the time limit.

8.2.5 Hierarchical counterexample generation

To show the feasibility of the hierarchical counterexample generation [10], see Section 5.2, we proceed as follows: SCC-based model checking is invoked on a benchmark yielding an abstract DTMC. On this DTMC, COMICS computes a critical subsystem, concretizes heuristically chosen states and again computes a critical subsystem. This is iterated until the *concrete subsystem* results. Note that using our tool properly, a user would decide at an earlier stage of this process that already enough debugging information is present. The results for our benchmark set are depicted in Table 8.6.

Both for the local search and for the global search approach we present the size of the critical subsystem in terms of the number of states ($|S_{min}|$), its probability (Prob.), the computation time and the memory. The best results in terms of the size and the time are printed boldface. We observe that for graphs having a complicated loop structure as for the CROWDS-benchmark, the local search is superior both in running times as in the size of the critical subsystem. This is due to the fact, that the global search will list many paths that differ only in the number of unrollings

	global search				local search			
Model	$\lvert S_{\min}\rvert$	Prob.	Time	Mem.	$\lvert S_{\min}\rvert$	Prob.	Time	Mem.
CONT-5-2	6824	0.5	**0.09**	27.04	**6657**	0.5	11.56	167.18
CONT-5-8	37603	0.5	**1.53**	126.69	**37377**	0.5	68.16	905.49
CONT-7-2	**134551**	0.480	**96.13**	590.39	–	–	TO	–
CONT-7-4	–	–	TO	–	–	–	TO	–
CROWDS-5-4	**898**	0.230	18.36	842.10	976	0.230	**0.81**	12.88
CROWDS-5-6	–	–	–	MO	**5843**	0.4	**118.52**	47.86
CROWDS-5-8	–	–	–	MO	**24675**	0.59	**2425.91**	3094.04
SLEADER-4-4	**456**	0.5	**0.015**	2.93	459	0.5	0.06	5.49
SLEADER-4-6	**2100**	0.5	**0.28**	6.69	2103	0.5	1.76	73.70
SLEADER-4-8	**6422**	0.500	**0.90**	18.15	6423	0.500	23.61	679.02
SLEADER-8-4	**229431**	0.5	**583.60**	?	–	–	TO	–
NAND-5-2	**102**	0.210	**0.001**	2.64	**102**	0.210	0.001	2.64
NAND-5-3	185	0.203	**0.002**	3.59	**168**	0.203	0.02	3.51
NAND-5-4	234	0.203	**0.004**	4.51	**222**	0.206	0.06	4.37

Table 8.6: Hierarchical counterexample generation

of the same loop. For tree-like or directed-acyclic graphs, the global search performs better, as the number iterations will be asymptotically equal while each iteration of the global search is less complex.

8.2.6 Explicit counterexample generation

We now basically perform the same tests as for the hierarchical counterexample generation. The only difference is, that we start directly with a concrete graph. An interesting observation is, that the hierarchical approach using the fragment search seems to perform better than using the local (fragment) search directly on the concrete graph. That might be due to the fact, that certain branches of the graph are excluded on an abstract level avoiding to explore them in detail on the concrete graph. In general, and as expected, the running times are superior to the ones when performing the hierarchical concretization.

8.2.7 Symbolic counterexample generation

The symbolic counterexample generation is dedicated to enable the handling of very large DTMCs and is therefore the only approach that can handle the largest benchmark instances we used. In Table 8.8, we present results for the approach concerning bounded model checking, see Section 5.6 and the ones concerning BDD-based graph algorithms, see Section 5.7. For the former ones, we list results for the global search (Section 5.6.1), the fragment search (Section 5.6.2) as well as the fragment search together with the heuristics for choosing more probable paths (Section 5.6.3). For the BDD-based approaches, we give results for the adaptive global search (Section 5.7.2) and the adaptive symbolic fragment search (Section 5.7.3). For all methods, we

	global search				fragment search			
Model	$\lvert S_{\min}\rvert$	Prob.	Time	Mem.	$\lvert S_{\min}\rvert$	Prob.	Time	Mem.
CONT-5-2	6822	0.5	**0.07**	30.96	**6657**	0.5	10.72	172.14
CONT-5-8	37601	0.5	**1.30**	143.24	**37377**	0.5	62.63	922.31
CONT-7-2	**134545**	0.480	**81.44**	655.24	–	–	–	MO
CONT-7-4	**354825**	0.480	**217.75**	1491.41	–	–	–	MO
CROWDS-5-4	1071	0.230	**0.52**	104.29	**900**	0.230	1.41	35.01
CROWDS-5-6	5248	0.400	**15.06**	3554.73	**3260**	0.400	49.01	295.15
CROWDS-5-8	–	–	–	MO	–	–	–	MO
CROWDS-12-6	**591**	0.100	**1.15**	779.43	–	–	TO	–
SLEADER-4-4	**395**	0.5	**0.005**	1.85	462	0.505	0.06	4.93
SLEADER-4-6	**1957**	0.5	**0.09**	5.57	1962	0.500	1.44	59.84
SLEADER-4-8	**6157**	0.5	**0.17**	15.48	6423	0.5	21.50	577.16
SLEADER-8-4	**229431**	0.5	**586.51**	516.33	–	–	–	MO
NAND-5-2	**102**	0.210	**0.0007**	2.45	**102**	0.210	0.0008	2.46
NAND-5-3	185	0.203	**0.001**	3.34	**168**	0.203	0.02	3.26
NAND-5-4	234	0.203	**0.003**	4.23	**222**	0.206	0.05	4.07

Table 8.7: Concrete counterexample generation using `COMICS`

list the number of states and the probability of the subsystem, the computation time in seconds and the memory consumption in MB. The results that were completed within the timeout of 3600 seconds and didn't exceed the bound of 4 GB on the memory consumption, are printed boldfaced. If the computation time was exceeded, we give the intermediate results.

The results in Table 8.8 show that the adaptive BDD-based global and fragment search significantly outperform all other symbolic approaches on our benchmark sets. To evaluate the limits of adaptive BDD-based search strategies, we generated the instance CROWDS/20-30 with more than 10^{16} states and $3.8 \cdot 10^{16}$ transitions. Adaptive BDD-based fragment search computed, for a probability bound $\lambda = 0.2$, a subsystem with 76 007 states, and probability 0.208446 within 2972.36 seconds using less than 873 MB of main memory. The adaptive global search returned a subsystem with 82 944 states and probability 0.207726 within 2497.89 seconds, using roughly the same amount of memory.

None of the explicit-state tools was able to handle this instance, as we were not even able to store the explicit state space on hard disk. `DiPro` did not immediately fail due to the limited memory, but ran into a timeout with all three search methods.

When comparing the SAT- and the BDD-based approaches one can recognize that the former performs much worse. The SAT-based approaches ignore the actual transition probabilities, while the BDD-based approaches always compute the most probable paths. Therefore the BDD-based methods need in general fewer paths to reach a critical subsystem. Not only the computation time is higher for the SAT-based approaches, but also the memory consumption: for each found path an additional clause has to be added to the solver's clause database to exclude it from the search space. Moreover a large number of conflict clauses is computed by the solver during the search process, which significantly contribute to the memory consumption.

For a more detailed evaluation of the symbolic approaches for counterexample generation, we refer to [3].

Model	Adaptive BDD global	Adapt. BDD fragment	SAT global	SAT fragment	SAT fragment + H
	# states	# states	# states	# states	# states
	Prob.	Prob.	Prob.	Prob.	Prob.
	Time	Time	Time	Time	Time
	Mem.	Mem.	Mem.	Mem.	Mem.
CROWDS-5-4	**830**	**3190**	735	**1071**	**1071**
	0.231	**0.234**	0.216	**0.234**	**0.234**
	2.04	**1.3**	TO	**63.56**	**197.66**
	21	**19**	475	**80**	**87**
CROWDS-12-6	**1027**	**436**	**492**	3891	5351
	0.106	**0.1**	**0.1**	0.04	0.062
	6.02	**32.08**	**515.91**	TO	TO
	51	**51**	**202**	544	562
CROWDS-10-20	**167157**	**28771**	1394	8249	2229
	0.40	**0.40**	0.15	0.04	0.04
	43.49	**71.59**	TO	TO	TOt
	178	**178**	1139	1708	854
CONT-5-2	**7010**	**6995**	**6684**	**6684**	**6684**
	0.51	**0.507**	**0.501**	**0.501**	**0.501**
	0.15	**0.15**	**23.11**	**2494.83**	**4044.64**
	33	**33**	**184**	**2366**	**2374**
CONT-7-2	**141060**	**141029**	**139302**	16535	19833
	0.50	**0.502**	**0.501**	0.05	0.06
	0.32	**0.29**	**495.70**	TO	TO
	41	**41**	**314**	6560	7535
CONT-7-4	**372228**	**372197**	**368706**	13985	3196
	0.50	**0.502**	**0.501**	0.01	0.004
	1.18	**1.15**	**2759.95**	TO	TO
	88	**87**	**1202**	14295	4021
SLEADER-4-8	**6161**	**6160**	**6161**	**6297**	
	0.50	**0.50**	**0.50**	**0.50**	
	1325.39	**119.86**	**512.15**	**1662.90**	
	238	**238**	**4587**	**4771**	MO
SLEADER-8-4	6742	9257			
	0.02	0.02			
	TO	TO			
	1632	1632	MO	MO	MO

Table 8.8: Symbolic counterexample generation

8.2.8 Minimal critical subsystems

This section concerns the generation of minimal critical subsystems for DTMCs and MDPs by means of a C++-prototype which uses `Gurobi` version 5.6 [Gur13] as MILP solver. We present results for MILP encodings for computing minimal critical subsystems for reachability properties of DTMCs and MDPs and, see Sections 6.1.1 and 6.2.1. We refrain from showing results for the approaches that use SMT solving, as the MILP approaches are clearly superior. For a detailed comparison as well as results for ω-regular properties of DTMCs, we refer to [4].

		without cuts				best cut combination					
Model	$\lvert S_{\min}\rvert$	|Vars|	|Constr|	Time	Mem.	F	B	S	R	Time	Mem.
CONT-5-2	6656	14030	13502	0.06	27.68	×	×	×	×	0.06	27.68
CONT-5-8	37375	77510	76982	0.30	114.18	×	×	×	×	0.30	114.18
CONT-7-2	133710	282134	273878	1.78	335.36	×	×	×	×	1.78	335.36
CONT-7-4	353931	744526	736270	4.47	833.15	×	×	×	×	4.47	833.15
CROWDS-5-4	732	2140	2119	2.86	19.54	√	√	inp	×	1.50	20.58
CROWDS-5-6	–	34794	55025	TO(2298)	–	√	×	inp	bw	TO(2292)	–
SLEADER-4-4	395	5200	8832	TO(395)	–	√	√	inp	both	0.13	16.54
SLEADER-4-6	–	7804	11705	TO(1953)	–	×	×	×	×	TO(1953)	–
SLEADER-8-4	229389	917694	917693	433.97	907.67	×	×	inp	×	271.84	907.13
NAND-5-2	102	2936	2935	0.08	13.16	×	×	×	×	0.08	13.16
NAND-5-3	165	4532	4531	0.40	17.15	×	×	×	×	0.40	17.15
NAND-5-4	217	6128	6127	0.80	21.52	×	×	×	×	0.80	21.52
NAND-25-2	–	1302116	1745352	TO(2187)	–	×	×	inp	×	TO(2187)	–

Table 8.9: Minimal critical subsystems for reachability properties of DTMCs

Table 8.9 shows the results for DTMCs and Table 8.10 the results for MDPs. For every benchmark instance, we list the number of states of a critical subsystem ($\lvert S_{\min}\rvert$), if it was possible to determine this value within the time limit of 3600 seconds. The block of columns "without cuts" shows the computation time and memory consumption that was needed without using any optimization in the form of redundant constrains, see Section 6.1.2. If the time limit was exceeded, we give in parentheses the computed lower bound on the size of the subsystem. Furthermore, we give the number of used variables (|Vars|) and the number of constraints (|Constr|).

For the last block of columns ("best cut combination") we ran our tool with all possible combinations of redundant constraints and report one which needed the smallest computation time or—in case none terminated within the time limit—the one with the best lower bound on the size of the MCS. For the reported combination of constraints, the first four columns show which optimizations were enabled: forward cuts ("F"), backward cuts ("B"), SCC cuts ("S"), and reachability constraints ("R"). Note that for MDPs we do not have SCC cuts. For the SCC cuts, we specify whether input cuts ("inp"), output cuts ("out") or both ("both") are used. For reachability constraints, either forward ("fw"), backward ("bw") constraints or both ("both") can be used. Additionally we report, as before, the computation time and memory consumption.

The results show that for many benchmarks the optimizations don't have much impact on the contract signing and on the nand benchmark. Contrarily, for crowds and for synchronous leader

				without cuts		best cut combination					
Model	$\|S_{\min}\|$	|Vars|	|Constr|	Time	Mem.	F	B	S	R	Time	Mem.
CONSENSUS-2-1	7	876	680	2.24	13.8203	√	×	×	×	0.22	11.5156
CONSENSUS-2-2	15	1692	1320	TO(15)	—	√	×	×	×	1683.17	341.37
CONSENSUS-2-4	—	3324	4528	TO(35)	—	√	√	×	×	TO(34)	—
CSMA-2-2	195	5444	4175	TO(195)	—	√	×	outp	×	338.51	169.23
CSMA-2-4	410	42442	61002	3135.78	1512.28	×	√	inp	bw	1205.23	239.42
CSMA-2-6	415	359960	200170	1629.26	1103.65	×	×	inp	×	1221.08	864.84
WLAN-0-1	7	8980	3160	0.03	15.8398	×	×	×	both	0.02	15.8594
WLAN-0-2	–	13769	9387	TO(121)	–	×	×	none	none	TO(121)	–
WLAN-0-5	–	28136	28287	TO(643)	–	√	√	none	none	TO(631)	–
WLAN-1-1	7	25837	8779	0.06	30.5898	×	×	×	bw	0.04	30.6289
WLAN-2-1	7	85402	28595	0.18	82.1133	√	×	input	fw	0.15	82.1289
WLAN-2-2	–	81374	31922	TO(121)	–	×	×	none	none	TO(121)	–

Table 8.10: Minimal critical subsystems for reachability properties of MDPs

election, the optimizations are the only way to compute solutions within the time limit. Note that it is not always possible to "predict" good cut combinations. However, in our benchmarking we achieved good results by testing all possible combinations for small instances of the same benchmark and using the best combinations for larger ones.

8.2.9 Comparison of the approaches

We now give a short summary of the experiments we conducted above. First, Figure 8.3 shows the sizes of critical subsystems for the CROWDS-5-4 benchmark for different probability thresholds. For all thresholds, we chose the optimal results for computing a minimal subsystem, DiPro, COMICS, and both the symbolic fragment and global search, respectively. Observe first, that the sizes for DiPro are in general larger as the other ones. This is due to the fact, that DiPro explores the state space in an on-the-fly manner while the other approaches have the whole state space at hand yielding a more informed search. The sizes COMICS computes are only slightly larger than the minimal subsystems while the symbolic approaches yield larger subsystems. Note that the adaptive symbolic fragment search yields a very large subsystem for thresholds that are near the actual model checking result. The reason lies in the adaptive nature of the method; in one step a large number of paths of the same probability is added to the subsystem while the probability does not increase in a way such that a backtracking is performed.

Figure 8.4 shows the running times for the aforementioned setting. The (explicit) counterexample generation of COMICS is clearly superior to the other ones. The computation of minimal critical subsystems is for some thresholds significantly slower than the other methods while one has to keep in mind that an *optimal* counterexample is computed.

In Figure 8.5, we compare the sizes of critical subsystems for different instances and thresholds as tested in the previous sections. In case a method was not able to compute the result due to running out of time or memory, this is indicated by the bar pointing downwards.

For all instances, each method except of DiPro yields a subsystem that is very near the actual

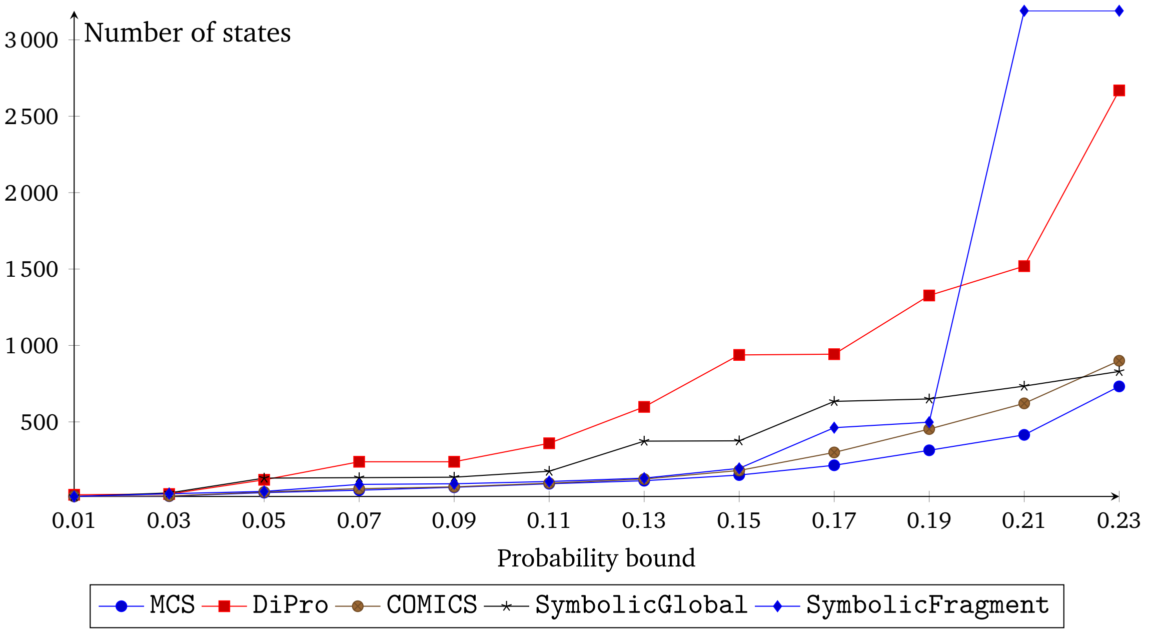


Figure 8.3: Size of the critical subsystem of CROWDS-5-4 for different probability bounds

minimal subsystem. However, this will not hold for larger benchmarks, where many unnecessary parts of a system are explored by path searching algorithms. Note furthermore, that we depict the best results for the heuristic methods here while other configurations often yield significantly larger systems.

Finally, we present the running times for all methods and different benchmarks in Figure 8.6. If the running time exceeds 100 seconds, we cut the bar there indicating that the computation could be finished within the time limit of 3600 seconds.

For contract, only DiPro was not able to compute results within the time limit for all instances, while the symbolic implementation performs worse than COMICS and the implementation for MCS. For larger crowds instances, our implementation is not able to compute a minimal critical subsystem. For synchronous leader election, the results vary in a way that for one instance DiPro performs best while for the next one DiPro and COMICS ran into a timeout while the symbolic implementation and the implementation for MCS are able to compute a result.

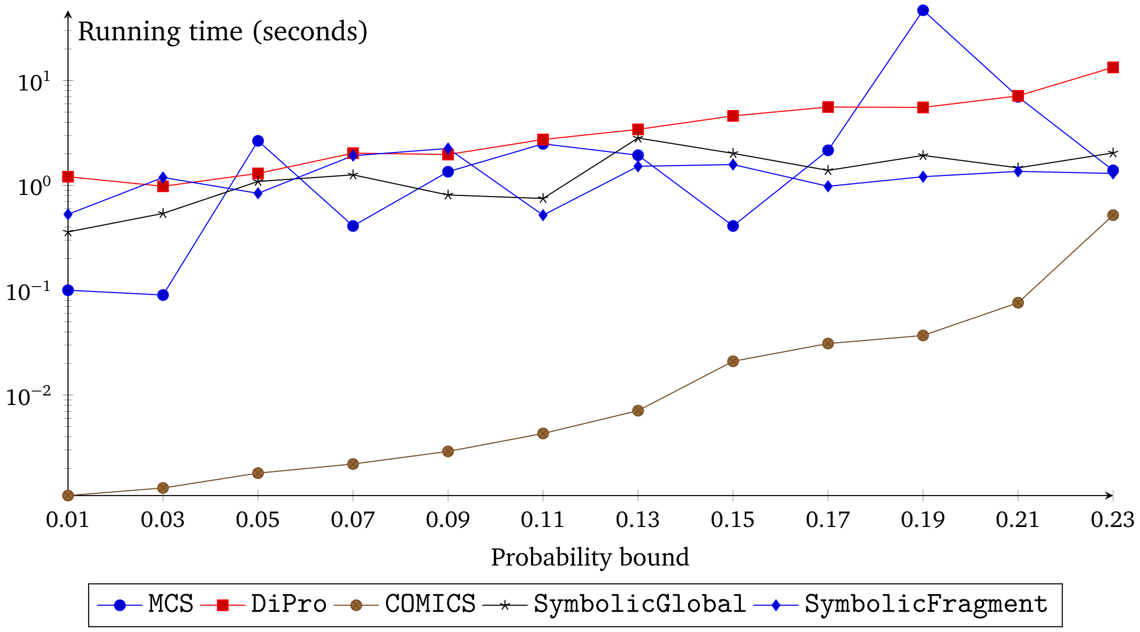


Figure 8.4: Computation time for different probability bounds

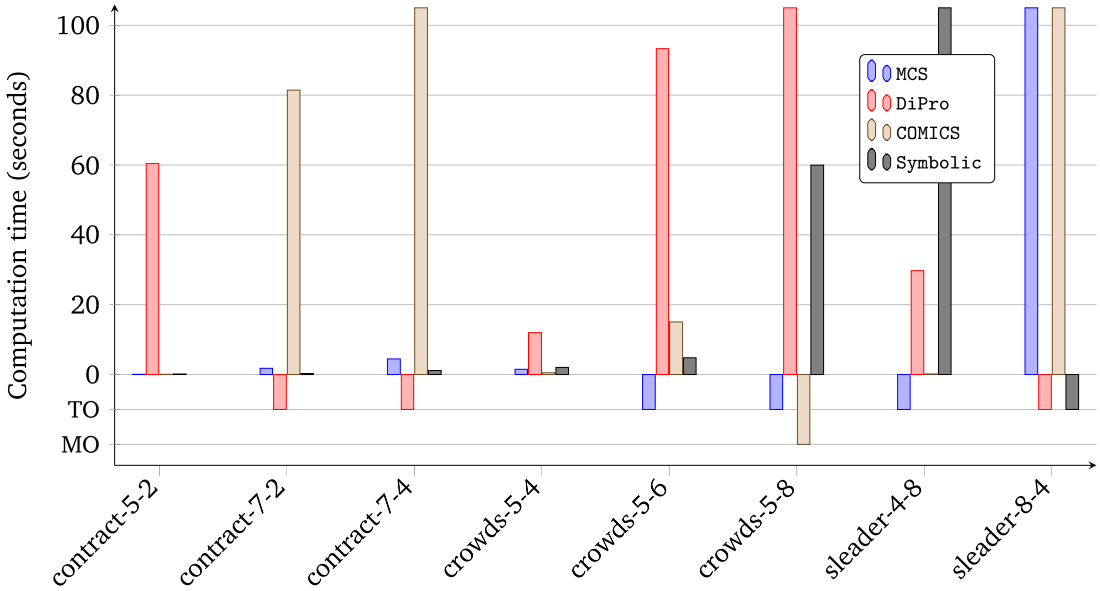


Figure 8.6: Computation time for different benchmarks

8.2.10 High-level counterexamples

Apart from the methods generating explicit or symbolic representations of the *state space* of a counterexample, we now present results for the *high-level* counterexamples introduced in Chapter 7. For this problem, we developed a prototype in C++ using again the MILP solver Gurobi [Gur13]. The experiments were performed on an Intel® Xeon® CPU with 3.3 GHz clock

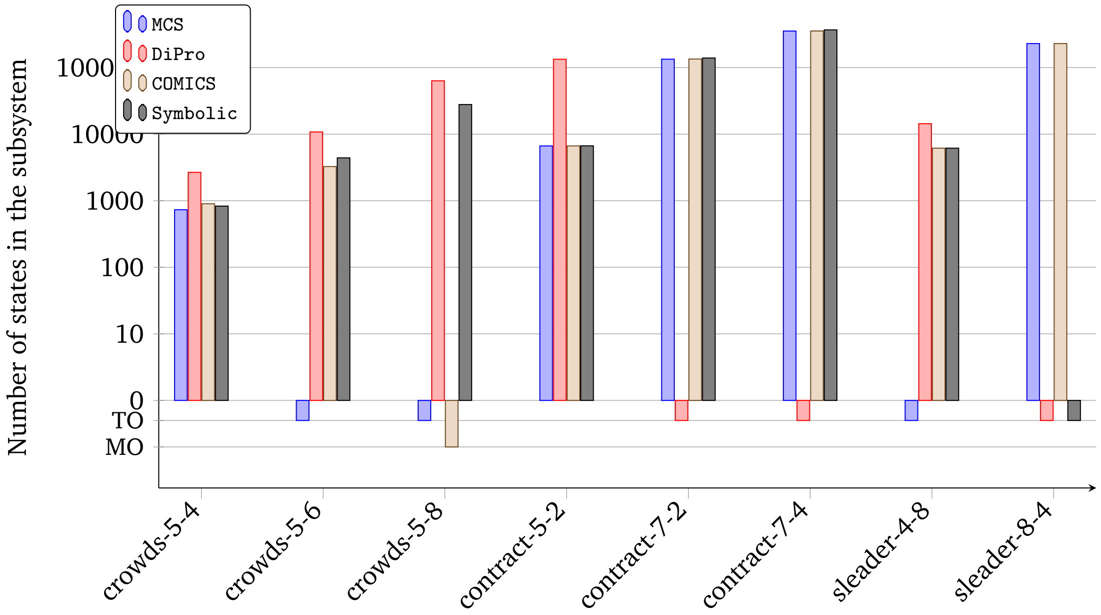


Figure 8.5: Size of the subsystem for different benchmarks

frequency and 32 GB of main memory, running Ubuntu 12.04. For this setting we use a timeout of 600 seconds, again denoted by TO.

The results of our experiments computing a smallest critical command set are displayed in Table 8.11. As in the previous section, we show the results without any optimizations and the results for the best combination of redundant constraints.

For both variants, we give the computation time in seconds, the memory consumption in MB, the number of commands in the critical command set (“n”), and—in case the time limit was exceeded—a lower bound on the size of the smallest critical command set (“lb”).

	no cuts						best cut combination							
Model	Var.	Constr.	Time	Mem.	n	lb	Time	Mem.	n	lb	σ_f	σ_b	ℓ	‖
CONSENSUS-2-1	277	492	TO	855	9	8	69.05	42	9	opt	√	√	√	×
CONSENSUS-2-2	533	1004	TO	1140	9	6	TO	1111	9	8	×	×	√	√
CONSENSUS-2-4	1045	2028	TO	553	9	6	TO	974	9	7	×	√	√	√
CSMA-2-2	2123	5990	7.30	26	32	opt	4.19	28	32	opt	×	×	×	√
CSMA-2-4	15977	46882	196.11	215	36	opt	77.43	261	36	opt	×	√	√	√
WLAN-0-2	7072	6602	TO	474	33	15	TO	373	33	31	√	√	√	√
WLAN-0-5	19012	25808	TO	1083	??	??	TO	1083	??	??	×	×	×	×
WLAN-2-1	28538	192	1.12	45	8	opt	0.82	45	8	opt	√	√	×	×
WLAN-2-2	29607	6602	TO	517	33	15	TO	416	33	31	√	√	×	√

Table 8.11: Command minimization

For the experiments without optimizations, we give the number of variables (“Var.”) and constraints (“Constr.”) of the MILP, and, in—case the time limit was exceeded—a lower bound on the size of the smallest critical command set (“lb”). An entry “??” for the number of commands means that the solver was not able to find a non-trivial critical command set within the time limit. For the best cut combination, the last four columns specify the combination of cuts leading to the best running time. Here the column "σ_f" corresponds to scheduler forward cuts, "σ_b" to scheduler backward cuts, "ℓ" to label cuts, and "$\|$" to synchronization cuts. An entry "$\surd$" indicates that the corresponding constraints have been added to the MILP, "×" that they were not used. For details about these optimizations we refer to Section 7.2.4.

Although we ran into many timeouts, in particular without any cuts, in almost all cases a solution could be found within the time limit. We suppose that also the solutions of the aborted instances are optimal or close to optimal. It seems that the MILP solver can quickly find good (or even optimal) solutions due to sophisticated heuristics, but proving their optimality is hard. Further experiments have shown that the scheduler forward cuts have the strongest effect on the lower bound. Choosing good cuts consequently helps the solver to obtain optimal solutions.

CHAPTER 9

Conclusion and future work

In this thesis we presented results obtained over the last years concerning the efficient generation of counterexamples for discrete-time Markov models. Our approaches range over

- *heuristic methods* based on graph search algorithms and the utilization of SAT solvers,
- the representation and computation using *symbolic data structures* such as BDDs,
- and exact methods computing *optimal counterexamples* in terms of their size.

Moreover, we presented *model checking algorithms* both for DTMCs and parametric DTMCs. As a first approach on counterexamples that are readable for humans, we showed how to compute *high-level counterexamples* at the design level of stochastic systems.

Our future work will concern the *application* of counterexamples, for instance further approaches on probabilistic CEGAR [HWZ08] or assume-guarantee reasoning [KPC12a]. We will explore possibilities and identify needs, for instance in the field of *robotics* where environment and behavior of robots can be modeled probabilistically [PPMT13]. A natural question is whether the behavior can be improved by means of counterexamples showing undesired behavior.

Another still open question is how a counterexample can actually be used for debugging a system. Further research will concern the application of counterexamples in *model repair* [BGK^{+}11, 19], where a faulty system is automatically repaired such that a certain property holds.

Our model checking algorithm for *parametric* DTMCs yields improved running times for determining functions in the systems parameters that describe reachability probabilities. In the future we will explore techniques that enable parameter synthesis for larger systems with more parameters, where as first step recently the new tool `PROHPhESY` has been released [17]. A user needs to be provided with easily accessible information on how parameters need to be instantiated in order to satisfy certain requirements. Moreover, it would be interesting to investigate the extension to richer models such as continuous-time Markov chains (CTMCs).

High-level counterexamples indicate how a system designer can access debug information at the design level. Out of the scope of this thesis we presented another approach based on utilizing a MAX-SAT solver which yielded significant improvements in terms of running times and system sizes [22]. Current work targets high-level counterexamples for CTMCs.

[Ach09] Tobias Achterberg. SCIP: Solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009.

[ADvR08] Miguel E. Andrés, Pedro D'Argenio, and Peter van Rossum. Significant diagnostic counterexamples in probabilistic model checking. In *Proc. of HVC*, volume 5394 of *LNCS*, pages 129–148. Springer, 2008.

[AH90] James Aspnes and Maurice Herlihy. Fast randomized consensus using shared memory. *Journal of Algorithms*, 15(1):441–460, 1990.

[AH99] Rajeev Alur and Thomas A. Henzinger. Reactive modules. *Formal Methods in System Design*, 15(1):7–48, 1999.

[AHL05] Husain Aljazzar, Holger Hermanns, and Stefan Leue. Counterexamples for timed probabilistic reachability. In *Proc. of FORMATS*, volume 3829 of *LNCS*, pages 177–195. Springer, 2005.

[AL06] Husain Aljazzar and Stefan Leue. Extended directed search for probabilistic timed reachability. In *Proc. of FORMATS*, volume 4202 of *LNCS*, pages 33–51. Springer, 2006.

[AL09] Husain Aljazzar and Stefan Leue. Generation of counterexamples for model checking of Markov decision processes. In *Proc. of QEST*, pages 197–206. IEEE Computer Society, 2009.

[AL10] Husain Aljazzar and Stefan Leue. Directed explicit state-space search in the generation of counterexamples for stochastic model checking. *IEEE Transactions on Software Engineering*, 36(1):37–60, 2010.

[AL11] Husain Aljazzar and Stefan Leue. K^*: A heuristic search algorithm for finding the k shortest paths. *Artificial Intelligence*, 175(18):2129–2154, 2011.

[ALFLS11] Husain Aljazzar, Florian Leitner-Fischer, Stefan Leue, and Dimitar Simeonov. DiPro – A tool for probabilistic counterexample generation. In *Proc. of SPIN*, volume 6823 of *LNCS*, pages 183–187. Springer, 2011.

[AS09] Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern SAT solvers. In *Proc. of IJCAI*, pages 399–404, 2009.

[BCC$^+$03] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, Ofer Strichman, and Yunshan Zhu. Bounded model checking. *Advances in Computers*, 58:118–149, 2003.

[BCF$^+$08] Roberto Bruttomesso, Alessandro Cimatti, Anders Franzén, Alberto Griggio, and Roberto Sebastiani. The MathSAT 4 SMT solver. In *Proc. of CAV*, volume 5123 of *LNCS*, pages 299–303. Springer, 2008.

[BCH$^+$97] Christel Baier, Edmund M. Clarke, Vasiliki Hartonas-Garmhausen, Marta Z. Kwiatkowska, and Mark Ryan. Symbolic model checking for probabilistic processes. In *Proc. of ICALP*, volume 1256 of *LNCS*, pages 430–440. Springer, 1997.

[BCM$^+$92] Jerry R. Burch, Edmund M. Clarke, Kenneth L. McMillan, David L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, 1992.

[BFK02] Christian Bauer, Alexander Frink, and Richard Kreckel. Introduction to the GiNaC framework for symbolic computation within the C++ programming language. *Journal of Symbolic Computing*, 33(1):1–12, 2002.

[BGK$^+$11] Ezio Bartocci, Radu Grosu, Panagiotis Katsaros, C.R. Ramakrishnan, and Scott A Smolka. Model repair for probabilistic systems. In *Proc. of TACAS*, volume 6605 of *LNCS*, pages 326–340. Springer, 2011.

[BHvMW09] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. The MIT Press, 2008.

[BP12] Simon Busard and Charles Pecheur. Rich counter-examples for temporal-epistemic logic model checking. In *Proc. of IWIGP*, volume 78 of *EPTCS*, pages 39–53, 2012.

[Bry86] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.

[Bry91] Randal E. Bryant. On the complexity of VLSI implementations and graph representations of boolean functions with application to integer multiplication. *IEEE Transactions on Computers*, 40(2):205–213, 1991.

[BT02] Dimitri P. Bertsekas and John N. Tsitsiklis. *Introduction to Probability*. Athena Scientific books. Athena Scientific, 2002.

[BW96] Beate Bollig and Ingo Wegener. Improving the variable ordering of OBDDs is NP-complete. *IEEE Transactions on Computers*, 45(9):993–1002, 1996.

[BW07] Rabi Bhattacharya and Edward C. Waymire. *A Basic Course in Probability Theory*. Universitext. Springer, 2007.

[CBRZ01] Edmund M. Clarke, Armin Biere, Richard Raimi, and Yunshan Zhu. Bounded model checking using satisfiability solving. *Formal Methods in System Design*, 19(1):7–34, 2001.

[CGJ$^+$00] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In *Proc. of CAV*, volume 1855 of *LNCS*, pages 154–169. Springer, 2000.

[CGMZ95] Edmund M. Clarke, Orna Grumberg, Kenneth L. McMillan, and Xudong Zhao. Efficient generation of counterexamples and witnesses in symbolic model checking. In *Proc. of DAC*, pages 427–432. IEEE Computer Society, 1995.

[CH11] Krishnendu Chatterjee and Monika Henzinger. Faster and dynamic algorithms for maximal end-component decomposition and related graph problems in probabilistic verification. In *Proc. of SODA*, pages 1318–1336. SIAM, 2011.

[CJLV02] Edmund M. Clarke, Somesh Jha, Yuan Lu, and Helmut Veith. Tree-like counterexamples in model checking. In *Proc. of LICS*, pages 19–29. IEEE Computer Society, 2002.

[Cla08] Edmund M. Clarke. The birth of model checking. In *25 Years of Model Checking – History, Achievements, Perspectives*, volume 5000 of *LNCS*, pages 1–26. Springer, 2008.

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proc. of STOC*, pages 151–158. ACM Press, 1971.

[cpl12] IBM CPLEX optimization studio, version 12.4. `http://www-01.ibm.com/software/integration/optimization/cplex-optimization-studio/`, 2012.

[CSS03] Jean-Michel Couvreur, Nasser Saheb, and Grégoire Sutre. An optimal automata approach to LTL model checking of probabilistic systems. In *Proc. of LPAR*, volume 2850 of *LNCS*, pages 361–375. Springer, 2003.

[CV03] Edmund M. Clarke and Helmut Veith. Counterexamples revisited: Principles, algorithms, applications. In *Verification: Theory and Practice*, volume 2772 of *LNCS*, pages 208–224. Springer, 2003.

[CV10] Rohit Chadha and Mahesh Viswanathan. A counterexample-guided abstraction-refinement framework for Markov decision processes. *ACM Transactions on Computational Logic*, 12(1):1–45, 2010.

[dA97] Luca de Alfaro. *Formal Verification of Probabilistic Systems*. PhD thesis, Stanford University, 1997.

[Daw04] Conrado Daws. Symbolic and parametric model checking of discrete-time Markov chains. In *Proc. of ICTAC*, volume 3407 of *LNCS*, pages 280–294. Springer, 2004.

[DdM06] Bruno Dutertre and Leonardo Mendonça de Moura. A fast linear-arithmetic solver for DPLL(T). In *Proc. of CAV*, volume 4144 of *LNCS*, pages 81–94. Springer, 2006.

[DHK08] Berteun Damman, Tingting Han, and Joost-Pieter Katoen. Regular expressions for PCTL counterexamples. In *Proc. of QEST*, pages 179–188. IEEE Computer Society, 2008.

[Dij59] Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[Dij75] Edsger W. Dijkstra. Guarded commands, non-determinacy and formal derivation of programs. *Communications of the ACM*, 18(8):453–457, 1975.

[DJJL01] Pedro R. D'Argenio, Bertrand Jeannet, Henrik Ejersbo Jensen, and Kim Guldstrand Larsen. Reachability analysis of probabilistic systems by successive refinements. In *Proc. of PAPM/PROBMIV*, volume 2165 of *LNCS*, pages 39–56. Springer, 2001.

[DKRT97] Pedro R. D'Argenio, Joost-Pieter Katoen, Theo C. Ruys, and Jan Tretmans. The bounded retransmission protocol must be on time! In *Proc. of TACAS*, volume 1217 of *LNCS*, pages 416–431. Springer, 1997.

[DLL62] Martin Davis, George Logemann, and Donald W. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.

[dMB08] Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *Proc. of TACAS*, volume 4963 of *LNCS*, pages 337–340. Springer, 2008.

[dMB11] Leonardo Mendonça de Moura and Nikolaj Bjørner. Satisfiability modulo theories: Introduction and applications. *Communications of the ACM*, 54(9):69–77, 2011.

[DP60] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960.

[Epp98] David Eppstein. Finding the k shortest paths. *SIAM Journal on Computing*, 28(2):652–673, 1998.

[ES03] Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In *Proc. of SAT*, volume 2919 of *LNCS*, pages 502–518. Springer, 2003.

[Fel68] William Feller. *An Introduction to Probability Theory and Its Applications*, volume 1. Wiley, 1968.

[FMY97] Masahiro Fujita, Patrick C. McGeer, and Jerry Chih-Yuan Yang. Multi-terminal binary decision diagrams: An efficient data structure for matrix representation. *Formal Methods in System Design*, 10(2/3):149–169, 1997.

[FWA09] Gordon Fraser, Franz Wotawa, and Paul Ammann. Issues in using model checkers for test case generation. *Journal of Systems and Software*, 82(9):1403–1418, 2009.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co Ltd, 1979.

[GMZ04] Paul Gastin, Pierre Moro, and Marc Zeitoun. Minimization of counterexamples in SPIN. In *Proc. of SPIN*, volume 2989 of *LNCS*, pages 92–108. Springer, 2004.

[GSS10] Michael Günther, Johann Schuster, and Markus Siegle. Symbolic calculation of k-shortest paths and related measures with the stochastic process algebra tool CASPA. In *Proc. of DYADEM-FTS*, pages 13–18. ACM Press, 2010.

[Gur13] Gurobi Optimization, Inc. Gurobi optimizer reference manual. `http://www.gurobi.com`, 2013.

[Han09] Tingting Han. *Diagnosis, Synthesis and Analysis of Probabilistic Models*. PhD thesis, RWTH Aachen University, 2009.

[HHWZ10] Ernst Moritz Hahn, Holger Hermanns, Björn Wachter, and Lijun Zhang. Param: A model checker for parametric Markov models. In *Proc. of CAV*, pages 660–664. Springer, 2010.

[HHZ09] Ernst Moritz Hahn, Holger Hermanns, and Lijun Zhang. Probabilistic reachability for parametric markov models. In *Proc. of SPIN*, volume 5578 of *LNCS*. Springer, 2009.

[HHZ10] Ernst Moritz Hahn, Holger Hermanns, and Lijun Zhang. Probabilistic reachability for parametric Markov models. *Software Tools for Technology Transfer*, 13(1):3–19, 2010.

[HJ79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, Cambridge, 1979.

[HJ94] Hans Hansson and Bengt Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994.

[HK07a] Tingting Han and Joost-Pieter Katoen. Counterexamples in probabilistic model checking. In *Proc. of TACAS*, volume 4424 of *LNCS*, pages 72–86. Springer, 2007.

[HKD09] Tingting Han, Joost-Pieter Katoen, and Berteun Damman. Counterexample generation in probabilistic model checking. *IEEE Transactions on Software Engineering*, 35(2):241–257, 2009.

[HKN$^+$03] Holger Hermanns, Marta Z. Kwiatkowska, Gethin Norman, David Parker, and Markus Siegle. On the use of MTBDDs for performability analysis and verification of stochastic systems. *Journal of Logic and Algebraic Programming*, 56(1–2):23–67, 2003.

[How79] Ronald A. Howard. *Dynamic Probabilistic Systems*, volume 1: Markov Chains. John Wiley and Sons, 1979.

[HSM97] Jifeng He, Karen Seidel, and Annabelle McIver. Probabilistic models for the guarded command language. *Science of Computer Programming*, 28(2–3):171–192, 1997.

[HWZ08] Holger Hermanns, Björn Wachter, and Lijun Zhang. Probabilistic CEGAR. In *Proc. of CAV*, volume 5123 of *LNCS*, pages 162–175. Springer, 2008.

[IR90] Alon Itai and Michael Rodeh. Symmetry breaking in distributed networks. *Information and Computation*, 88(1):60–87, 1990.

[JdM12] Dejan Jovanovic and Leonardo Mendonça de Moura. Solving non-linear arithmetic. In *Proc. of ijcar*, volume 7364 of *LNCS*, pages 339–354. Springer, 2012.

[JGP99] Edmund M. Clarke Jr., Orna Grumberg, and Doron A. Peled. *Model Checking*. The MIT Press, 1999.

[JKO$^+$07] David N. Jansen, Joost-Pieter Katoen, Marcel Oldenkamp, Mariëlle Stoelinga, and Ivan S. Zapreev. How fast and fat is your probabilistic model checker? An experimental performance comparison. In *Proc. of HVC*, volume 4899 of *LNCS*, pages 69–85. Springer, 2007.

[JM99] Víctor M. Jiménez and Andrés Marzal. Computing the k shortest paths: A new algorithm and an experimental comparison. In *Proc. of WAE)*, volume 1668 of *LNCS*, pages 15–29. Springer, 1999.

[Kat13] Joost-Pieter Katoen. Model checking meets probability: A gentle introduction. In *Engineering dependable software systems*, volume 34 of *NATO Science for Peace and Security Series - D: Information and Communication Security*, pages 177–205. IOS Press, Amsterdam, 2013.

[KKNP10] Mark Kattenbelt, Marta Z. Kwiatkowska, Gethin Norman, and David Parker. A game-based abstraction-refinement framework for Markov decision processes. *Formal Methods in System Design*, 36(3):246–280, 2010.

[KNP02] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. Probabilistic symbolic model checking with PRISM: A hybrid approach. In *Proc. of TACAS*, volume 2280 of *LNCS*, pages 52–66. Springer, 2002.

[KNP07] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. Stochastic model checking. In *Proc. of SFM*, volume 4486 of *LNCS*, pages 220–270. Springer, 2007.

[KNP11] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *Proc. of CAV*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

[KNP12] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. The PRISM benchmark suite. In *Proc. of QEST*, pages 203–204. IEEE Computer Society, 2012.

[KNS02] Marta Z. Kwiatkowska, Gethin Norman, and Jeremy Sproston. Probabilistic model checking of the IEEE 802.11 wireless local area network protocol. In *Proc. of PAPM/PROBMIV*, volume 2399 of *LNCS*, pages 169–187. Springer, 2002.

[KNS03] Marta Z. Kwiatkowska, Gethin Norman, and Jeremy Sproston. Probabilistic model checking of deadline properties in the IEEE 1394 FireWire root contention protocol. *Formal Aspects of Computing*, 14(3):295–318, 2003.

[KPC12a] Anvesh Komuravelli, Corina S. Pasareanu, and Edmund M. Clarke. Assume-guarantee abstraction refinement for probabilistic systems. In P. Madhusudan and Sanjit A. Seshia, editors, *Proc. of CAV*, volume 7358 of *LNCS*, pages 310–326. Springer, 2012.

[Kri63] Saul A. Kripke. Semantical considerations on modal logic. *Acta Philosophica Fennica*, 16(1963):83–94, 1963.

[KS69] John George Kemeny and James Laurie Snell. *Finite Markov chains*. University series in undergraduate mathematics. VanNostrand, repr edition, 1969.

[Kul95] Vidyadhar G. Kulkarni. *Modeling and Analysis of Stochastic Systems*. Chapman & Hall, Ltd., 1995.

[KZH$^+$11] Joost-Pieter Katoen, Ivan S. Zapreev, Ernst Moritz Hahn, Holger Hermanns, and David N. Jansen. The ins and outs of the probabilistic model checker MRMC. *Performance Evaluation*, 68(2):90–104, 2011.

[Nor97] James R. Norris. *Markov Chains*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 1997.

[NPKS05] Gethin Norman, David Parker, Marta Z. Kwiatkowska, and Sandeep Shukla. Evaluating the reliability of NAND multiplexing with PRISM. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(10):1629–1637, 2005.

[NSY92] Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Compiling real-time specifications into extended automata. *IEEE Transactions on Software Engineering*, 18(9):794–804, 1992.

[Par02] David Parker. *Implementation of Symbolic Model Checking for Probabilistic Systems*. PhD thesis, University of Birmingham, 2002.

[PPMT13] Shashank Pathak, Luca Pulina, Giorgio Metta, and Armando Tacchella. Ensuring safety of policies learned by reinforcement: Reaching objects in the presence of obstacles with the icub. In *Proc. of IEEE/RSJ*, pages 170–175. IEEE, 2013.

[PRI10] PRISM Website, 2010. `http://prismmodelchecker.org`.

[QSS00] Alfio Quarteroni, Riccardo Sacco, and Fausto Saleri. *Numerical Mathematics*. Springer, 2000.

[RR98] Michael K. Reiter and Aviel D. Rubin. Crowds: Anonymity for web transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, 1998.

[Rud93] Richard Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Proc. of ICCAD*, pages 42–47. IEEE Computer Society, 1993.

[Saf89] Shmuel Safra. *Complexity of Automata on Infinite Objects*. PhD thesis, The Weizmann Institue of Science, Rehovot, Israel, 1989.

[SB05] Viktor Schuppan and Armin Biere. Shortest counterexamples for symbolic model checking of LTL with past. In *Proc. of TACAS*, volume 3440 of *LNCS*, pages 493–509. Springer, 2005.

[Sch86] Alexander Schrijver. *Theory of Linear and Integer Programming*. Wiley, 1986.

[Sch12] Johann Schuster. *Towards faster numerical solution of continuous time Markov chains stored by symbolic data structures*. PhD thesis, Universität der Bundeswehr München, 2012. `http://d-nb.info/102057920X/34`.

[SR13] Guoxin Su and David S. Rosenblum. Asymptotic bounds for quantitative verification of perturbed probabilistic systems. In *Proc. of ICFEM*, volume 8144 of *LNCS*, pages 297–312. Springer, 2013.

[Ste94] William J. Stewart. *Introduction to the numerical solution of Markov chains*. Princeton Univ. Press, Princeton, NJ, 1994.

[Sto03] Mariëlle Stoelinga. Fun with FireWire: A comparative study of formal verification methods applied to the IEEE 1394 Root Contention Protocol. *Formal Aspects of Computing*, 14(3):328–337, 2003.

[Tar72] Robert Endre Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

[TBT06] Axel Thümmler, Peter Buchholz, and Miklós Telek. A novel approach for phase-type fitting with the em algorithm. *IEEE Transactions on Dependable and Secure Computing*, 3(3):245–258, 2006.

[Tse83] Grigori S Tseitin. On the complexity of derivation in propositional calculus. In *Automation of Reasoning*, pages 466–483. Springer, 1983.

[Var85] Moshe Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *Proc. of FOCS*, pages 327–338. IEEE Computer Society, 1985.

[Var99] Moshe Y. Vardi. Probabilistic linear-time model checking: An overview of the automata-theoretic approach. In *Proc. of ARTS*, volume 1601 of *LNCS*, pages 265–276. Springer, 1999.

[vN56] John von Neumann. Probabilistic logics and synthesis of reliable organisms from unreliable components. In *Automata Studies*, pages 43–98. Princeton University Press, 1956.

[Wac10] Bjorn Wachter. *Refined Probabilistic Abstraction*. PhD thesis, 2010.

[WBB09] Ralf Wimmer, Bettina Braitling, and Bernd Becker. Counterexample generation for discrete-time Markov chains using bounded model checking. In *Proc. of VMCAI*, volume 5403 of *LNCS*, pages 366–380. Springer, 2009.

Index