BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INTELLIGENT SYSTEMS
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

COMPUTER-AIDED SYNTHESIS OF PROBABILISTIC MODELS

POČÍTAČEM PODPOROVANÁ SYNTÉZA PRAVDĚPODOBNOSTNÍCH MODELŮ

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR **Bc. ROMAN ANDRIUSHCHENKO**
AUTOR PRÁCE

SUPERVISOR **RNDr. MILAN ČEŠKA, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2020

Brno University of Technology
Faculty of Information Technology

Department of Intelligent Systems (DITS) Academic year 2019/2020

Master's Thesis Specification

Student: **Andriushchenko Roman, Bc.**
Programme: Information Technology Field of study: Mathematical Methods in Information Technology
Title: **Computer-Aided Synthesis of Probabilistic Models**
Category: Formal Verification
Assignment:

1. Study the current methods for automated design and synthesis of probabilistic models including methods based on MDP abstraction and counter-example guided inductive synthesis.
2. Evaluate the methods on practically relevant case-studies and identify their limitations.
3. Design possible improvements and extensions of the methods including a fusion of the methods from the item 1.
4. Implement the improvements and extension within an existing probabilistic model-checker (e.g. STORM or PRISM)
5. Carry out a detailed evaluation of the implemented methods and assess improvements over the existing methods.

Recommended literature:

1. Milan Češka, Nils Jansen, Sebastian Junges, and Joost-Pieter Katoen. Shepherding hordes of Markov chains. In Proc. of TACAS'19. Springer, 2019.
2. Milan Češka, Christian Hensel, Sebastian Junges, and Joost-Pieter Katoen. Counterexample-Driven Synthesis for Probabilistic Program Sketches. In Proc. of FM'19. Springer, 2019.

Requirements for the semestral defence:

- Items 1, 2 and partially item 3.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor: **Češka Milan, RNDr., Ph.D.**
Head of Department: Hanáček Petr, doc. Dr. Ing.
Beginning of work: November 1, 2019
Submission deadline: June 3, 2020
Approval date: October 31, 2019

Abstract

This thesis considers the problem of automated synthesis of probabilistic systems: having a family of Markov chains, how can one efficiently identify a chain satisfying a given specification? Such families often arise in various domains of engineering when modeling systems under uncertainty, and deciding even the simplest problems shows to be $\mathcal{NP}$-hard. To tackle this problem, we adopt the principles of counterexample-guided inductive synthesis (CEGIS) and abstraction refinement (CEGAR) and develop a novel integrated technique for probabilistic synthesis. Experiments on practically relevant case studies demonstrate that the designed technique is not only comparable to state-of-the-art synthesis approaches, in most cases it manages to significantly outperform existing methods, sometimes by a margin of orders of magnitude.

Abstrakt

Předkládaná práce se zabývá problémem automatizované syntézy pravděpodobnostních systémů: máme-li rodinu Markovských řetězců, jak lze efektivně identifikovat ten který odpovídá zadané specifikaci? Takové rodiny často vznikají v nejrůznějších oblastech inženýrství při modelování systémů s neurčitostí a rozhodování i těch nejjednodušších syntézních otázek představuje $\mathcal{NP}$-těžký problém. V dané práci my zkoumáme existující techniky založené na protipříklady řízené induktivní syntéze (counterexample-guided inductive synthesis, CEGIS) a na zjemňování abstrakce (counterexample-guided abstraction refinement, CEGAR) a navrhujeme novou integrovanou metodu pro pravděpodobnostní syntézu. Experimenty nad relevantními modely demonstrují, že navržená technika je nejenom srovnatelná s moderními metodami, ale ve většině případů dokáže výrazně překonat, někdy i o několik řádů, existující přístupy.

Keywords

Markov models, probabilistic model checking, synthesis of probabilistic models

Klíčová slova

Markovovy modely, probabilistický model checking, syntéza pravděpodobnostních modelů

Reference

ANDRIUSHCHENKO, Roman. *Computer-Aided Synthesis of Probabilistic Models.* Brno, 2020. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor RNDr. Milan Češka, Ph.D.

Rozšířený abstrakt

Modelování systémů vykazující žádoucí chování – např. návrh síťového protokolu umožňujícího zvýšit paketovou propustnost, nebo výběr optimální strategie pro řízení spotřeby – je netriviální úkol vyžadující uvažování nad mnoha alternativami. Lze použít tzv. *program sketch* [41, 5] – popis systému obsahující volby – a pak uvažovat jak dané volby instancovat tak, aby výsledný program vyhovoval zadané specifikaci. Tyto volby mohou pokrývat počáteční hodnoty proměnných nebo dokonce podmínky pro větvení a tímpádem ovlivňovat topologii stavového prostoru programu. Navíc modelování systému s nepředvídatelným nebo nespolehlivým chováním vyžaduje použití matematických aparátů založených na teorii pravděpodobnosti, což vede k tzv. *pravděpodobnostním programům*. Aplikace pravděpodobnostních programů pokrývá širokou škálu výzkumných oblastí, včetně např. analýzy softwarových produkčních linek [40, 22, 16], syntézy ovladaců pro částečně pozorovatelné modely [30, 36], adaptivního managementu [7, 20] nebo návrhu komunikačních protokolů [28, 34].

Použijeme-li Markovův řetězec jako operační model pro pravděpodobnostní program, automatizovaný syntezátor zkoumá rodinu těchto řetězců a vybíra ten který odpovídá zadané specifikaci. Automatizovaná syntéza sama o sobě představuje obrovskou výzvu, zejména kvůli problému stavové exploze, který ovlivňuje syntézu dvojím způsobem: nejen počet kandidátních řešení je exponenciální vůči počtu uvažovaných voleb, ale stavový prostor každého jednotlivého řetězce obvykle také roste exponenciálně vůči délce popisu programu. Problém syntézy může být naivně vyřešen analýzou všech jednotlivých členů rodiny – tzv. *one-by-one* přístup [16, 17] – nebo modelováním rodiny kandidátních řešení jako jeden *all-in-one* Markovův rozhodovací proces [16, 18], jehož velikost je úměrná velikosti rodiny. Bohužel problém dvojí stavové exploze prostoru způsobuje, že oba tyto přístupy nejsou použitelné pro velké rodiny. K potlačení problému stavové exploze autoři [22] úspěšně používají evoluční vyhledávací algoritmy pro syntézu softwarových systémů. Metoda však zůstává neúplná a není schopna účinně řešit náročnější problémy syntézy, například který člen rodiny vyhovuje specifikaci *optimálně*. K zajištění úplnosti byly navrženy sofistikovanější metody, které se obvykle pokoušejí poskytnout analýzu celých podrodin řetězců najednou. V této práci se zaměříme na dva nejmodernější úplné přístupy k pravděpodobnostní syntéze. První metodou je *protipříklady řízená induktivní syntéza* (counterexample-guided inductive synthesis, CEGIS) [13], která analyzuje každý řetězec jeden po druhém a využívá kritické podsystémy protipříkladů pro zamítnutí celých podrodin řetězců, které neodpovídají specifikaci. Druhým přístupem je *protipříklady řízené zjemňování abstrakce* (counterexample-guided abstraction refinement, CEGAR) [14], který analyzuje celou rodinu řetězců najednou a pokud analýza dává neprůkazné výsledky, dělí rodinu kandidátů na jemnější podrodiny, které jsou zpracovány analogicky. I když obě tyto techniky prokázaly přesvědčivé výsledky, nejsou zbaveny omezení a obadva přístupy jsou nesrovnatelné v tom smyslu, že jedna technika může být vhodnější pro specifické třídy pravděpodobnostních programů a naopak.

V této práci se zabýváme fúzí induktivních a abstrakčních přístupů a navrhujeme novou integrovanou metodu, která kombinuje silné stránky obou technik. Konkrétněji, navržená metoda nejprve analyzuje rodinu najednou, podobně jako CEGAR, a sbírá některé obecné informace o nejhorším a nejlepším chování (vůči zadané specifikaci) každého Markovského řetězce v rodině. Poté metoda analyzuje rodinu analogicky jako CEGIS, ale používá shromážděná data ke zlepšení kvality kritických podsystémů, což vede na zrychlení prohledávání stavovým prostorem kandidátních řešení. Všechny tři metody – CEGIS, CEGAR a novou techniku – vyhodnocujeme na rozsáhlé sadě případových studií a ukazujeme, že nově navržený přístup je korektní, úplný a navíc zmírňuje rozdíly mezi induktivními a

abstraktními přístupy, takže umožňuje efektivní syntézu jakéhokoliv konkrétního modelu. Z hlediska výkonu navržená technika je nejenom srovnatelná s moderními metodami, ale ve většině případů dokáže výrazně překonat, někdy i o několik řádů, existující přístupy.

Analýza jednotlivých Markovových řetězců je kritickou části procesu syntézy a představuje výzvu ortogonální k syntéze pravděpodobnostních systémů. Zatímco syntéza bojuje s exponenciálním růstem velikosti rodiny, analýza jednotlivých řetězců čelí explozi základního stavového prostoru. Součástí práce je i krátká prezentace vybraných výsledků kterých jsme dosáhli v oblasti aproximačních technik pro Markovovy modely. Výsledky byly původně prezentovány jako bakalářská práce [6], ale od té doby byly významně rozšířeny a vylepšeny během práce na této diplomové práci[1]. Tyto výsledky představují důležitý příspěvek v oblasti verifikace pravděpodobnostních programů. Konkrétněji, my rozšiřujeme existující metody pro agregovanou analýzu Markovových modelů tím, že jim dovolujeme zpracovávat řetězce s libovolnou strukturou stavového prostoru a zlepšovat odhad aproximační chyby. Vyhodnocení existujících aproximačních technik pro analýzu Markovských řetězců naznačuje, že nově navržené schéma výrazně překonává stávající přístupy a poskytuje zrychlení analýzy až pětkrát s odpovídající aproximační chybou ohraničenou na 0.1%. I přesto že v současné době tyto výsledky nelze přímo použít během syntézního procesu (navrhovaná rozšíření analýzy založené na agregaci předpokládají mírně odlišné specifikace), stále představují pro takovéto účely důležitý základ a budou sloužit jako odrazový můstek pro následný výzkum.

[1] Navrhovaná vylepšení vedla k článku (přiloženého jako doplňkový materiál) pro Performance Evaluation, prestižní mezinárodní časopis, v únoru 2020.

Computer-Aided Synthesis of Probabilistic Models

Declaration

I hereby declare that this master's thesis was prepared as an original work by the author under the supervision of RNDr. Milan Češka, Ph.D. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....
Roman Andriushchenko
June 17, 2020

Acknowledgements

My sincere gratitude goes to my supervisor, RNDr. Milan Češka, Ph.D., for his guidance and continuous support throughout my master's studies.

Contents

Chapter 1

Introduction

Designing a system exhibiting a desirable behavior – e.g. a network protocol allowing to increase the packet throughput, or selecting the optimal power management strategy – is a difficult task that involves reasoning over multiple alternative designs. One might start with the so-called *program sketch* [41, 5] – a system description with 'holes' (options) – and then consider filling these holes so that the obtained program satisfies a given specification. These holes may represent initial values of variables or even branching conditions that affect the topology of the underlying state space of the program. Furthermore, modeling a system with unpredictable or unreliable behavior calls for using mathematical apparata based on the probability theory, resulting in the so-called *probabilistic programs*. Application of probabilistic programs covers a wide variety of research fields, including e.g. analysis of software product lines [40, 22, 16], controller synthesis for partially observable models [30, 36], adaptive management [7, 20] or design of communication protocols [28, 34].

Using a Markov chain as the operational model of a probabilistic program, an automated synthesizer explores the family of such chains and picks the one that satisfies a given specification. Automated synthesis, in itself, represents a tremendous challenge, particularly due to the state-space explosion problem that affects the synthesis in a twofold manner: not only the number of candidate solutions is exponential wrt. the number of options being considered, the state space of each particular chain usually also grows exponentially wrt. the length of the description of the program. The problem of efficient model checking for Markov chains has been tackled using simulation approaches [24, 42] or various approximate methods. The latter include state-space truncation [19] and state aggregation techniques [1]. State-space truncation [19] bypasses the state explosion problem by dynamically neglecting states with insignificant probability, while state aggregation approach [1] employs clustering of the state space. In both cases, an approximation error has to be quantified and highly accurate estimates are crucial.

Returning to the synthesis problem, it can be naively solved by analyzing all individual family members – the so-called *one-by-one* approach [16, 17] – or by modeling the family of candidate solutions as a single *all-in-one* Markov decision process [16, 18], the size of which is proportional to the family size. Unfortunately, the double state-space explosion problem renders both of these approaches infeasible for large families. To circumvent this issue, the authors of [22] have successfully employed evolutionary search algorithms for the synthesis of software systems. However, the method remains incomplete and is unable to efficiently solve more challenging synthesis problems, e.g. which family member satisfies the specification *optimally*.

To provide completeness, more sophisticated methods have been proposed. These methods usually attempt to provide analysis for whole subfamilies of chains at once. In this work, we will focus on two frontline complete approaches for probabilistic synthesis. The first method is a *counterexample-guided inductive synthesis (CEGIS)* [13] that analyzes each chain one by one and exploits critical subsystems of counterexamples to prune all chains behaving incorrectly. The second approach is a *counterexample-guided abstraction refinement (CEGAR)* [14] that analyzes the complete design space at once and, if the analysis yields inconclusive results, partitions the family of candidate solutions into refined subfamilies that are handled analogously. While both of these techniques have demonstrated compelling results, they are not void of limitations and the two approaches are incomparable in the sense that one technique may be more suitable for specific classes of probabilistic programs, and vice versa.

Key contributions.

In this thesis, we consider a fusion of both inductive and abstracting approaches and develop a novel integrated method that combines the power of all-in-one abstraction with the precision of one-by-one analysis. In particular, the designed method first analyzes the family at once, in the spirit of CEGAR, collecting some general information about the worst-case and best-case behavior – wrt. the given specification – of each Markov chain in the family. Then, the method analyzes the family analogously to CEGIS, but uses the collected data to improve the quality of critical subsystems in order to accelerate pruning of the state space of candidate solutions. We evaluate all three methods – CEGIS, CEGAR, and a novel technique – on an extensive set of real-world case studies, and show that the newly designed approach is not only sound and complete, but it also mitigates differences between inductive and abstracting aproaches, enabling efficient synthesis of any particular model. From the performance perspective, the designed method is not only comparable with the state-of-the-art synthesis techniques, we demonstrate that, in most cases, it manages to significantly outperform existing approaches, sometimes by a margin of orders of magnitude.

Analysis of individual Markov chains represents a core stage of the synthesis pipeline and presents a challenge orthogonal to that of the synthesis of probabilistic systems. While synthesis deals with the exponential growth of a family size, analysis of each particular member deals with the explosion of the underlying state space of the chain. In this work, we also include a short presentation of selected results we have achieved in the area of approximation techniques for Markov models. The results were originally presented as a bachelor's thesis [6], but were since significantly extended and improved during the work on this master's thesis[1]. These results represent an important contribution to the verification of probabilistic programs. In particular, we extend existing frameworks for aggregation-based analysis of Markov models by allowing them to handle chains with an arbitrary structure of the underlying state space and improve upon existing bounds on the approximation error. We carry out a comparative evaluation of existing approximative techniques for Markov chain analysis and demonstrate that the newly designed scheme significantly outperforms existing approaches and provides a speedup of the analysis up to a factor of five with the corresponding approximation error bounded within 0.1%. Although, at the moment, these results cannot be directly used during the synthesis process – the proposed extensions of aggregation-based analysis assume slightly different specifications that those we consider

[1]The proposed improvements had led to a paper (attached as supplementary material) submitted to Performance Evaluation, a top-ranked international journal, in February 2020.

for synthesis – they still represent an important groundwork for such integration and will serve as a stepping stone to the follow-up research.

Structure of this paper.

In Chapter 2 we give an overview of the necessary theory regarding discrete-time Markov chains and Markov decision processes. In Chapter 3 we briefly present our selected results in the area of approximation techniques for Markov chains. In Chapter 4 we formulate a probabilistic synthesis problem and give an overview of two modern techniques for its solution: CEGIS and CEGAR. In Chapter 5 we develop key ideas associated with the integration of the two approaches and then carry out this integration in two ways: first rather naively, by blindly applying the developed integration principles, and then in a more sophisticated way, thus introducing the main integrated scheme. Then, in Chapter 6, we put the designed techniques to a test and compare them to state-of-the-art synthesis methods on a broad set of real-world case studies. Finally, Chapter 7 closes the thesis with the collection of notes and issues that could serve as a departure point for the follow-up research.

Chapter 2

Preliminaries

In this chapter, we review the necessary theory and introduce notation that will be used throughout the paper. We first cover the simplest probabilistic models – discrete-time Markov chains – representing an operational model for probabilistic programs. We describe basic techniques for their analysis, including the notion of critical subsystems, which will be important when introducing the counterexample-guided inductive synthesis (CEGIS) [13] method. We then describe a slightly more complex construct – Markov decision processes. A Markov decision process represents the same Markov chain, but is equipped with extra nondeterminism. Markov decision processes will play an essential role when describing counterexample-guided abstraction refinement (CEGAR) [14], a counterpart of CEGIS.

2.1 Discrete-Time Markov Chains

Definition 1. [13] Let S be a finite set. A *(discrete) probability distribution* on S is a function $\mu : S \to [0,1]$ s.t. $\sum_{s \in S} \mu(s) = 1$. Let $Distr(S)$ denote the set of all probability distributions on S. The support of a distribution μ is $\text{supp}(\mu) = \{s \in S \mid \mu(s) > 0\}$.

Example 1. Let $S = \{s_0, s_1, s_2, s_3\}$. A function $\mu : S \to [0,1]$ defined as $\mu : [s_0 \mapsto \frac{1}{2}, s_1 \mapsto \frac{1}{6}, s_2 \mapsto \frac{1}{3}, s_3 \mapsto 0]$ is a probability distribution on S, i.e. $\mu \in Distr(S)$. The support of μ is $\text{supp}(\mu) = \{s_0, s_1, s_2\}$. To simplify notation, we might write this distribution as $\mu = \frac{1}{2} : s_0 + \frac{1}{6} : s_1 + \frac{1}{3} : s_2$.

Definition 2. A *discrete-time Markov chain (MC)* is a tuple $D = (S, s_{init}, \boldsymbol{P})$, where S is a finite set of states, $s_{init} \in S$ is an initial state and $\boldsymbol{P} : S \to Distr(S)$ is a transition probability matrix.

Intuitively, an MC is a state-transition system where, for each state $s \in S$, a probability distribution $\boldsymbol{P}(s)$ represents a stochastic choice of transitioning from state s to one of its *successor states* from the set $\text{supp}(\boldsymbol{P}(s))$. In the following, we will write $\boldsymbol{P}(s,t)$ to denote $\boldsymbol{P}(s)(t)$. Under this semantics, it follows that, at any particular time, the probability of transitioning from state s to state $t \in \text{supp}(\boldsymbol{P}(s))$ is constant and is independent of the path the chain has already taken to reach state s - this is known as the *Markov property* (memorylessness). We say that the state s is *absorbing* iff $\boldsymbol{P}(s) = 1 : s$, i.e. $\boldsymbol{P}(s,s) = 1$. Sometimes it is helpful to lay out the chain in the so-called *transition probability graph*, whose nodes are the states and whose arcs are non-zero transitions, as illustrated in Figure 2.1.

A *path* π of an MC, representing one possible execution of the chain, is a (possibly infinite) sequence of states $s_0 s_1 s_2 \ldots$, where $s_0 = s_{init}$ and $\forall i \in \mathbb{N}_0 : \boldsymbol{P}(s_i, s_{i+1}) \geq 0$. Let

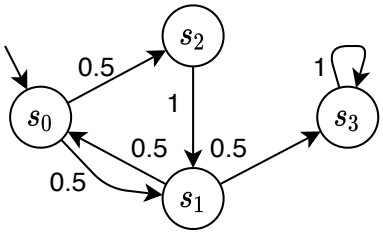


Figure 2.1: A simple Markov chain.

$\text{Paths}^D(s)$ and $\text{Paths}^D_{\text{fin}}(s)$ denote a set of all infinite and finite paths taken from state s, respectively. For a finite path $\pi = s_0 s_1 \dots s_n$, let $\text{last}(\pi) = s_n$ denote its last state. Markov property allows us to quantify the probability of individual (finite) paths using the transition probability matrix: $\mathbb{P}[s_0 s_1 \dots s_n] = \prod_{i=0}^{n-1} \boldsymbol{P}(s_i, s_{i+1})$.

Example 2. Consider a Markov chain with the initial state s_0 depicted in Figure 2.1. A path $\pi = s_0 s_1 s_0 s_2 s_1$ is one possible trajectory of this MC. The probability that π is executed is $\boldsymbol{P}(s_0, s_1) \cdot \boldsymbol{P}(s_1, s_0) \cdot \boldsymbol{P}(s_0, s_2) \cdot \boldsymbol{P}(s_2, s_1) = \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot 1 = \frac{1}{8}$. The probability that the chain eventually reaches state s_2 can be computed by summing the probabilities of all finite paths that end up at s_2. In this particular case, where the chain has a rather trivial structure, the sum can be computed by hand:

$$\mathbb{P}[s_0 s_2] + \mathbb{P}[s_0 s_1 s_0 s_2] + \mathbb{P}[s_0 s_1 s_0 s_1 s_0 s_2] + \dots = \frac{1}{2} + \frac{1}{8} + \frac{1}{32} + \dots = \frac{2}{3}.$$

2.1.1 Model Checking MCs

In general, a stochastic model checking [32] is a method for verifying whether a system exhibits a certain property by calculating the likelihood of occurrence of various events during its execution. Model checking algorithms take as input a description of a model along with specification expressed in probabilistic temporal logic and return a probability for a given model to satisfy this property. In the context of Markov chains, an essential property that will be the primary focus of this work is (unbounded) *reachability*: for a set $T \subseteq S$ of *target states*, let $\mathbb{P}[s \models \text{F } T]$ denote the probability of, by starting in the state $s \in S$, eventually reaching any of the states in T. Qualitative property is then of the form $\varphi \equiv \mathbb{P}_{\bowtie\lambda}[\text{F } T]$, where $\lambda \in [0,1] \cap \mathbb{Q}$ and $\bowtie \in \{<, \leq, >, \geq\}$. Property φ is satisfied in state s iff the corresponding probability $\mathbb{P}[s \models \text{F } T]$ meets threshold λ, i.e. $s \models \varphi :\Leftrightarrow \mathbb{P}[s \models \text{F } T] \bowtie \lambda$. Finally, an MC D satisfies a property φ iff it is satisfied in the initial state: $D \models \varphi :\Leftrightarrow s_{init} \models \varphi$. Computation of reachability represents a cornerstone technique of MC model checking since deciding more complex properties – e.g. formulae in probabilistic computation tree logic (PCTL) or ω-regular properties – can be reduced to the computation of reachability.

Model checking an MC D against a reachability property $\varphi \equiv \mathbb{P}_{\bowtie\lambda}[\text{F } T]$ proceeds by calculating, for each state $s \in S$, exact probabilities $\mathbb{P}[s \models \text{F } T]$ of eventually reaching any of the states in T from state s, and then checking whether $\mathbb{P}[s_{init} \models \text{F } T] \bowtie \lambda$. These exact values of reachability probabilities are computed as a solution to a linear system of equations, as summarized in Algorithm 1.

Example 3. Let us return to Example 2, where we derived that $\mathbb{P}[s_0 \models \text{F } \{s_2\}] = \frac{2}{3}$. We can arrive at the same value systematically by applying the Algorithm 1. We see that

Algorithm 1: Computing unbounded reachability probabilities.

Input : MC $D = (S, s_{init}, \boldsymbol{P})$, a set of target states $T \subseteq S$.
Output: Values of $\boldsymbol{x}(s) = \mathbb{P}[s \models \mathrm{F}\ T]$ for each $s \in S$.

Function `reachabilityMC(`D, T`):`
  $S_0 \leftarrow \{s \in S \mid \text{reachabilityMC}(s, T) = 0\}$ `// a graph problem`
  $S_1 \leftarrow T$
  $S_? \leftarrow S \setminus (S_0 \cup S_1)$
  Find the solution $\boldsymbol{x}$ of the following system of equations:

$$\boldsymbol{x}(s) = \begin{cases} 0 & \text{if } s \in S_0, \\ 1 & \text{if } s \in S_1, \\ \sum_{s' \in S} \boldsymbol{P}(s, s') \cdot \boldsymbol{x}(s') & \text{if } s \in S_?. \end{cases} \tag{2.1}$$

  return $\boldsymbol{x}$

$S_1 = \{s_2\}$ and that $S_0 = \{s_3\}$ (consider MC as a directed unweighted graph and identify states from which the set $T = \{s_2\}$ is unreachable). We then solve the following system of equations:

$$\begin{aligned} \boldsymbol{x}(s_0) &= \frac{1}{2}\boldsymbol{x}(s_1) + \frac{1}{2}\boldsymbol{x}(s_2) \\ \boldsymbol{x}(s_1) &= \frac{1}{2}\boldsymbol{x}(s_0) + \frac{1}{2}\boldsymbol{x}(s_3) \\ \boldsymbol{x}(s_2) &= 1 \\ \boldsymbol{x}(s_3) &= 0 \end{aligned}$$

obtaining vector $\boldsymbol{x} = (\frac{2}{3}, \frac{1}{3}, 1, 0)^T$. Having computed this vector, we can now easily check, for instance, that $D \not\models \mathbb{P}_{\leq 0.6}[\mathrm{F}\ \{s_2\}]$ and $D \models \mathbb{P}_{\geq 0.5}[\mathrm{F}\ \{s_2\}]$.

2.1.2 Counterexamples for MCs

If the system refutes a given specification φ, it is sometimes useful to provide a *counterexample* in the form of a concrete execution of the system that violates φ. In the context of Markov chains and probabilistic model checking, a counterexample [2] is actually a (finite) set of paths, the probabilities of which add up to a quantity that violates $\bowtie \lambda$. Based on the form of the operator $\bowtie$, we distinguish between two main categories of reachability properties $\varphi \equiv \mathbb{P}_{\bowtie\lambda}[\mathrm{F}\ T]$. A *safety* property is a reachability property φ with $\bowtie \in \{<, \leq\}$. A chain D violates a safety property if it is possible, with probability exceeding λ, to reach any of the states in T. In other words, a counterexample to φ would be a set of *finite* paths that satisfy F T and add up to more than λ of the probability. A *liveness* property is a property φ with $\bowtie \in \{>, \geq\}$. A chain D violates a liveness property if it is possible, with probability exceeding $(1 - \lambda)$, to never reach any of the states in T. That is, a counterexample to φ would be a set of *infinite* paths that violate F T and add up to more than $(1 - \lambda)$ of the probability. In our case, where the state space of the chain is finite, a set of

infinite paths can be represented by a finite prefix (see the notion of cylinder sets of paths e.g. in [32]) that ends in a bottom strongly-connected component of D, from which none of the states in T is reachable - a set of such prefixes that constitute more than $(1-\lambda)$ of the probability then represents a counterexample. In the following text, safety (liveness) properties are considered of the form $\mathbb{P}_{\leq\lambda}[\mathrm{F}\ T]$ ($\mathbb{P}_{\geq\lambda}[\mathrm{F}\ T]$): properties with $<$ ($>$) are handled similarly.

Example 4. Previously, in Example 3, we have demonstrated that the Markov chain D from Figure 2.1 violates specification $\mathbb{P}_{\leq 0.6}[\mathrm{F}\ \{s_2\}]$. A counterexample to this property would be e.g. a set $\{s_0s_2, s_0s_1s_0s_2\}$ of two finite paths that satisfy F $\{s_2\}$ and amount to a probability of $\frac{1}{2}+\frac{1}{8}=0.625>0.6$. Similarly, to show that $D \not\models \mathbb{P}_{\geq 0.9}[\mathrm{F}\ \{s_2\}]$, we need to 'trap' at least $1-0.9=0.1$ of the probability mass in paths that never reach $\{s_2\}$. One can see, for instance, that all (infinite) paths sharing prefix $s_0s_1s_3$ indeed violate F $\{s_2\}$ and amount to a probability of at least $\frac{1}{2}\cdot\frac{1}{2}=0.25>0.1$. Therefore, a set $\{s_0s_1s_3\}$ represents a valid counterexample for a liveness property $\mathbb{P}_{\geq 0.9}[\mathrm{F}\ \{s_2\}]$.

The problem with representing counterexamples as sets of paths is that, sometimes, providing a user with a large set of possible executions of chain D that violate F T can be quite overwhelming. A more compact approach is to provide a user with the so-called *critical subsystem* $D{\downarrow}C$: a fraction of the original chain D that contains enough of paths that violate F T. Essentially, a critical subsystem $D{\downarrow}C$ includes only the *critical states* (those in $C \subseteq S$), and therefore contains only some of the paths of the full system. Then, a set of all paths $\mathrm{Paths}^{D\downarrow C} \cup \mathrm{Paths}^{D\downarrow C}_{\mathrm{fin}}$ executable in the critical subsystem $D{\downarrow}C$ corresponds to a counterexample. In the following definitions, we formalize this notion for safety properties. A representation of counterexamples based on critical subsystems can be used for liveness properties as well, although the corresponding construction algorithms require additional manipulations with the model and/or the property. For more details, please refer to [2].

Definition 3. Let $D=(S, s_{init}, \boldsymbol{P})$ be an MC with $s_\perp \notin S$ and let $C \subseteq S$ with $s_{init} \in C$. The *sub-MC* of D wrt. C is an MC $D{\downarrow}C = (C \cup \{s_\perp\}, s_{init}, \boldsymbol{P'})$, where the transition probability matrix $\boldsymbol{P'}$ is defined as follows:

$$\boldsymbol{P'}(s,s') = \begin{cases} \boldsymbol{P}(s,s') & \text{if } s,s' \in C, \\ 1-\sum_{s''\in S\setminus C} \boldsymbol{P}(s,s'') & \text{if } s \in C \text{ and } s' = s_\perp, \\ 1 & \text{if } s = s' = s_\perp. \end{cases}$$

Definition 4. Let $D=(S, s_{init}, \boldsymbol{P})$ be an MC and let $\varphi \equiv \mathbb{P}_{\leq\lambda}[\mathrm{F}\ T]$ be a safety property s.t. $D \not\models \varphi$. If, for some set C, it holds $D{\downarrow}C \not\models \varphi$, then this set C and the corresponding subsystem $D{\downarrow}C$ are called *critical*. A critical set C is called *minimal* iff $|C| \leq |C'|$ for all critical sets C'.

Both counterexample representations – as a set of paths or via a critical subsystem – have their advantages and both have dedicated and well-researched methods for their calculation. Keeping in mind what follows, we will restrict ourselves to the representation based on critical sets and reserve the term *counterexample* for the corresponding critical subsystem. Algorithms for counterexample generation [2] usually proceed by gradually building up a reachable state space of D by adding more and more states, therefore introducing more and more paths. After adding a state, the obtained subsystem is model checked and, if the property φ is satisfied (implying that there are still not enough paths leading to T), expansion of the reachable state space continues. Once the subsystem contains enough paths

to violate φ, the counterexample is completed. The procedure is guaranteed to terminate since we assume that $D{\downarrow}S \not\models \varphi$. Notice that a chain D can have multiple critical sets C, although we are usually interested in the most compact representation, i.e. we wish to find a critical set with the least number of states. The quality (wrt. the number of states) of a critical set is, naturally, determined by the specific order in which the states are added during the counterexample construction: ideally, we want to add only those states that would yield the shortest and most probable paths leading to target states T. This inexact formulation makes space for a number of heuristics, most notable ones include e.g. Best-First search [3] or its enhanced variant eXtended Best-First search [4].

Example 5. Consider again the Markov chain D from Figure 2.1 (reproduced again in Figure 2.2a) and a safety property $\varphi \equiv \mathbb{P}_{\leq 0.6}[\mathrm{F}\ \{s_2\}]$. As we have argued previously in Example 4, a set $CE_\pi = \{s_0 s_2, s_0 s_1 s_0 s_2\}$ represents a path-based counterexample for φ. In Figure 2.2a, the transitions that induce these critical paths are highlighted in red. The corresponding critical set is $C = \{s_0, s_1, s_2\}$: these states induce paths among which we can find paths in CE_π. Therefore, a sub-MC $D{\downarrow}C$ depicted in Figure 2.2b is a critical subsystem of D wrt. specification φ: notice that we do not need states in $S \backslash C$ (in our case, states in $\{s_3\}$) to refute φ, therefore, we replace them with an absorbing dummy state $s_\perp$ which represents a 'sink' for all states and corresponding transitions that are omitted from the critical subsystem. Here we would like to mention one additional detail that will play a crucial important later on. Notice that, to provide user with a counterexample, we do not even need to include s_2 to a set C of critical states. Yes, this violates Definition 4 of a critical subsystem, but, for the sake of argument, instead of considering which *states* are important to refute φ, let us again return to the path-based representation of a counterexample, and consider which *transitions* induce violating paths. Indeed, from the diagram in Figure 2.2a we see that only transitions emanating from states s_0 and s_1 are decisive when we refute φ. Hence, as Figure 2.2c suggests, if we replace all target states with one target state $s_\top$, the resulting transformation D' violates $\varphi' \equiv \mathbb{P}_{\leq 0.6}[\mathrm{F}\ \{s_\top\}]$ for the same reason why D violates φ, therefore representing (a semantical equivalent of) a valid counterexample.

2.2 Markov Decision Processes.

Each state of a Markov chain has a unique probability distribution over its successors. In this section we introduce an extension of MCs that introduces for each state a nondeterministic choice between multiple probability distributions.

Definition 5. [14] A *Markov decision process (MDP)* is a tuple $M = (S, s_{init}, Act, \mathcal{P})$, where S and s_{init} are as before, Act is a finite set of actions and $\mathcal{P} : S \times Act \nrightarrow Distr(S)$ is a partial transition probability function. The set $Act(s) = \{a \in Act \mid \mathcal{P}(s)(a) \neq \perp\}$ denotes the set of available actions in state $s \in S$.

During its execution, an MDP that currently resides as state $s \in S$ has a nondeterministic choice of an action $a \in Act(s)$ yielding one possible probability distribution $\mathcal{P}(s)(a)$ over possible successors. For simplicity, we will write $\mathcal{P}(s, a, s')$ do denote $\mathcal{P}(s)(a)(s')$ in cases where $\mathcal{P}(s)(a)$ is defined. A path of an MDP M is a (possibly infinite) sequence $\pi = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \ldots$, where $\forall i \in \mathbb{N}_0 : \mathcal{P}(s_i, a_i, s_{i+1}) > 0$. As before, let $\mathrm{Paths}^M(s)$ and $\mathrm{Paths}^M_{\mathrm{fin}}(s)$ denote a set of all infinite and finite paths taken in M from state s, respectively, and let $\mathrm{last}(\pi) = s_n$ again denote the last state of a finite path $\pi = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \ldots \xrightarrow{a_{n-1}} s_n$. The probability of such path is evaluated similarly to

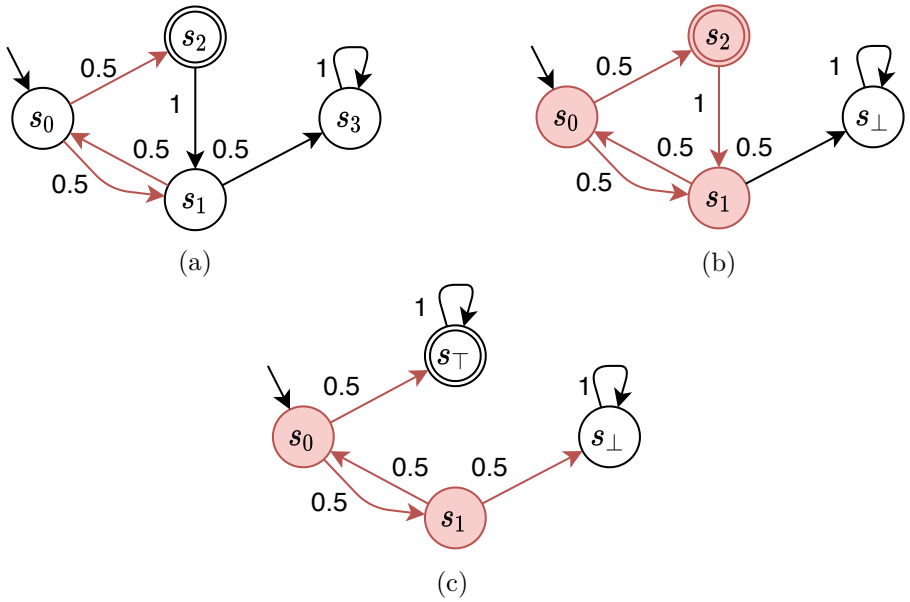


Figure 2.2: Counterexample construction for a simple Markov chain and a safety property $\mathbb{P}_{\leq 0.6}[\mathrm{F}\ \{s_1\}]$.

the path of an MC: $\mathbb{P}[\pi] = \prod_{i=0}^{n-1} \mathcal{P}(s_i, a_i, s_{i+1})$. Although MDP behaves nondeterministically due to the presence of actions, one might use a *scheduler* to resolve this nondeteremeinism.

Definition 6. A *(deterministic) scheduler* for an MDP $M = (S, s_{init}, Act, \mathcal{P})$ is a function $\sigma : \mathrm{Paths}^M_{\mathrm{fin}} \to Act$ such that $\sigma(\pi) \in Act(\mathrm{last}(\pi))$ for all $\pi \in \mathrm{Paths}^M_{\mathrm{fin}}$. The set of all schedulers for M is denoted as Σ^M.

When MDP M enters a state $\mathrm{last}(\pi)$ via a path π, a scheduler σ deterministically chooses an action $a = \sigma(\pi) \in Act(\mathrm{last}(\pi))$ and the system proceeds to one of the successors of state $\mathrm{last}(\pi)$ according to the distribution $\mathcal{P}(\mathrm{last}(\pi), a)$. The resulting stochastic process is devoid of any nondetereminism, thus representing an infinite-state Markov chain.

Definition 7. Let $M = (S, s_{init}, Act, \mathcal{P})$ be an MDP. We say that the scheduler $\sigma \in \Sigma^M$ *induces an MC* $M^\sigma = (\mathrm{Paths}^M_{\mathrm{fin}}, s_{init}, \boldsymbol{P}^\sigma)$ iff $\boldsymbol{P}^\sigma(\pi, \pi \xrightarrow{\sigma(\pi)} s') = \mathcal{P}(\mathrm{last}(\pi), \sigma(\pi), s')$ and $\boldsymbol{P}^\sigma(\cdot, \cdot) = 0$ otherwise.

If a scheduler chooses an action solely based on the current state $\mathrm{last}(\pi)$, and not on the path π that has been taken so far, then such scheduler is called *memoryless*.

Definition 8. Scheduler $\sigma \in \Sigma^M$ of an MDP $M = (S, s_{init}, Act, \mathcal{P})$ is called *memoryless* iff for all $\pi, \pi' \in \mathrm{Paths}^M_{\mathrm{fin}}$, $\mathrm{last}(\pi) = \mathrm{last}(\pi') \Rightarrow \sigma(\pi) = \sigma(\pi')$.

A memoryless scheduler essentially assigns to each state $s \in S$ an action $\sigma(s)$ that will be taken whenever a process M enters s. This allows us to ignore actions from the set $Act(s) \backslash \{\sigma(s)\}$ completely, thus obtaining a finite-state Markov chain consistent with Definition 2, as illustrated in the Proposition 1.

Proposition 1. Let $M = (S, s_{init}, Act, \mathcal{P})$ be an MDP and let $\sigma \in \Sigma^M$ be a memoryless scheduler. An MC M^σ induced by M and σ is homomorphic to an MC $(S, s_{init}, \boldsymbol{P}^\sigma)$, where $\boldsymbol{P}^\sigma(s) \equiv \mathcal{P}(s, \sigma(s))$.

Example 6. Consider an MDP $M = (S, s_0, Act, \mathcal{P})$ with $S = \{s_0, s_1, s_2, s_3\}$, $Act = \{a_0, a_1, a_2, a_3\}$ and $\mathcal{P}$ encoded in a diagram depicted in Figure 2.3. In a diagram, for each state $s \in S$, outgoing edges lead to actions $a \in Act(s)$ that yield a concrete probability distribution over successor states. For instance, in the initial state s_0, we have a choice between three actions ($Act(s_0) = \{a_0, a_1, a_3\}$) where action a_0 leads deterministically to state s_1, action a_1 leads deterministically to state s_2, and action a_3 yields a 50/50 distribution between states s_1 and s_2. Meanwhile, action a_2 is not available in state s_0: $\mathcal{P}(s_0, a_2) = \bot$. One possible execution of M might be the path $\pi = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_0 \xrightarrow{a_1} s_2 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_3 \xrightarrow{a_2} s_3$. The probability of executing π can be computed as

$$\begin{aligned}\mathbb{P}[\pi] &= \mathcal{P}(s_0, a_0, s_1) \cdot \mathcal{P}(s_1, a_1, s_0) \cdot \mathcal{P}(s_0, a_1, s_2) \cdot \mathcal{P}(s_2, a_0, s_1) \cdot \mathcal{P}(s_1, a_1, s_3) \cdot \mathcal{P}(s_3, a_2, s_3) \\ &= 1 \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot 1 \cdot \frac{1}{2} \cdot 1 = \frac{1}{8}.\end{aligned}$$

Now consider a memoryless scheduler $\sigma \in \Sigma^M$ defined as

$$\sigma : [s_0 \mapsto a_1, s_1 \mapsto a_1, s_2 \mapsto a_0, s_3 \mapsto a_2].$$

Applying scheduler σ on M is equivalent to dropping, in each state $s \in S$, all actions except $\sigma(s)$, thus resolving the nondeterminism and inducing a Markov chain M^σ homomorphic to a chain depicted previously in Figure 2.1.

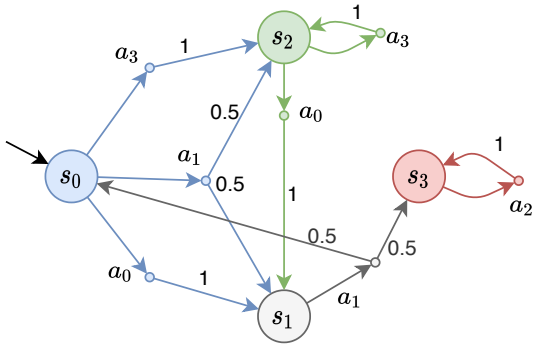


Figure 2.3: A simple Markov decision process.

2.2.1 Model Checking MDPs

For model checking MDPs, we use the same specification language as for MCs, although the properties now have slightly different semantics. In particular, we say that a specification φ holds for an MDP M iff it holds for the induced MCs of all schedulers, that is, $M \models \varphi :\Leftrightarrow \forall \sigma \in \Sigma^M : M^\sigma \models \varphi$. Instead of enumerating over an infinite set of schedulers, it is sufficient to consifer only memoryless schedulers, as illustrated in Proposition 2.

Proposition 2. [21] Let $M = (S, s_{init}, Act, \mathcal{P})$ be an MDP and φ be a property. Then there exist *memoryless* schedulers $\sigma_{\min}, \sigma_{\max} \in \Sigma^M$ such that $\forall \sigma \in \Sigma^M : \mathbb{P}[M^{\sigma_{\min}} \models \varphi] \leq \mathbb{P}[M^\sigma \models \varphi] \leq \mathbb{P}[M^{\sigma_{\max}} \models \varphi]$.

Proposition 2 is applied as follows. If $\varphi \equiv \mathbb{P}_{\leq\lambda}[\mathrm{F}\ T]$ is a safety formula, we compute a maximizing scheduler $\sigma_{\max}$ and set $M \models \varphi \Leftrightarrow M^{\sigma_{\max}} \models \varphi \Leftrightarrow \mathbb{P}[M^{\sigma_{\max}} \models \mathrm{F}\ T] \leq \lambda$. Similarly to deciding reachability for MCs, MDP model checking against reachability properties boils down to computing, for each $s \in S$, a maximum probability $\boldsymbol{x}_{\max}(s) \coloneqq \max_{\sigma\in\Sigma^M} \mathbb{P}[M^\sigma, s \models \mathrm{F}\ T]$ of reaching T, and then asserting that $\boldsymbol{x}_{\max}(s_{init}) \leq \lambda$. This upper bound $\boldsymbol{x}_{\max}$ on the reachability probability can be obtained as a solution to a mixed-integer linear program (MILP), as illustrated in Algorithm 2. We included this algorithm to help us later prove some key theorems, although be aware that modern model checkers apply heuristic methods for estimating reachability probability, e.g. value iteration or policy iteration methods [21]. Notice that, in general, for the purpose of model checking, we are primarily interested in maximum probabilities and not the corresponding maximizing scheduler $\sigma_{\max}$, although the latter can be easily obtained by checking which of the constraints $\boldsymbol{x}_{\max}(s) \geq \sum_{s'\in S} \mathcal{P}(s,a,s') \cdot \boldsymbol{x}_{\max}(s')$ is satisfied as equality. Similarly, deciding liveness properties involves computing, for each state $s \in S$, minimum probabilites $\boldsymbol{x}_{\min}$ of reaching target states and then checking whether $\boldsymbol{x}_{\min}(s_{init}) \geq \lambda$. The computation of $\boldsymbol{x}_{\min}$ is carried out analogously to Algorithm 2 up to the interchange of some operators.

Algorithm 2: Unbounded reachability probabilities for MDP.

Input : MDP $M = (S, s_{init}, Act, \mathcal{P})$, a set $T \subseteq S$ of target states.
Output: Values of $\boldsymbol{x}_{\max}(s) = \max_{\sigma\in\Sigma^M} \mathbb{P}[M^\sigma, s \models \mathrm{F}\ T]$ for each $s \in S$.

Function `reachabilityMDP(`M, T`):`
  $S_0 \leftarrow \{s \in S \mid \forall \sigma \in \Sigma^M : \mathbb{P}[M^\sigma, s \models \varphi] = 0\}$ `// a graph problem`
  $S_1 \leftarrow \{s \in S \mid \exists \sigma \in \Sigma^M : \mathbb{P}[M^\sigma, s \models \varphi] = 1\}$ `// a graph problem`
  $S_? \leftarrow S \setminus (S_0 \cup S_1)$
  Find $\boldsymbol{x}_{\max}$ as the solution to the following linear program:
  $\boldsymbol{x}_{\max} = \arg\min_{\boldsymbol{x}\in[0,1]^{|S|}} \sum_{s\in S_?} \boldsymbol{x}(s)$ subject to

$$\begin{aligned} \forall s \in S_0 &: \boldsymbol{x}(s) = 0, \\ \forall s \in S_1 &: \boldsymbol{x}(s) = 1, \\ \forall s \in S_? \ \forall a \ \in Act(s) &: \boldsymbol{x}(s) \geq \sum_{s'\in S} \mathcal{P}(s,a,s') \cdot \boldsymbol{x}(s'). \end{aligned}$$

  return $\boldsymbol{x}_{\max}$

Example 7. Assume an MDP M from Example 6 and a safety specification $\varphi \equiv \mathbb{P}_{\leq 0.6}[\mathrm{F}\ \{s_2\}]$. To check whether $M \models \varphi$, we first compute, for each state $s \in S$, an upper bound $\boldsymbol{x}_{\max}$ on the reachability probability across all (memoryless) schedulers. Following Algorithm 2, it is easy to see that $S_0 = \{s_3\}$ and $S_1 = \{s_0, s_2\}$. Therefore, $\boldsymbol{x}_{\max}(s_0) = \boldsymbol{x}_{\max}(s_2) = 1$, $\boldsymbol{x}_{\max}(s_3) = 0$ and $\boldsymbol{x}_{\max}(s_1)$ is a minimum value for which

$$\boldsymbol{x}_{\max}(s_1) \geq 0.5 \cdot \boldsymbol{x}_{\max}(s_0) + 0.5 \cdot \boldsymbol{x}_{\max}(s_3) = 0.5,$$

i.e. $\boldsymbol{x}_{\max}(s_1) = 0.5$. We obtain an upper bound $\boldsymbol{x}_{\max} = (1, 0.5, 1, 0)^T$ and can see that $\boldsymbol{x}_{\max}(s_0) = 1 > 0.6$, concluding that $M \not\models \varphi$: there exist schedulers which induce Markov chains violating φ. Examples of such schedulers include e.g. schedulers that map state s_0 to action a_3; another example is a scheduler σ we investigated previously in Example 6: since

σ induces the chain D we considered in the previous section, we know that it violates $\varphi \equiv \mathbb{P}_{\leq 0.6}[\mathrm{F}\ \{s_2\}]$. Furthermore, if we take vector $\boldsymbol{x} = (\frac{2}{3}, \frac{1}{3}, 1, 0)^T$ of exact values of reachability probabilities for Markov chain D we computed in Example 3, we can additionally confirm that $\boldsymbol{x} \leq \boldsymbol{x}_{\max}$.

Chapter 3

Adaptive Aggregation of Markov Chains

As was outlined previously in the introductory Chapter 1, analysis of individual Markov chains represents a core stage of the synthesis pipeline and presents a challenge orthogonal to that of the synthesis of probabilistic systems. While synthesis deals with the exponential growth of a family size, analysis of each particular member deals with the explosion of the underlying state space of the chain. In this chapter we tackle the second problem and provide a short presentation of selected results we have achieved in the area of approximation techniques for Markov models. The results were originally presented as a bachelor's thesis [6], but were since significantly extended and improved during the work on this thesis[1]. These results represent an important contribution to the verification of probabilistic programs. In particular, the techniques improve the performance and scalability of the verification process by reducing the state space of a given probabilistic model while providing the upper bound on the approximation error. As such, they simplify the evaluation of candidate programs explored during the synthesis process.

Problem statement

Let $D = (S, \boldsymbol{p}_0, \boldsymbol{P})$ be a Markov chain with the state space S and transition probability matrix $\boldsymbol{P}$. Here, instead of starting process D from initial state s_{init}, we initialize the chain with the initial probability distribution $\boldsymbol{p}_0$. Also, rather than viewing transition probability matrix $\boldsymbol{P}$ as a collection of probability distributions, in this chapter we will primarily interpret $\boldsymbol{P}$ as an actual matrix $\boldsymbol{P} : S \times S \to [0, 1]$ (hence the name). Since $\forall s \in S : \sum_{s' \in S} \boldsymbol{P}(s, s') = 1$, we say that the matrix $\boldsymbol{P}$ is *stochastic*. *Transient analysis* of an MC D considers the problem of computing, for a given time horizon $k > 0$, a *transient probability distribution* $\boldsymbol{p}_k \in Distr(S)$ of D, where the value $\boldsymbol{p}_k(s)$ denotes the probability that process D resides at state s after k time steps. These probabilities can be computed using the following recurrent formula:

$$\boldsymbol{p}_k(s) = \sum_{r \in S} \boldsymbol{p}_{k-1}(r) \boldsymbol{P}(r, s), \tag{3.1}$$

or, using matrix notation, $\boldsymbol{p}_k = \boldsymbol{p}_{k-1} \cdot \boldsymbol{P}$, where $\boldsymbol{p}_i$ are row vectors of transient probabilities.

[1] The proposed improvements had led to a paper (attached as supplementary material) submitted to Performance Evaluation, a top-ranked international journal, in February 2020.

Computing transient probability distribution for a given Markov chain plays an important role when deciding *bounded* reachability properties: formulae of the form $\mathbb{P}_{\bowtie\lambda}[\mathrm{F}^{\leq k}\ T]$ representing reachability of $T \subseteq S$ in a limited time horizon k. Computing vector $\boldsymbol{p}_k$ directly using (3.1) suffers from the state explosion problem, and the primary focus of this chapter will be to develop methods providing an efficient and accurate approximation of $\boldsymbol{p}_k$. As was mentioned previously, extending these results to unbounded reachability (which is considered in the rest of the thesis) represents an open challenge, as discussed in Chapter 7.

3.1 State Space Clustering

Let $D = (S, \boldsymbol{p}_0, \boldsymbol{P})$ be an MC and assume that we are interested in approximating its transient probability distribution $\boldsymbol{p}_k$, $k > 0$. Let Φ be a partition of the state space S. We treat clusters in Φ as abstract states of a new aggregated chain $\Delta = (\Phi, \boldsymbol{\pi}_0, \boldsymbol{\Pi})$, where $\boldsymbol{\Pi}$ represents a suitable abstract transition probability matrix $\boldsymbol{\Pi} \colon \Phi \times \Phi \to \mathbb{R}_{\geq 0}$. The chain Δ is initialized using the probability distribution $\boldsymbol{\pi}_0 \colon \Phi \to \mathbb{R}_{\geq 0}$ computed as

$$\boldsymbol{\pi}_0(\sigma) = \sum_{s \in \sigma} \boldsymbol{p}_0(s), \tag{3.2}$$

i.e. the probability of residing in cluster σ is the sum of transient probabilities for the concrete states in σ. We can now recursively compute the transient probability distribution $\boldsymbol{\pi}_k$ of Δ using vector-matrix multiplications, similar to (3.1), as:

$$\boldsymbol{\pi}_k = \boldsymbol{\pi}_{k-1} \cdot \boldsymbol{\Pi}. \tag{3.3}$$

Having obtained $\boldsymbol{\pi}_k$, i.e. the probability of residing in individual clusters at time k, the approximation $\tilde{\boldsymbol{p}}_k \colon S \to \mathbb{R}_{\geq 0}$ of the transient probability distribution $\boldsymbol{p}_k$ is defined as

$$\tilde{\boldsymbol{p}}_k(s) = \frac{\boldsymbol{\pi}_k(\sigma)}{|\sigma|}, s \in \sigma, \tag{3.4}$$

where $|\sigma|$ denotes the size of cluster σ (namely the number of concrete states comprising it). That is, the probability of residing in cluster σ is distributed uniformly among its concrete states. To recapitulate, state-space aggregation is the process where we cluster the state space S into Φ and treat these clusters as states of a new abstract Markov chain Δ, where we assume that, if Δ resides at cluster $\sigma \in \Phi$, then D resides in one of the states $s \in \sigma$ with equal probability. Transitions of chain Δ are encoded in matrix $\boldsymbol{\Pi}$, which, as its definition suggests, might not necessarily be stochastic. As a consequence, probability vectors $\boldsymbol{\pi}_k$ might not necessarily be probability distributions according to the Definition 1 (elements of $\boldsymbol{\pi}_k$ might not sum to one). However, we still want to associate elements of $\boldsymbol{\Pi}$ with transition probabilities and elements of $\boldsymbol{\pi}_k$ with transient probabilities of the aggregated model. To avoid any confusion, we will reserve the term 'stochastic' for matrices and vectors that satisfy the corresponding normalization property. Lack of stochasticity should not discourage us from using such abstractions: function $\boldsymbol{\Pi}$ (respectively, $\boldsymbol{\pi}_k$) simply serves as a higher-level representative of function $\boldsymbol{P}$ (respectively, $\boldsymbol{p}_k$) on new state space and provides us with an approximation of its concrete counterpart.

A specific choice of an aggregation scheme (a shape of the abstract transition matrix $\boldsymbol{\Pi}$) decides how accurate approximation $\tilde{\boldsymbol{p}}_k$ will be. Estimating approximation error is

the interest of Section 3.2. For now, let us simply present three possible ways to define $\mathbf{\Pi}$ and discuss their applicability.

State-space aggregation based on average incoming transition probabilities

The approximate transition matrix for this aggregation is defined as follows:

$$\mathbf{\Pi}_{in}(\rho, \sigma) = \frac{1}{|\sigma|} \sum_{r \in \rho} \sum_{s \in \sigma} P(r, s). \tag{3.5}$$

The intuition behind this equation is that it encompasses the average *incoming* probability to cluster σ from cluster ρ. This shape of the transition matrix $\mathbf{\Pi}$ was previously introduced in [1] and the corresponding error bound on the approximation error (discussed in Section 3.2) was derived specifically for this scheme.

State-space aggregation based on average outgoing transition probabilities

This scheme is similar to the previous one, except that now we utilize average outgoing transition probabilities:

$$\mathbf{\Pi}_{out}(\rho, \sigma) = \frac{1}{|\rho|} \sum_{r \in \rho} \sum_{s \in \sigma} P(r, s). \tag{3.6}$$

In [6] we show that outgoing averaging (3.6) is the most natural (wrt. semantics of the abstract model) approach to define transitions between clusters of states. Additionally, the transition probability function $\mathbf{\Pi}_{out}$ has another property, namely that, for each cluster ρ,

$$\begin{aligned} \sum_{\sigma \in \Phi} \mathbf{\Pi}_{out}(\rho, \sigma) &= \sum_{\sigma \in \Phi} \frac{1}{|\rho|} \sum_{r \in \rho} \sum_{s \in \sigma} P(r, s) = \frac{1}{|\rho|} \sum_{r \in \rho} \sum_{\sigma \in \Phi} \sum_{s \in \sigma} P(r, s) \\ &= \frac{1}{|\rho|} \sum_{r \in \rho} \sum_{s \in S} P(r, s) = \frac{1}{|\rho|} \sum_{r \in \rho} 1 = 1, \end{aligned}$$

i.e. matrix $\mathbf{\Pi}_{out}$ is *stochastic*, unlike general abstract matrices $\mathbf{\Pi}$. Since $\boldsymbol{p}_0$ is always a stochastic vector, then so is $\boldsymbol{\pi}_0$, by definition. This implies that all vectors $\boldsymbol{\pi}_k$ and therefore all $\tilde{\boldsymbol{p}}_k$ are stochastic as well. So, the abstract chain $(\Phi, \boldsymbol{\pi}_0, \mathbf{\Pi}_{out})$ is actually a Markov chain. This subtle difference has two benefits. First, from a technical standpoint, this leads to a slightly better approximation compared to the incoming-based scheme (3.5), as will be shown later. Second, preserving the stochasticity of $\tilde{\boldsymbol{p}}_k$ is the key to enable a transient analysis of continuous-time Markov chains (beyond the scope of this thesis).

Median-based state-space aggregation

This scheme is defined as follows:

$$\mathbf{\Pi}_{med}(\rho, \sigma) = \frac{|\sigma|}{|\rho|} \operatorname*{med}_{s \in \sigma} \left\{ \sum_{r \in \rho} P(r, s) \right\}. \tag{3.7}$$

The median-based scheme was derived to minimize the error bound (see Section 3.2) on the approximation error. As a consequence, in the short run, using median-based scheme proves to be more accurate as compared to the previous two approaches. However, since it was designed to minimize error bound and not the actual error, and since the matrix $\mathbf{\Pi}_{med}$ is not necessarily stochastic, in the long run, this scheme results in very poor accuracy. Empirical evaluation of all three proposed schemes is included in Section 3.3.

3.2 Approximation Error

Let $\boldsymbol{e}_k \coloneqq \tilde{\boldsymbol{p}}_k - \boldsymbol{p}_k$ denote an error vector associated with the approximation $\tilde{\boldsymbol{p}}_k$ of $\boldsymbol{p}_k$. When carrying out the transient analysis of the Markov chain, we cannot compute this vector directly since we do not know the exact probabilities $\boldsymbol{p}_k$. However, we can formally bound the L_1-norm of this vector: this norm is often used with stochastic vectors and is efficiently computable from the structure of $(\Phi, \boldsymbol{\pi}_0, \mathbf{\Pi})$. In [1], authors introduce the quantity

$$\epsilon(\rho, \sigma) \coloneqq \max_{s \in \sigma} \left| \mathbf{\Pi}(\rho, \sigma) - \frac{|\sigma|}{|\rho|} \sum_{r \in \rho} \boldsymbol{P}(r, s) \right| \tag{3.8}$$

and denote $\epsilon(\rho) \coloneqq \sum_{\sigma \in \Phi} \epsilon(\rho, \sigma)$. Then for the L_1-norm of the error vector at time $k > 0$ it holds that

$$\|\boldsymbol{e}_k\|_1 \leq \|\boldsymbol{e}_{k-1}\|_1 + \sum_{\rho \in \Phi} \boldsymbol{\pi}_{k-1}(\rho) \cdot \epsilon(\rho), \tag{3.9}$$

where

$$\|\boldsymbol{e}_0\|_1 = \sum_{s \in S} |\boldsymbol{p}_0(s) - \tilde{\boldsymbol{p}}_0(s)|. \tag{3.10}$$

However, in [6] we derive a stronger bound. Namely, for each pair ρ, σ of clusters, define the quantity

$$\tau(\rho, \sigma) \coloneqq \sum_{s \in \sigma} \left| \frac{\mathbf{\Pi}(\rho, \sigma)}{|\sigma|} - \frac{1}{|\rho|} \sum_{r \in \rho} P(r, s) \right|, \tag{3.11}$$

and denote $\tau(\rho) \coloneqq \sum_{\sigma \in \Phi} \tau(\rho, \sigma)$. Then the L_1-norm of the error vector $\boldsymbol{e}_k$ is estimated similarly to (3.9):

$$\|\boldsymbol{e}_k\|_1 \leq \|\boldsymbol{e}_{k-1}\|_1 + \sum_{\rho \in \Phi} \boldsymbol{\pi}_{k-1}(\rho) \cdot \tau(\rho). \tag{3.12}$$

One can show that $\tau(\rho, \sigma) \leq \epsilon(\rho, \sigma)$ and therefore τ-terms in (3.12) provide a better error bound than that in (3.9) based on ϵ-terms. Furthermore, the new bound (3.12) is derived independently on the form of abstract transition probability matrix $\mathbf{\Pi}$, and therefore we can use (3.12) to gauge the precision of aggregation schemes proposed in Section 3.1 and pick those that demonstrate the best behavior.

Now it should be clear how aggregation helps to mitigate the state space explosion problem. If we select a partition Φ such that $|\Phi| \ll |S|$, working with the abstract chain

$(\Phi, \boldsymbol{\pi}_0, \boldsymbol{\Pi})$ and performing vector-matrix multiplications (3.3) allows to approximate $\boldsymbol{p}_k$ using $\boldsymbol{\pi}_k$, and to reduce the computational demands when compared to performing the discrete steps in (3.1) with the concrete model. The computational overhead associated with estimating bound on the approximation error using (3.12) is almost negligible since this update represents a simple scalar multiplication of two vectors in the aggregated (i.e. with the reduced state space) setting. Another notable observation is that when we perform discrete steps in the aggregated setting, the probability distribution shifts, so adapting the clustering of the state space can reduce the error: *an adaptive state-space aggregation* is therefore a method of using different clusterings sequentially in time, where the quality of each clustering is quantified using (3.12).

3.3 Experimental Evaluation of Approximate Methods

3.3.1 Precision of Aggregation Schemes

Table 3.1 summarizes the experimental evaluation of the three proposed aggregation schemes on a simple real-world case study (uniformized Lotka-Volterra model [25], 160k states). In this experiment, we explore the precision of all three aggregation schemes – median-based (Med), as in (3.7), the one based on incoming averaging (In), as in (3.5) and the one based on outgoing (Out) averaging, as in (3.6). We carry out the transient analysis of the chain for a short ($k = 1$) and a long ($k = 100$) time horizon; for each strategy, we report both empirical error $\|\tilde{\boldsymbol{p}}_k - \boldsymbol{p}_k\|_1$ **(e)** and its theoretical upper bound **(t)** computed using (3.12). Also, for $k = 100$, we allowed reclusterings at fixed time steps. In the case of average incoming probabilities, we also compute theoretical bound using the ϵ-factors, as in (3.9) (In'), and using the new bound (3.12) based on τ-factors (In).

		Med	In'	In	Out
$k = 1$	**(e)**	**1.93E-23**	**2.07E-23**	**2.07E-23**	2.51E-23
	(t)	**9.77E-24**	1.31E-20	**1.28E-23**	1.65E-23
$k = 100$	**(e)**	3.50E-4	2.01E-17	2.01E-17	**2.85E-20**
	(t)	3.50E-4	7.17E-14	2.81E-17	**2.04E-19**

Table 3.1: Precision of different aggregation schemes.

First, from all experiments we confirm that In $\ll$ In', i.e. the newly derived error bound (3.12) that utilizes τ-terms provides several orders of magnitude better bounds than that based on ϵ-terms (3.9). Second, experiment $k = 1$ shows us that median-based aggregation exhibits the best one-step behavior, followed by that based on incoming probability, followed by that based on outgoing probability. Finally, in case $k = 100$, we see that median-based scheme results in very poor accuracy compared to those based on incoming and outgoing probabilities, with the latter having an additional edge on the former of a couple of orders of magnitude.

3.3.2 Speedup of Approximate Methods

In this set of experiments, we investigate the speedup (acceleration wrt. the exact computation) of approximate methods. We compare adaptive state-space aggregation (utilizing outgoing averaging, the one that proved to be the most precise), described in this chapter,

with the state space truncation. Recall from Chapter 1 that truncation is another approach how to deal with the state space explosion problem, where, instead of aggregating multiple states into clusters, we (dynamically) ignore a large portion of the state space, thus retaining information about the probability distribution only in the selected subset of states. Table 3.2 presents the results for three different case studies, where, for each method and for five different levels of precision, we report acceleration wrt. the exact computation. We can see that in all cases the state-space aggregation performs considerably better than truncation, demonstrating a two- to fivefold acceleration.

Precision	(a)		(b)		(c)	
	Tru	Agg	Tru	Agg	Tru	Agg
1E-7	6.66	**9.10**	7.02	**21.71**	4.81	**11.54**
1E-6	7.55	**10.56**	9.16	**28.40**	5.76	**14.97**
1E-5	8.28	**11.21**	9.86	**35.78**	6.54	**16.75**
1E-4	8.86	**16.55**	10.97	**40.16**	7.37	**18.74**
1E-3	9.78	**17.96**	11.81	**51.00**	8.46	**24.13**

Table 3.2: Speedup (acceleration wrt. concrete computation for guaranteeing a given precision) comparison. Models: (a) workstation cluster [26], dimension of the state space: 1M states, time horizon: 10804; (b) uniformized prokaryotic gene expression [31], dimension of the state space: 1.2M states, time horizon: 10000; (c) uniformized two-component signalling pathway [15], population bounds [14,46]; property of interest is bounded reachability with $k = 10000$; dimension of the state space after PCTL driven transformation: 1M states.

Chapter 4

Synthesis of Probabilistic Programs

4.1 Families of Markov Chains

Having explored techniques for *analyzing* an individual Markov chain and deciding whether it exhibits desirable properties, the next step is to be able to *synthesize* a chain possessing these properties. In other words, having a family of MCs, how can one efficiently identify a chain satisfying a given specification? Synthesizing a Markov chain having a fixed topology but unknown transition probabilities have been addressed by techniques for parameter synthesis [12, 39] or model repair [8, 38]. In this work, we want to focus on families of Markov chains having different topologies of the state space and, as a result, different sets of reachable states. The following definition introduces a construct for describing such families.

Definition 9. [13] A *family of MCs* is a tuple $\mathcal{D} = (S, s_{init}, K, \mathcal{B})$ with S and s_{init} as before, K is a finite set of parameters with domains $T_k \subseteq S$ for each $k \in K$, and $\mathcal{B} : S \to Distr(K)$ is a family of transition probability functions.

Function $\mathcal{B}$ of a family $\mathcal{D}$ of MCs maps each state to a distribution over parameters K. In the context of the synthesis of probabilistic models, these parameters represent unknown options or features of a system under design. If we assign specific values (states) to each of the parameters, function $\mathcal{B}$ will map state to a distribution over states, yielding a concrete Markov chain. This notion is captured in the following definition.

Definition 10. A *realization* of a family $\mathcal{D} = (S, s_{init}, K, \mathcal{B})$ of MCs is a function $r : K \to S$ s.t. $\forall k \in K : r(k) \in T_k$. We say that realization r induces MC $\mathcal{D}_r = (S, s_{init}, \mathcal{B}_r)$ iff $\mathcal{B}_r(s, s') = \sum_{k \in K, r(k)=s'} \mathcal{B}(s)(k)$ for any two states $s, s' \in S$. A set of all realizations of $\mathcal{D}$ is denoted as $\mathcal{R}^{\mathcal{D}}$.

Notice that the set $\prod_{k \in K} T_k$ of possible parameter combinations and the set $\mathcal{R}^{\mathcal{D}}$ of realizations of $\mathcal{D}$ are semantically equivalent, i.e. $|\mathcal{R}^{\mathcal{D}}| = |\prod_{k \in K} T_k| = \prod_{k \in K} |T_k| =: |\mathcal{D}|$ is the size of the family. Since parameter domains T_k are finite, then so is the family, i.e. $|\mathcal{D}| < \infty$. The number $|\mathcal{D}|$ of family members is exponential in $|K|$, giving rise to the first state space explosion problem (another state explosion affects the number $|S|$ of states of individual family members). To simplify notation, we might sometimes use the same symbol $\mathcal{D}$ do denote the set $\{\mathcal{D}_r\}_{r \in \mathcal{R}^{\mathcal{D}}}$ of induced MCs, i.e. the set of family members.

Example 8. Assume a family $\mathcal{D} = (S, s_0, K, \mathcal{B})$ of MCs with the state space $S = \{s_0, s_1, s_2, s_3\}$ and a set $K = \{X, Y, k_0, k_3\}$ of parameters with domains $T_X = T_Y = \{s_1, s_2\}$, $T_{k_0} = \{s_0\}$ and $T_{k_3} = \{s_3\}$, where the family $\mathcal{B}$ of transition probability functions is defined as follows:

$$\begin{aligned}
\mathcal{B}(s_0) &= \frac{1}{2} : X + \frac{1}{2} : Y, \\
\mathcal{B}(s_1) &= \frac{1}{2} : k_0 + \frac{1}{2} : k_3, \\
\mathcal{B}(s_2) &= 1 : X, \\
\mathcal{B}(s_3) &= 1 : k_3.
\end{aligned}$$

We have a total of $|T_X| \cdot |T_X| \cdot |T_{k_0}| \cdot |T_{k_3}| = 2 \cdot 2 \cdot 1 \cdot 1 = 4$ realizations. All four members of the family $\mathcal{D}$ are depicted in Figure 4.1. Notice that these chains have a different topology of the underlying state space, which even results in different sets of reachable states (in Figure 4.1, unreachable states are grayed out). Also, notice that the chain $\mathcal{D}_{r_1}$ is the same chain D we analyzed previously in Chapter 2, see Figure 2.1.

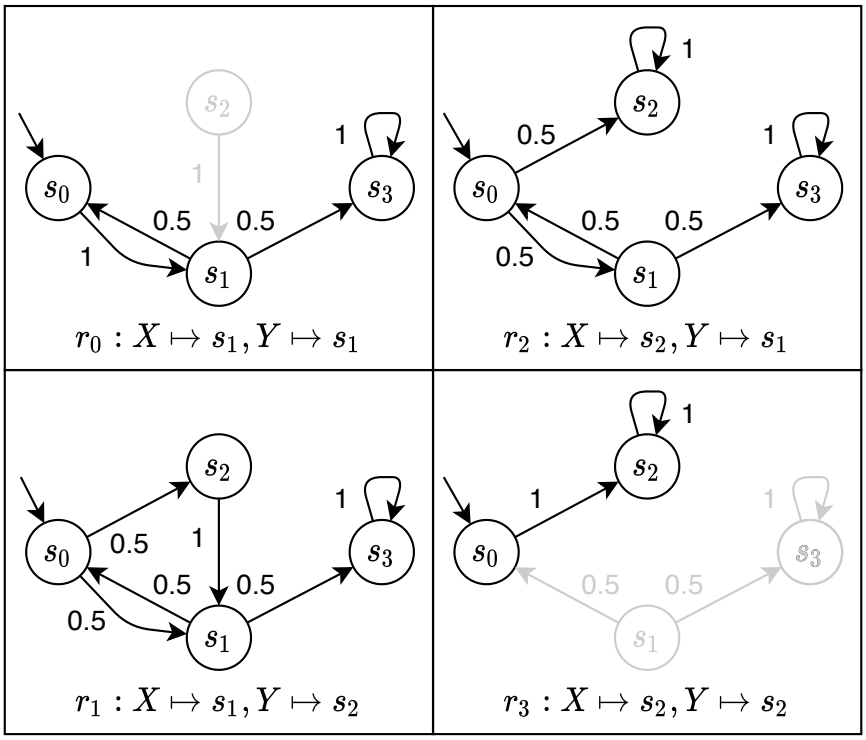


Figure 4.1: A family $\mathcal{D}$ of four Markov chains. For each chain, unreachable states are grayed out.

Assume $\mathcal{D}$ is the family of Markov chains and $\varphi \equiv \mathbb{P}_{\bowtie\lambda}[\mathrm{F}\ T]$ is a reachability specification. We distinguish between the following synthesis problems:

- *feasibility synthesis*: identify $r \in \mathcal{R}^{\mathcal{D}}$ such that $\mathcal{D}_r \models \varphi$,

- *threshold synthesis*: partition the set $\mathcal{R}^{\mathcal{D}}$ of realizations into sets $\mathcal{R}_T$ and $\mathcal{R}_F$ such that $\forall r \in \mathcal{R}_T : \mathcal{D}_r \models \varphi$ and $\forall r \in \mathcal{R}_F : \mathcal{D}_r \not\models \varphi$,
- *maximum synthesis*: identify $r^* \in \mathcal{R}^{\mathcal{D}}$ such that $r^* \in \arg\max_{r \in \mathcal{R}^{\mathcal{D}}} \mathbb{P}[\mathcal{D}_r \models \mathrm{F}\ T]$.

In the following, we focus on the simplest problem – feasibility. In our case, when the state space S and the set K of parameters are finite, the feasibility problem is decidable and, to be more specific, $\mathcal{NP}$-hard [12]. A possible solution to the problem – the so-called *one-by-one approach* [17] – is to enumerate through each realization $r \in \mathcal{R}^{\mathcal{D}}$ and check whether $\mathcal{D}_r \models \varphi$. As was argued previously, the double state-space explosion renders this approach unusable for large families, necessitating the usage of advanced techniques. These techniques usually attempt either to analyze whole subfamilies at once or generalize analysis results to subfamilies, thus accelerating the pruning of the state space of candidate solutions. Definition 11 below formalizes the concept of subfamily by restricting a family to a subset of realizations. Additionally, Definition 12 gives us one way to define these subsets of realizations.

Definition 11. Let $\mathcal{D} = (S, s_{init}, K, \mathcal{B})$ be a family of MCs and $\mathcal{R} \subseteq \mathcal{R}^{\mathcal{D}}$ a subset of realizations. A *subfamily* of $\mathcal{D}$ wrt. $\mathcal{R}$ is a family $\mathcal{D}[\mathcal{R}] = (S, s_{init}, K, \mathcal{B})$ with $\mathcal{R}^{\mathcal{D}[\mathcal{R}]} = \mathcal{R}$.

Definition 12. Let $\mathcal{D} = (S, s_{init}, K, \mathcal{B})$ be a family of MCs and $r \in \mathcal{R}^{\mathcal{D}}$ be any realization. A *generalization* of r wrt. a subset $\overline{K} \subseteq K$ of parameters is a set $r{\uparrow}\overline{K} = \{r' \in \mathcal{R}^{\mathcal{D}} \mid \forall k \in \overline{K} : r(k) = r'(k)\}$.

A set $r{\uparrow}\overline{K}$ describes a maximum set of realizations that share with realization r the same assignment of parameters in $\overline{K}$: values of parameters $K \setminus \overline{K}$ are irrelevant.

Example 9. Assume a family $\mathcal{D}$ of MCs from Example 8 and realization r_1 (see Figure 4.1). A generalization of r_1 wrt. set $\{\mathrm{Y}\}$ of parameters is the set $r_1{\uparrow}\{Y\} = \{r_1, r_3\}$ of realizations that assign a value $r_1(Y) = 2$ to parameter Y. Furthermore, we can define a subfamily $\mathcal{D}[r_1{\uparrow}\{Y\}]$ of $\mathcal{D}$ as a family which considers only realizations r_1 and r_3.

4.2 Counterexample-Guided Inductive Synthesis

Let $\mathcal{D} = (S, s_{init}, K, \mathcal{B})$ be a family of MCs and φ be a property of interest. One approach for synthesizing a Markov chain satisfying φ is the *counterexample-guided inductive synthesis (CEGIS)* [13] and its main idea goes as follows. First, we assume a set $\mathcal{R} = \mathcal{R}^{\mathcal{D}}$ of candidate solutions. We pick any realization $r \in \mathcal{R}$ and construct the corresponding instance $\mathcal{D}_r$. We then check whether $\mathcal{D}_r \models \varphi$ and, if so, the synthesis is complete and we return realization r to the user. Otherwise, we compute a critical set C for $\mathcal{D}_r$ and φ, according to Definition 4. Recall that a critical set C (usually) represents only a fraction of the original state space S. Hence, it is not improbable, that, for some parameter $k \in K$, the value of $r(k)$ is never used in the construction of the critical subsystem $\mathcal{D}_r{\downarrow}C$. Therefore, such parameter k is *irrelevant*. Naturally, there might be more of such parameters. Definition 13 below formalizes this notion.

Definition 13. Let $\mathcal{D} = (S, s_{init}, K, \mathcal{B})$ be a family of MCs. For a subset $C \subseteq S$ of critical states, a set of *relevant parameters* (conflict) is a set $\overline{K} = \bigcup_{s \in C} \mathrm{supp}(\mathcal{B}(s))$.

Having obtained a set $\overline{K}$ of relevant parameters, we construct a generalization $r{\uparrow}\overline{K}$ of realization r wrt. $\overline{K}$, according to Definition 12. Recall that $r{\uparrow}\overline{K}$ is the set of realizations that differ from unsatisfying realization r only in the values of irrelevant parameters.

Therefore, since $\mathcal{D}_r \not\models \varphi$, then it must hold that $\forall r' \in r{\uparrow}\overline{K} : \mathcal{D}_{r'} \not\models \varphi$. In other words, we reject r, construct its critical subsystem, identify which parameters are relevant for this counterexample, and infer that any other realization r', which differs from r in the values of irrelevant parameters only, must violate φ as well. The profound effect of this generalization is that, by checking one candidate solution, we reject the whole subfamily $\mathcal{D}[r{\uparrow}\overline{K}]$ of Markov chains, which allows us to prune the state space of solutions more efficiently. We then subtract $r{\uparrow}\overline{K}$ from the set $\mathcal{R}$ of candidate solutions and repeat the procedure described above until either a satisfying realization is found, or the whole state space is exhausted, which indicates that no feasible solution exists. The approach is summarized in Algorithm 3.

Algorithm 3: Counterexample-guided inductive synthesis.

Input : A family $\mathcal{D} = (S, s_{init}, K, \mathcal{B})$ of MCs, a reachability property φ.
Output: Realization $r \in \mathcal{R}^{\mathcal{D}}$ s.t. $\mathcal{D}_r \models \varphi$, or UNSAT if no such realization exists.

Function `CEGIS`$(\mathcal{D}, \varphi)$**:**
  $\mathcal{R} \leftarrow \mathcal{R}^{\mathcal{D}}$
  while $\mathcal{R} \neq \emptyset$ **do**
    $r \leftarrow \text{any}(\mathcal{R})$
    if $\mathcal{D}_r \models \varphi$ **then**
      return r
    end if
    $C \leftarrow \text{criticalSubsystem}(\mathcal{D}_r, \varphi)$
    $\overline{K} \leftarrow \text{relevantParameters}(\mathcal{D}, C)$ `// using Def. 13`
    $\mathcal{R} \leftarrow \mathcal{R} \backslash (r{\uparrow}\overline{K})$
  end while
  return UNSAT

4.3 Counterexample-Guided Abstraction Refinement

Counterexample-guided abstraction refinement (CEGAR) [14] is another synthesis method that takes approach opposite to that of CEGIS. Namely, instead of model checking individual realizations and then generalizing them to subfamilies of unsatisfying chains, we instead first assume a stochastic process in which all realizations are possible *at once*. In particular, whenever the process visits a state $s \in S$, it has a nondeterministic choice of realization $r \in \mathcal{R}^{\mathcal{D}}$ (assignment of parameters K), which in turn yields a specific distribution $\mathcal{B}_r(s)$ over successor states. We recognize this informal description as a Markov decision process, which we will call *a quotient MDP*.

Definition 14. [14] Let $\mathcal{D} = (S, s_{init}, K, \mathcal{B})$ be a family of MCs. A *quotient MDP* of $\mathcal{D}$ is an MDP $M^{\mathcal{D}} = (S, s_{init}, \mathcal{R}^{\mathcal{D}}, \mathcal{P})$, where $\mathcal{P}(\cdot)(r) \equiv \mathcal{B}_r$. For $\mathcal{R} \subseteq \mathcal{R}^{\mathcal{D}}$, a *restriction* of $M^{\mathcal{D}}$ wrt. $\mathcal{R} \subseteq \mathcal{R}^{\mathcal{D}}$ is an MDP $M^{\mathcal{D}}[\mathcal{R}] \coloneqq M^{\mathcal{D}[\mathcal{R}]}$.

Quotient MDP $M^{\mathcal{D}}$ is capable of simulating the behavior of each family member $\mathcal{D}_r$ and can even 'switch' to that of another member $\mathcal{D}_{r'}$ mid-execution. In other words, if the path $s_0 s_1 s_2 \dots$ is executable in some member $\mathcal{D}_r$, then it is executable in $M^{\mathcal{D}}$ as $s_0 \xrightarrow{r} s_1 \xrightarrow{r} s_2 \xrightarrow{r} \dots$. On the other hand, if a path π is executable in $M^{\mathcal{D}}$, then it might be a path that is impossible in neither of the members of $\mathcal{D}$. Quotient MDP essentially

overapproximates the behavior of each of the members of family $\mathcal{D}$. A restriction of a quotient MDP wrt. $\mathcal{R} \subseteq \mathcal{R}^{\mathcal{D}}$ is this same MDP where we take into account only transitions associated with $\mathcal{R}$. To ensure that the execution of a quotient MDP always picks the same realization, we can make use of consistent schedulers.

Definition 15. Let $\mathcal{D} = (S, s_{init}, K, \mathcal{B})$ be a family of MCs and let $M^{\mathcal{D}} = (S, s_{init}, \mathcal{R}^{\mathcal{D}}, \mathcal{P})$ be a quotient MDP of $\mathcal{D}$. For $r \in \mathcal{R}^{\mathcal{D}}$, a (memoryless) scheduler $\sigma_r \in \Sigma^{M^{\mathcal{D}}}$ is called *r-consistent* iff $\forall s \in S : \sigma(s) = r$. A scheduler is called *consistent* iff it is consistent for some $r \in \mathcal{R}^{\mathcal{D}}$.

Proposition 3. Let $M^{\mathcal{D}}$ be a quotient MDP of the family $\mathcal{D}$ of MCs and let $\sigma_r \in \Sigma^{M^{\mathcal{D}}}$ be an r-consistent scheduler. Then $M^{\mathcal{D}}_{\sigma_r} \equiv \mathcal{D}_r$.

Proposition 3 essentially states that a consistent scheduler for a quotient MDP yields a valid member of the family. The proof follows directly from Definition 14. Returning to the problem of probabilistic synthesis, having a quotient MDP $M^{\mathcal{D}}$ that overapproximates the behavior of each Markov chain in a family $\mathcal{D}$, model checking this MDP against a specification φ can yield interesting results. In particular, assume that $\varphi \equiv \mathbb{P}_{\leq\lambda}[\mathrm{F}\ T]$ is a safety property. Let us compute minimizing and maximizing schedulers $\sigma_{\min}$ and $\sigma_{\max}$, respectively, as well as the corresponding lower ($\boldsymbol{x}_{\min}$) and upper ($\boldsymbol{x}_{\max}$) bounds on the reachability probability. If $\boldsymbol{x}_{\min}(s_{init}) > \lambda$, we know for sure that, for each realization $r \in \mathcal{R}^{\mathcal{D}}$, $\mathcal{D}_r \not\models \varphi$, that is, the synthesis problem has no feasible solutions. On the other hand, if $\boldsymbol{x}_{\max}(s_{init}) \leq \lambda$, then each family member $\mathcal{D}_r$ satisfies φ and we can return any realization to the user as a solution to the synthesis problem. Finally, if $\boldsymbol{x}_{\min}(s_{init}) \leq \lambda < \boldsymbol{x}_{\max}(s_{init})$, then we cannot decide anything unless $\sigma_{\min}$ is a consistent scheduler, in which case $M^{\mathcal{D}}_{\sigma_{\min}}$ represents a valid family member with $\boldsymbol{x}_{\min}(s_{init}) \leq \lambda$, i.e. it is a solution to the synthesis problem. Otherwise, if $\sigma_{\min}$ is not consistent, the problem remains undecided since the abstraction $M^{\mathcal{D}}$ is too coarse.

Borrowing from the philosophy of abstraction refinement, we can try to fix such undecidable case by splitting the family of Markov chains into two subfamilies, and then attempt to analyze each of them separately using the procedure described above. If these subfamilies still represent undecidable cases, we continue to refine them into smaller and smaller subfamilies until either we find a feasible solution, or we reject all family members. The procedure is guaranteed to terminate for two reasons. First, the number of family members is finite and we cannot refine subfamilies indefinitely. Second, in the extreme case, where after numerous refinements we have obtained a subfamily with exactly one realization, the corresponding quotient MDP is essentially an MC, for which $\boldsymbol{x}_{\min} = \boldsymbol{x}_{\max}$ and the case is necessarily conclusive. The described method is summarized in Algorithm 4. Liveness properties are handled analogously.

Refinement of families is carried out through the process of *splitting* (see Line 18 of Algorithm 4), which involves picking a suitable parameter $k \in K$ having possible values T_k and a predicate P over S, and then constructing two subfamilies: a maximum subfamily $\mathcal{R}_\top \subset \mathcal{R}$, for which $P(r(k))$ holds for any $r \in \mathcal{R}_\top$, and $\mathcal{R}_\bot = \mathcal{R} \setminus \mathcal{R}_\top$. This results in a pair $\mathcal{R}_\top, \mathcal{R}_\bot$ of subfamilies of chains, for which the described procedure is called recursively until MDP model checking call yields conclusive results.

CEGAR represents an approach diametrically opposite to that of CEGIS, and, therefore, for different models and different specifications, these methods tend to behave differently. Sometimes the shape of the family or the topology of the state space allow construction of small counterexamples and CEGIS can prune the state space completely by analyzing

Algorithm 4: Counterexample-guided abstraction refinement.

Input : A family $\mathcal{D} = (S, s_{init}, K, \mathcal{B})$ MCs, a safety property $\varphi \equiv \mathbb{P}_{\leq\lambda}[\mathrm{F}\ T]$.
Output: Realisation $r \in \mathcal{R}^{\mathcal{D}}$ s.t. $\mathcal{D}_r \models \varphi$, or UNSAT if no such realization exists.

Function `CEGAR`$(\mathcal{D}, \varphi)$**:**
  $M^{\mathcal{D}} \leftarrow$ buildQuotientMDP$(\mathcal{D})$ `// using Def. 14`
  $\mathfrak{R} \leftarrow \{\mathcal{R}^{\mathcal{D}}\}$
  while $\mathfrak{R} \neq \emptyset$ **do**
    $\mathcal{R} \leftarrow$ any$(\mathfrak{R})$
    $\mathfrak{R} \leftarrow \mathfrak{R} \setminus \{\mathcal{R}\}$
    $M \leftarrow M^{\mathcal{D}}[\mathcal{R}]$ `// using Def. 14`
    $(\boldsymbol{x}_{\min}, \sigma_{\min}, \boldsymbol{x}_{\max}, \sigma_{\max}) \leftarrow$ reachabilityMDP(M, T) `// using e.g. Alg. 2`
    if $\boldsymbol{x}_{\min}(s_{init}) > \lambda$ **then**
      continue
    end if
    if $\boldsymbol{x}_{\max}(s_{init}) \leq \lambda$ **then**
      return any$(\mathcal{R})$
    end if
    if $\sigma_{\min}$ *is* r*-consistent for some* $r \in \mathcal{R}$ **then**
      return r
    end if
    $(\mathcal{R}_\top, \mathcal{R}_\bot) \leftarrow$ split$(\mathcal{R})$
    $\mathfrak{R} \leftarrow \mathfrak{R} \cup \{\mathcal{R}_\bot, \mathcal{R}_\top\}$
  end while
  return UNSAT

only a few realizations. Or it might be the case that CEGIS is unable to generalize any of the parameters (yielding $\overline{K} = K$ as a conflict set) and therefore must analyze each realization individually. Likewise, the form of the quotient MDP might yield very tight bounds on the reachability probability, thus allowing to synthesize a solution after only a couple of refinements, or we might be 'unlucky' and have to refine subfamilies up to the level of individual MCs to obtain conclusive results.

The behavior of the methods also depends on the bound λ of the reachability property. If the bound is too low or too high (wrt. the given model and a property), then such cases are usually easily solvable. In particular, when analyzing a safety property, then for low values of λ CEGIS is capable of constructing extremely small counterexamples (we need only a handful of paths that end up in T), thus pruning the state space faster; on the other hand, if the bound λ is too high, it will most likely imply that a majority of realizations are satisfiable, thus increasing the chance that CEGIS picks one for model checking. Likewise, very low or very high bound λ increases the chance that CEGAR, even after obtaining conservative bounds on the reachability probability, is still able to decide whether the subfamily is definitely satisfiable or definitely not. For this reason, when we will later, in Chapter 6, evaluate the performance of the synthesis methods, we will usually consider formulae with bound λ in the vicinity of the feasibility threshold λ_f, for which the synthesis problem changes from 'feasible' to 'unfeasible'.

The duality of these bottom-up and top-down approaches suggests that there might be a middle ground: a method capable of combining the power of all-at-once analysis with

the precision of MC model checking. Designing such a method represents a formidable challenge that we will tackle in Chapter 5. Before that, however, let us make one final step and describe the connection between Markov chains and probabilistic programs.

4.4 Probabilistic Programs

When modeling a probabilistic system, we may describe it as a Markov chain, as per Definition 2, but, in practice, the state space explosion problem renders such an approach unjustifiable. Therefore, we usually describe the system using a high-level programming language, such as PRISM [33], PIOA [43], JANI [11] or MODEST [10]. Probabilistic model checkers then parse this high-level description $\mathcal{P}$ and construct the corresponding Markov chain $[\![\mathcal{P}]\!]$, which is then used as an input for various model checking algorithms. A *sketch* [41, 5, 13] represents a high-level yet incomplete description of a system under development. A sketch usually describes the fixed behavior of the modeled system but leaves so-called 'holes' – parts of a program that must be filled in a way so that the resulting description $\mathcal{P}$ satisfies a given specification. In terms of constructs we are already familiar with, a sketch is a high-level analog of a family of Markov chains, where holes correspond to parameters and filling all holes yields a concrete Markov chain; program-level synthesis then translates to recognizing how to fill these holes (complete the description) in order to satisfy given constraints. In this section, we give a very brief overview of the PRISM sketching language proposed in [13] and discuss a program-level analog of the previously described CEGIS method.

4.4.1 PRISM Sketch Language

A PRISM program [33] consists of one or more modules that interact with each other. Without loss of generality, we will consider programs with a single module (every program can be 'flattened' into this form). A module contains a definition of (bounded) variables (as well as their initial values) spanning the state space of the system and a set of commands that describe the transitions between these states. These commands are of the form

$$\texttt{guard} \to p_1 : \texttt{update}_1 + \ldots \cdots + p_n : \texttt{update}_n$$

The guard is a Boolean expression over the variables declared in the module. The command itself represents a probability distribution over a set of updates, where each update encodes a change in the values of variables. Essentially, the guard identifies states for which this command is applicable, and updates describe successor states as well as the probability distribution over these successors. Overlapping of guards yields nondeterminism and is thus disallowed. A (reachable) state which is not covered by a command guard represents a deadlock – in terms of model-based semantics such a state is considered absorbing. For a program description $\mathcal{P}$, *underlying MC* $[\![\mathcal{P}]\!]$ is a Markov chain that executes semantics of $\mathcal{P}$ on a state level. We say that a program $\mathcal{P}$ satisfies specification Φ iff it is satisfied by its underlying MC.

Example 10. Assume the following PRISM program $\mathcal{P}$:

```
int s:  [0..4] init 2;
s < 2 → 0.75 : (s' = max(s-1,0)) + 0.25 : (s' = s+1);
s = 2 → 0.5 : (s' = s-1) + 0.5 : (s' = s+1);
s > 2 → 0.25 : (s' = s-1) + 0.75 : (s' = min(s+1,4));
```

The corresponding underlying MC $[\![\mathcal{P}]\!]$ is depicted in Figure 4.2. A property of interest might be e.g. $\mathcal{P} \models \mathbb{P}_{\leq 0.6}[\mathrm{F}\ (s \geq 3)]$.

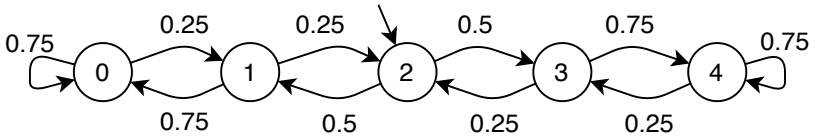


Figure 4.2: An underlying MC $[\![\mathcal{P}]\!]$ of a probabilistic program $\mathcal{P}$ from Example 10.

A *sketch* [13] is a program that contains holes: open parts of the program and can be filled with one of the finitely many options. These are declared as

$$\texttt{hole}\ h\ \texttt{either}\ \{\texttt{expr}_1, \cdots, \texttt{expr}_k\}$$

where h is a hole identifier and $\texttt{expr}_i$ is an expression over the program variables. A hole can be used in variable declarations and commands, both as part of a guard or an update expression. A program sketch represents a description of the system with an a priori known general structure – number of variables, a form of commands, etc. – but with some concrete details left out. Instantiating all holes yields a specific program and the goal of the synthesizer is to decide how to replace all holes with their options in order to satisfy a given specification. This synthesis is usually carried out on a state level using methods described earlier in this chapter. PRISM sketching language also allows some additional functionality: assigning costs to option holes, introducing constraints on hole values, etc. For a comprehensive description, please refer to [13].

Example 11. Assume the following PRISM sketch:

```
hole X either {0,1};
hole Y either {1,2,3};
int s:  [0..4] init X+1;
s < Y → 0.75 : (s' = max(s-X,0)) + 0.25 : (s' = s+X);
s = Y → 0.5 : (s' = s-1) + 0.5 : (s' = s+1);
s > Y → 0.25 : (s' = s-1) + 0.75 : (s' = min(s+1,4));
```

Instantiating $X = 1$ and $Y = 2$ yields program discussed in Example 10.

4.4.2 Program-Level CEGIS

Authors of [13] propose a lifting of the CEGIS method to a program level in order to improve its scalability. In the remainder of this section, we describe only its basic idea: we will not need this lifting when designing the integrated synthesis technique in Chapter 5, but we will consider this variant of CEGIS during the experimental evaluation of the synthesis techniques in Chapter 6, because program-level CEGIS, (alongside CEGAR), represents a state-of-the-art approach for the synthesis of probabilistic programs.

In its core, a program-level CEGIS behaves exactly like its state-level counterpart. Having a program sketch and a safety property, we execute the CEGIS loop: instantiate all holes, construct the underlying Markov chain and, if the chain does not satisfy the specification, construct a counterexample and reject those realizations (combinations of hole options) that also lead to programs violating the specification. The catch is that the counterexample is constructed on a program level. In this setting, a counterexample represents a subset of critical commands (as opposed to a subset of critical states) of the original program. Keeping only some commands has the following effect: states that are covered by a guard of a critical command have their original transitions, while states that are not covered represent deadlocks (i.e. such states are absorbing). This perfectly mirrors the philosophy of a state-level counterexample: states included in a critical set have their original semantics, while states that are not critical essentially exhibit no behavior. Relevant holes are those used in the critical commands: holes that are never used in this subprogram can be therefore safely generalized. A (minimal) set of critical commands is constructed by trying all possible combinations: we first try to select zero commands, then one command, then two, etc. until the subprogram becomes critical. Liveness properties are handled similarly, although they require some additional manipulations with the model and the property, see [13].

Chapter 5

Novel Integrated Methods for Probabilistic Synthesis

If we want to integrate both inductive and abstraction-refinement approaches into a single versatile technique, we can consider multiple ways. First, we might consider using (exact) results from MC model checking to improve the precision of MDP approximation, thus obtaining decidable subfamilies earlier in the refinement process. Or we could consider analyzing the relation between exact MC model checking results and approximate MDP model checking results in order to search for feasible solutions faster, in the spirit of informed search methods. Another approach would be to use lower and upper estimates on the reachability probability, obtained by model checking a quotient MDP, to somehow assist CEGIS in constructing counterexamples: either by computing them faster or by enabling the construction of smaller critical sets, thus pruning the state space of candidate solution at a higher rate. In this work, we focus on the last approach and present two possible ways how CEGIS can facilitate MDP model checking techniques in order to arrive at better counterexamples. The first method emerged during our work on the semester project and, although it was shown to have no advantages over existing methods, it still represents an adequate synthesis technique offering performance comparable to those of CEGIS or CEGAR. We will discuss the approach in Section 5.1 since it will help to illustrate some key aspects of integrated synthesis. Later, in Section 5.3, we will develop an advanced approach and present the primary contribution of this paper: an integrated technique capable of outperforming state-of-the-art methods.

Since we decided to design a technique aiming at enhancing the counterexample generation, let us begin with a motivational example where CEGIS performs poorly.

Example 12. Consider a rather artificial family $\mathcal{D} = (S, s_{init}, K', \mathcal{B})$ of Markov chains, where $s = \{s_{init}, s_1, s_2, t, f\}$, the parameters are $K' = \{X, Y, T, F\}$ with domains $T_X = \{s_1, s_2\}$, $T_Y = \{t, f\}$, $T_T = \{t\}$, $T_F = \{f\}$, and a family $\mathcal{B}$ of transition probability functions is defined as:

$$
\begin{aligned}
\mathcal{B}(s_{init}) &= 1 : X, \\
\mathcal{B}(s_1) &= 0.6 : T + 0.2 : Y + 0.2 : F, \\
\mathcal{B}(s_2) &= 0.2 : T + 0.2 : Y + 0.6 : F, \\
\mathcal{B}(t) &= 1 : T, \\
\mathcal{B}(f) &= 1 : F.
\end{aligned}
$$

Since T and F can each take only one value, these are not actual parameters, so from now on we will only consider the set $K = \{X, Y\}$ as the set of parameters. There are a total of $|T_X| \times |T_Y| = 4$ family members, all of them depicted in Figure 5.1. For simplicity, we did not write the values of transition probabilities unless it is necessary (keep in mind that probabilities must sum to one). For each member, the unreachable part of the state space is grayed out. Assume a safety property $\varphi \equiv \mathbb{P}_{\leq 0.3}[\mathrm{F}\ \{t\}]$. Notice that only realization r_3 satisfies φ. Let us verbalize how CEGIS and CEGAR would find this solution.

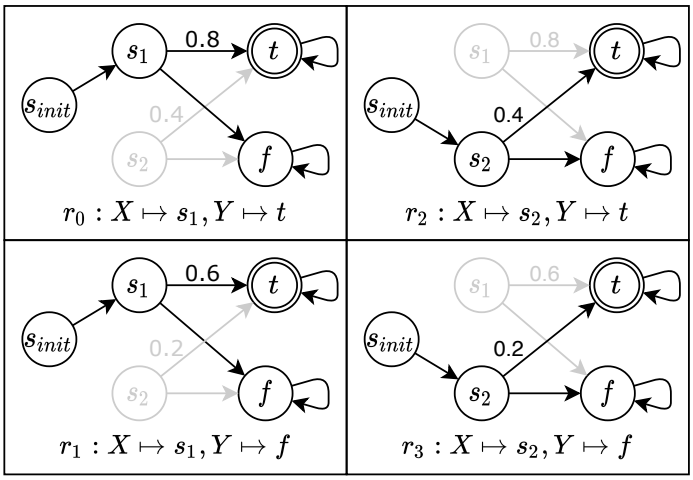


Figure 5.1: A family $\mathcal{D}$ of four Markov chains. For each chain, unreachable states are grayed out.

CEGAR. We begin with a set $\mathcal{R}_1 = \{s_1, s_2\} \times \{t, f\}$ of realizations and construct the corresponding quotient MDP $M^{\mathcal{D}}[\mathcal{R}_1]$ depicted in Figure 5.2. On a diagram, since multiple realizations might yield the same probability distribution, we included these realizations in set braces. Model checking this MDP yields $\boldsymbol{x}_{\min}(s_{init}) = 0.2 \leq 0.3 < 0.8 = \boldsymbol{x}_{\max}(s_{init})$ and, for the sake of argument, assume that the corresponding minimizing scheduler $\sigma_{\min}$ is not consistent (e.g. $\sigma_{\min} : s_{init} \mapsto r_2$ and $s \mapsto r_3$ for $s \neq s_{init}$. We obtain an undecidable case and need to refine our abstraction. Let us pick parameter X and split the set $\mathcal{R}_1$ of realizations into subcases $\mathcal{R}_2 = \{s_1\} \times \{t, f\}$ and $\mathcal{R}_3 = \{s_2\} \times \{t, f\}$. The procedure is repeated for these smaller subfamilies: model checking $M^{\mathcal{D}}[\mathcal{R}_2]$ allows to reject all realizations associated with this subfamily ($\boldsymbol{x}_{\min}(s_{init}) = 0.6 > 0.3$) and model checking $M^{\mathcal{D}}[\mathcal{R}_3]$ again yields undecidable result. Splitting $\mathcal{R}_3$ once more, this time at parameter Y, yields another two subcases: $\mathcal{R}_4 = \{s_2\} \times \{t\}$ and $\mathcal{R}_5 = \{s_2\} \times \{f\}$, each containing a single MC, where only for $M^{\mathcal{D}}[\mathcal{R}_5]$ we obtain $\boldsymbol{x}(s_{init}) = 0.2 \leq 0.3$, from which we conclude that realization r_3 represents the solution to the synthesis problem.

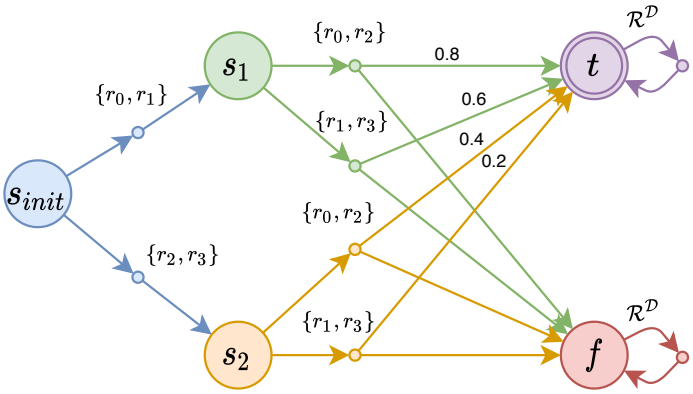


Figure 5.2: A quotient MDP $M^{\mathcal{D}}$ of a family $\mathcal{D}$. Multiple realizations that yield the same probability distribution are included in set braces.

CEGIS. We begin with a set $\{0,1\} \times \{t,f\}$ of realizations. Assume we pick realization $r_0 : [X \mapsto s_1, Y \mapsto t]$. MC model checker gives $\mathcal{D}_{r_0} \not\models \varphi$ since the corresponding reachability probability in the initial state is $0.8 > 0.3$. We now construct a counterexample for $\mathcal{D}_{r_0}$ and φ. The critical subsystem must contain a path $s_{init}s_1t$ having probability $0.8 > 0.3$, therefore, the corresponding critical set will be $C = \{s_{init}, s_1, t\}$. Conflict (a set of relevant parameters, see Definition 13) for C are parameters used in the definition of $\mathcal{B}(s)$ for $s \in C$, i.e. $\overline{K} = \{X, Y\}$. This implies that we reject only a single realization r_0 (none of the parameters were generalized) and continue with a set $(\{s_1, s_2\} \times \{t, f\}) \backslash \{(s_1, t)\}$ of candidate solutions. The same argument applies for any subsequent realization: the constructed counterexamples do not allow any generalization and we will be iterating through all realizations until we are lucky to pick r_3 since it is our only feasible solution.

The key observation from Example 12 is the following: notice how CEGIS, while constructing a counterexample for r_0, could potentially drop parameter Y from the conflict: looking at the family members, we know for sure that, once the system enters state s_1, it will definitely reach target state t with probability at least 0.6, exceeding threshold 0.3, regardless of the value of the parameter Y. If CEGIS managed to detect this, then in one iteration it could reject all realizations for which $r(X) = s_1$. However, since CEGIS focuses on a single Markov chain, it constructs a counterexample $\mathcal{D}_{r_0}{\downarrow}C$ wrt. this single chain: a constructed counterexample is the safest subsystem that refutes specification φ in the sense that, if an adversary attempted to add arbitrary states and paths to $\mathcal{D}_{r_0}{\downarrow}C$, it would still contradict φ. However, in our case, having a 'critical' subsystem $\mathcal{D}_{r_0}{\downarrow}\{s_{init}, s_1\}$, the adversary cannot extend the chain arbitrarily – he is restricted with his choice by the chains associated with realizations r_0 through r_3 since only these chains are valid from the point of view of the considered family $\mathcal{D}$, and therefore $\mathcal{D}_{r_0}{\downarrow}\{s_{init}, s_1\}$ is actually a perfectly valid counterexample.

How do we assist CEGIS in detecting that the set $\{s_{init}, s_1\}$ is critical for realization r_0? In other words, how can we be sure that, by entering state s_1, the system is bound to have at least 0.6 chance of reaching T? This is where MDP model checking comes into play: notice how CEGAR easily handled subfamily $\mathcal{D}[\mathcal{R}_2]$ for $\mathcal{R}_2 = \{s_1\} \times \{t, f\}$ by computing a lower bound 0.6 on the reachability probability. At this point, we come to a conclusion that, from the point of view of synthesis, a counterexample $\mathcal{D}_{r_0}{\downarrow}\{s_{init}, s_1\}$, and a quotient MDP for family $\mathcal{D}[\{s_1\} \times \{t, f\}]$ are the exact same construct: both manifest a minimum probability of 0.6 of reaching T and thus both allow to reject multiple realizations at once.

Both integrated methods discussed in the remainder of this chapter perform this kind of analysis, but both tackle it from different angles. A naive approach utilizes this peculiar correspondence between (incomplete) counterexamples and subfamilies of MCs, and switches between these representations on the fly in order to prune the state space of candidate solutions faster. An advanced technique executes this reasoning more precisely, at the level of individual states, allowing to compute a counterexample directly and avoid the computational overhead associated with frequent MDP model checker calls that is inherent to the naive approach.

5.1 The Naive Approach

The most straightforward way how to reinforce CEGIS with the power of MDP model checking goes as follows. Recall a CEGIS run in Example 12, where, for realization $r_0 : [X \mapsto s_1, Y \mapsto t]$ we constructed a critical set $C = \{s_{init}, s_1, t\}$ with $\{X, Y\}$ being the set of relevant parameters. Since $\mathcal{D}{\downarrow}C \not\models \varphi$, then, if we use the correspondence between counterexamples and quotient MDPs, it must hold that $M^{\mathcal{D}}[\{r_0\}] \not\models \varphi$. In hindsight, we know that parameter Y is actually irrelevant and parameter X isn't. At this point, having an MDP $M^{\mathcal{D}}[\{r_0\}]$, we might as well try to drop the assumption that e.g. Y is relevant, and instead model check MDP $M^{\mathcal{D}}[\{r_0, r_1\}]$ representing family members where $X = s_1$ and Y takes arbitrary value. In this case, MDP model checker gives a lower bound 0.6 on the reachability probability, confirming our belief that parameter Y is irrelevant. We have obtained a conflict $\{X\}$ and there is no reason to stop generalizing parameters other than Y. In particular, if we tried to generalize parameter X as well, we would obtain a quotient MDP $M^{\mathcal{D}}[\{r_0, r_1, r_2, r_3\}]$ with a minimum reachability probability 0.2 not exceeding threshold 0.3. Therefore, conflict $\{X\}$ cannot be improved, and we are now able, by analyzing in such way a counterexample for realization r_0, reject realization r_1 as well and continue with our search.

To summarize, a naive integration is essentially a CEGIS method, where, after computing a conflict set for an unsatisfying realization and before updating the state space of candidate solutions (between Lines 9 and 10 of Algorithm 3), we try to improve this conflict by attempting to generalize as many parameters as possible, as illustrated in Algorithm 5.

Algorithm 5: Improving conflicts via MDP model checking.

Input : A family $\mathcal{D} = (S, s_{init}, K, \mathcal{B})$ of MCs, a property φ, unsatisfying realization $r \in \mathcal{R}^{\mathcal{D}}$ (i.e. $\mathcal{D}_r \not\models \varphi$) and a conflict $\overline{K} \subseteq K$ (i.e. $M^{\mathcal{D}}[r{\uparrow}\overline{K}] \not\models \varphi$)

Output: An improved conflict $\hat{K} \subseteq \overline{K}$

1 **Function** `improveConflict`$(\mathcal{D}, \varphi, r, \overline{K})$**:**
2 $\hat{K} \leftarrow \overline{K}$
3 **for** $k \in \overline{K}$ **do**
4 **if** $M^{\mathcal{D}}[\hat{K} \backslash \{k\}] \not\models \varphi$ **then**
5 $\hat{K} \leftarrow \hat{K} \backslash \{k\}$
6 **end if**
7 **end for**
8 **return** $\hat{K}$

The designed method represents a compromise between bottom-up induction and top-down abstraction refinement. If the synthesis problem is the one that CEGIS handles easily, then the integrated method will also be able to find a solution quickly since it is essentially CEGIS. On the other hand, if the synthesis problem is the one that CEGAR handles easily because abstracting the family members into a quotient MDP does not yield too conservative bounds on the reachability probabilities (provided by MDP model checker), then the integrated method will be capable of efficiently generalizing conflicts computed in a CEGIS loop since now the method is equipped with MDP model checker as well. However, as will be shown during experimental evaluation in Chapter 6, the designed method does not offer anything new: it is comparable to both CEGIS and CEGAR, but, ultimately, it is unable to outperform any of these two. The main drawback is the generalization approach: for each counterexample, we attempt to generalize as many parameters as possible, thus invoking MDP model checker up to $|K|$ times per iteration: for families with many parameters, this leads to a huge computational overhead (recall that computing reachability for an MDP requires solving a linear program). Motivated by this, we go back to the roots of counterexample generation and try to apply everything we have learned so far in order to design a better integrated method.

5.2 Towards Improved Counterexample Generation

In the remainder of this chapter we wish to develop an integrated synthesis approach that utilizes bounds on the reachability probability for all (sub-)family members, computed via MDP model checking in a CEGAR loop, in order to construct smaller counterexamples in a CEGIS loop. Recall from Example 12 why supplying CEGIS with a bound on the reachability probability can be beneficial. Assume we computed, using MDP model checking techniques, a lower bound $\boldsymbol{x}_{\min}$ on the reachability probability for a quotient MDP $M^{\mathcal{D}}$ (no need for refinement). In particular, the value $\boldsymbol{x}_{\min}(s_1) = 0.6$ essentially says that, if the process $M^{\mathcal{D}}$ started in state s_1 and continued via paths present in members of family $\mathcal{D}$ only, then the probability of reaching T would be – across *all* realizations – at least 0.6. Going back to CEGIS and its analysis of a specific realization r_0, we have seen that it had computed a critical set $\{s_{init}, s_1, t\}$ inducing a set of relevant parameters $\{X, Y\}$; in this case, CEGIS is unable to detect that, given $X = s_1$, the state s_1 – and therefore parameter Y – are actually superfluous: for $X = s_1$ the system is guaranteed with probability 1 to reach state s_1 (simply from the definition of $\mathcal{B}(s_{init})$) and it is guaranteed (by the lower bound $\boldsymbol{x}_{\min}$) to have at least 0.6 probability of reaching T, thus violating the threshold $\lambda = 0.3$. Hence, the main challenge here is to somehow incorporate this bound on the reachability probability into counterexample construction. In this section, we only show how to carry out this incorporation in order to improve the quality of constructed counterexamples. Later, in Section 5.3, we demonstrate how this approach yields a powerful integrated synthesis method.

Let us perform a simple mental exercise, where we essentially explain the main process behind the advanced integration of CEGIS and CEGAR. Assume a Markov chain $D = (S, s_{init}, \boldsymbol{P})$ violating a safety property $\varphi \equiv \mathbb{P}_{\leq\lambda}[\mathrm{F}\ T]$. Remember that we consider chain D to be a member of the family $\mathcal{D}$, so let $\boldsymbol{\gamma} := \boldsymbol{x}_{\min}$ denote the lower bound on the reachability probability of T. To recapitulate, the value of $\boldsymbol{\gamma}(s)$ for arbitrary $s \in S$ represents that target states T are reachable from state s with probability at least $\boldsymbol{\gamma}(s)$. When constructing a counterexample to D and φ using search methods (see Section 2.1.2), we can imagine that, at the beginning, when no state has been discovered, each state $s \in S$ is 'connected'

to T via a transition with probability $\boldsymbol{\gamma}(s)$. Likewise, we can 'connect' each $s \in S$ to a new dummy state $s_\top$, which is also considered as a target one. By doing so, we are basically stating that the reachability probability for state s is $\boldsymbol{\gamma}(s)$, even when the chain D has no other transitions: this is consistent with $\boldsymbol{\gamma}(s)$ being a lower bound on the reachability probability. Then, we start to build up our reachable state space: we start from s_{init} and replace its transition to $s_\top$ with the original transitions $\boldsymbol{P}(s_{init})$. Since the successors of s_{init} are still connected to $s_\top$, the reachability probability for s_{init} must still be at least $\boldsymbol{\gamma}(s_{init})$ (otherwise, $\boldsymbol{\gamma}(s_{init})$ would not be a lower bound, see Algorithm 2), although it might actually be higher, since, by 'activating' transitions from state s_{init}, we actually introduced a concrete behavior inherent to chain D. By activating more and more states, we continue to increase the probability that $T \cup \{s_\top\}$ is reachable from the initial state s_{init}, up to a moment where it exceeds threshold λ (this is bound to happen since $D \not\models \varphi$). At this point, we have our counterexample. As was mentioned previously, this mental exercise was actually an illustration of how to employ lower bound $\boldsymbol{x}_{\min}$ for counterexample construction. The remainder of this section is dedicated to going through this exercise again, introducing all necessary constructs formally, and justifying how using $\boldsymbol{x}_{\min}$ in the process of counterexample construction described above actually yields smaller critical states. We begin with a formal description of 'connecting' unactivated states to $s_\top$ via transition $\boldsymbol{\gamma}(\cdot)$.

Definition 16. Let $D = (S, s_{init}, \boldsymbol{P})$ be an MC with $s_\top, s_\bot \notin S$. Let $C \subseteq S$ denote a set of *active states* and let $\boldsymbol{\gamma} : S \to [0, 1]$ denote a *rerouting vector*. *Rerouting* of Markov chain D wrt. active states C and rerouting vector $\boldsymbol{\gamma}$ is an MC $D{\downarrow}C[\boldsymbol{\gamma}] = (S \cup \{s_\bot, s_\top\}, s_{init}, \boldsymbol{P}^C_{\boldsymbol{\gamma}})$ with transition probability matrix $\boldsymbol{P}^C_{\boldsymbol{\gamma}}$ defined as follows:

$$\boldsymbol{P}^C_{\boldsymbol{\gamma}}(s) = \begin{cases} 1 : s & \text{if } s \in \{s_\top, s_\bot\}, \\ \boldsymbol{\gamma}(s) : s_\top + (1 - \boldsymbol{\gamma}(s)) : s_\bot & \text{if } s \in S \backslash C, \\ \boldsymbol{P}(s) & \text{otherwise.} \end{cases}$$

Rerouting an MC using a vector of probabilities $\boldsymbol{\gamma}$ involves introducing two absorbing states $s_\top$ and $s_\bot$ and then, for each non-activated state $s \in S \backslash C$, replacing its original distribution $\boldsymbol{P}(s)$ with two transitions: $\boldsymbol{\gamma}(s)$ leading to $s_\top$ and a complementary transition $(1 - \boldsymbol{\gamma}(s))$ leading to $s_\bot$. Algorithm 6 summarizes the state space exploration with the rerouting described above[1].

Example 13. Consider a family of MCs from Example 12. Let us pick an unsatisfiable realization r_0 and see how Algorithm 6 constructs a counterexample for the chain $D \coloneqq \mathcal{D}_{r_0}$ and the same safety property $\varphi \equiv \mathbb{P}_{\leq 0.3}[\mathrm{F}\ \{t\}]$, assuming we invoke it with the rerouting vector $\boldsymbol{\gamma} = \mathbf{0}$. Returning to the original semantics of $\boldsymbol{\gamma}(s)$ as a lower bound on the reachability probability, setting $\boldsymbol{\gamma}(s) = 0$ is equivalent to saying that T is unreachable from s until we activate concrete path from s to T. In other words, with $\boldsymbol{\gamma} = \mathbf{0}$ we compute an actual counterexample, without assuming that D is a family member of $\mathcal{D}$. Execution is illustrated in Figure 5.3. We introduce absorbing states $s_\top$ and $s_\bot$, where $s_\top$ is now also considered a target state. Although in this example these states are superfluous (since $\boldsymbol{\gamma} = \mathbf{0}$), they will play an important role later on (once $\boldsymbol{\gamma} \neq \mathbf{0}$), so let us think of them symbolically.

(a) We begin iteration 0 with an empty set $C^{(0)}$ of active states. Model checking $D^{(0)}$ from Figure 5.3a yields $\boldsymbol{x}^{(0)}(s_{init}) = 0 \leq 0.3$ and thus $D^{(0)}$ is not a counterexample: we need to introduce more paths.

[1] For now, assume that Algorithm 6 is invoked with a safety property.

Algorithm 6: Generation of a critical subsystem wrt. a reachability property φ and rerouting vector $\boldsymbol{\gamma}$.

Input : An MC $D = (S, s_{init}, \boldsymbol{P})$, a reachability property $\varphi \equiv \mathbb{P}_{\bowtie\lambda}[\mathrm{F}\ T]$ s.t. $D \not\models \varphi$, a rerouting vector $\boldsymbol{\gamma}$.
Output: A critical set C for D and φ.

Function `counterexampleRerouting`$(D, \varphi, \boldsymbol{\gamma})$**:**
- $i \leftarrow 0$
- $C^{(i)} \leftarrow \emptyset$
- **while** *true* **do**
  - $D^{(i)} \leftarrow D{\downarrow}C^{(i)}[\boldsymbol{\gamma}]$
  - $\boldsymbol{x}^{(i)} \leftarrow \mathrm{reachabilityMC}(D^{(i)}, T \cup \{s_\top\})$
  - **if** $\boldsymbol{x}^{(i)}(s_{init}) \not\bowtie \lambda$ **then**
    - **return** C
  - **end if**
  - $s^{(i)} \leftarrow \mathrm{next}(S \setminus C^{(i)})$
  - $C^{(i+1)} \leftarrow C^{(i)} \cup \{s^{(i)}\}$
  - $i \leftarrow i + 1$
- **end while**

(b) We activate the initial state s_{init} ($C^{(1)} = \{s_{init}\}$) and model check the corresponding subsystem $D^{(1)}$ depicted on Figure 5.3b. The obtained reachability probability $\boldsymbol{x}^{(1)}(s_{init})$ in the initial state is still $0 \leq 0.3$: set $C^{(1)}$ is not critical.

(c) We activate state s_1 ($C^{(2)} = \{s_{init}, s_1\}$) and model check $D^{(2)}$ (see Figure 5.3c), finally obtaining $\boldsymbol{x}^{(2)}(s_{init}) = 0.8 > 0.3$, i.e. $D^{(2)}$ is a critical subsystem.

Recall our discussion from Example 5 in Chapter 2 and notice that, again, a target state t does not belong to a critical set. Although this violates Definition 4 of a critical set (notice that $\mathcal{D}_{r_0}{\downarrow}\{s_{init}, s_1\}$ actually satisfies φ), we are safe since we are primarily interested in *transitions* of activated states: once transitions from states s_{init} and s_1 are activated, we can reach target state t with the probability exceeding threshold λ. Therefore, it is completely irrelevant what were the original transitions from state t (or from state f, for that matter). The reason we complicate this notion of a critical subset is that, from the point of view of synthesis, transitions emanating from t (and therefore values of parameters used for defining these transitions) were not important to refute φ. Therefore, we want to declare as critical only states that were activated (i.e. those the transitions from which are actually used), to define a conflict (a set of relevant parameters) as per Definition 13. In our example, we are interested in parameters X and Y used in activated states s_{init} and s_1, confirming that, from the point of view of CEGIS, both X and Y are relevant parameters to reject r_0.

The following proposition brings more light into how Algorithm 6 operates internally. The proposition is stated with only an outline of a proof.

Proposition 4. Let $\varphi \equiv \mathbb{P}_{\leq\lambda}[\mathrm{F}\ T]$ be a safety property and $D = (S, s_{init}, \boldsymbol{P})$ be an MC such that $D \not\models \varphi$. Let $\boldsymbol{\chi} = \mathrm{reachabilityMC}(D, T)$ be a vector of exact reachability probabilities for D. Similarly, let $\boldsymbol{x}^{(i)} = \mathrm{reachabilityMC}(D^{(i)}, T)$ for $0 \leq i \leq |S|$, where chains $D^{(i)}$ are consistent with Algorithm 6 executed with rerouting vector $\boldsymbol{\gamma} \leq \boldsymbol{\chi}$. Then $\boldsymbol{\gamma} \leq \boldsymbol{x}^{(0)} \leq \boldsymbol{x}^{(1)} \leq \cdots \leq \boldsymbol{x}^{(|S|)} = \boldsymbol{\chi}$.

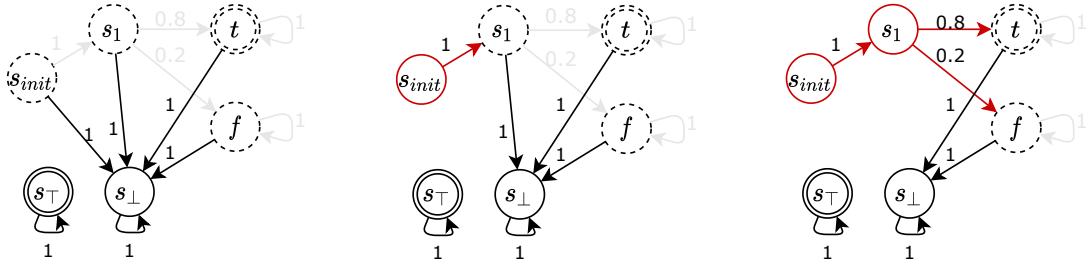


Figure 5.3: Constructing a counterexample to $\mathcal{D}_{r_0}$ and φ from Example 12 using Algorithm 6 and rerouting vector $\boldsymbol{\gamma} = \mathbf{0}$.

Proposition 4 essentially states that, if Algorithm 6 is executed on an MC D and a safety property φ with the rerouting vector $\boldsymbol{\gamma}$ being a lower bound of the true reachability probabilities $\boldsymbol{\chi}$ for D, then, in each iteration, by activating more and more states – and introducing more and more paths to the subsystem $D^{(i)}$ – the sequence $(\boldsymbol{x}^{(i)})_{i=0}^{|S|}$ of reachability probabilities computed on Line 6 is nondecreasing. To see why this is the case, let us consider separate cases. Recall notation associated with computing reachability probabilities in Algorithm 1. If $s \in S_1 = T$, i.e. $\boldsymbol{\chi}(s) = 1$, then also $\boldsymbol{x}^{(i)}(s) = 1$, implying $\boldsymbol{\gamma}(s) \leq \boldsymbol{x}^{(i)}(s) \leq \boldsymbol{\chi}(s)$. If $s \in S_0$ is a state from which T is unreachable, i.e. $\boldsymbol{\chi}(s) = 0$, then T is unreachable from s in any subsystem $D^{(i)}$; also, $s_\top$ must be unreachable from s as well since it is 'connected' to state s via a transition $\boldsymbol{\gamma}(s) \leq \boldsymbol{\chi}(s) = 0$. Finally, let us consider a general case $s \in S_?$. Until s is activated and original transitions are used, the only 'target' state reachable from s is $s_\top$, and therefore the reachability probability equals $\boldsymbol{\gamma}(s)$. Assume we observe some iteration i, when state s has already been activated. Reachability probability $\boldsymbol{x}^{(i)}(s)$ must satisfy

$$\boldsymbol{x}^{(i)}(s) = \sum_{s' \in S} \boldsymbol{P}(s,s')\boldsymbol{x}^{(i)}(s') = \sum_{s' \in C} \boldsymbol{P}(s,s')\boldsymbol{x}^{(i)}(s') + \sum_{s' \in S \setminus C} \boldsymbol{P}(s,s')\boldsymbol{\gamma}(s').$$

If none of the successors of s is activated, then $\boldsymbol{x}^{(i)}(s) = \sum_{s' \in S \setminus C} \boldsymbol{P}(s,s')\boldsymbol{\gamma}(s')$ must be at least $\boldsymbol{\gamma}(s)$ (otherwise, $\boldsymbol{\gamma}$ would not be a lower bound on the reachability probability), i.e. $\boldsymbol{\gamma}(s) \leq \boldsymbol{x}^{(i)}(s)$. On the other hand, if state s is connected to some activated states, there might exist some paths leading to T and contributing more probability mass to the overall value of $\boldsymbol{x}^{(i)}(s)$, in which case, again, $\boldsymbol{\gamma}(s) \leq \boldsymbol{x}^{(i)}(s)$. The value of $\boldsymbol{x}^{(i)}(s)$, however, cannot exceed $\boldsymbol{\chi}(s)$ since only by expanding all states and accounting all possible paths leading to T we obtain the value $\boldsymbol{\chi}(s)$. Hence, $\boldsymbol{\gamma}(s) \leq \boldsymbol{x}^{(i)}(s) \leq \boldsymbol{\chi}(s)$. The last remark also justifies why $\boldsymbol{x}^{(|S|)} = \boldsymbol{\chi}$.

Proposition 4 is a key to unlocking a superior counterexample generation technique. Let us again return to Example 13. According to Proposition 4, $\boldsymbol{\gamma} \leq \boldsymbol{x}^{(0)} \leq \boldsymbol{x}^{(1)} \leq \boldsymbol{x}^{(2)} \leq \boldsymbol{\chi}$. Without computing exact values of vectors $\boldsymbol{x}^{(i)}$, we can at least assert that the proposition holds for the initial state ($0 \leq 0 \leq 0 \leq 0.8 \leq 0.8$): recall that, in Example 13, we executed Algorithm 6 with the lower bound $\boldsymbol{\gamma} = \mathbf{0} \leq \boldsymbol{\chi}$ on the reachability probability. Now consider we execute Algorithm 6 with $\boldsymbol{\gamma} = \boldsymbol{\chi}$. We would obtain a rerouting of $\mathcal{D}_{r_0}$, where the initial state s_{init} is connected to $s_\top$ via a transition $\boldsymbol{\gamma}(s_{init}) = \boldsymbol{\chi}(s_{init}) = 0.8$, immediately yielding a 'counterexample' with the critical set $C = \emptyset$. The interpretation of such behavior goes as follows. The case $\boldsymbol{\gamma} = \mathbf{0}$ is the most pessimistic one: we assume a family of *all* Markov chains, having a lower bound $\mathbf{0}$ on a reachability probability, and

then we are asking, given $D \not\models \varphi$, what is an essential part of D that makes it violate φ? In other words, what fraction $D{\downarrow}C$ of D is sufficient to reject not only D but also those chains that share the same subsystem $D{\downarrow}C$? On the other hand, if we take $\boldsymbol{\gamma} = \boldsymbol{\chi}$, we are essentially asking: given that $D \not\models \varphi$ (remember that $\boldsymbol{\chi}(s_{init}) > \lambda$), what fraction of D is sufficient to prove that $D \not\models \varphi$? Well, $C = \emptyset$ is enough: $D \not\models \varphi$ follows trivially from $D \not\models \varphi$, period. The two cases above represent two extreme cases for counterexample generation: the pessimistic case $\boldsymbol{\gamma} = \mathbf{0}$, where we deal with a family of all Markov chains and try to safely reject as many possible chains as possible, and a trivial case $\boldsymbol{\gamma} = \boldsymbol{\chi}$ where we deal with a family $\{D\}$ of Markov chains and there is nothing more to reject other than D that a priori violates φ. This comparison is peculiar for one specific reason. Notice how computing counterexample for a family of all Markov chains is hard (it took Algorithm 6 three iterations to arrive at a critical set inducing two relevant parameters) and computing counterexample for a small family $\{D\}$ is almost trivial (one model checking call, after which all parameters are declared irrelevant), and the only difference in the execution of Algorithm 6 was the lower bound $\boldsymbol{\gamma}$ it was supplied with. Essentially, the closer $\boldsymbol{\gamma}$ to $\boldsymbol{\chi}$, the better the quality of produced counterexamples.

At this point it should be obvious how one would improve counterexample generation for a family $\mathcal{D}$ of MCs and a safety property φ: construct a quotient MDP $M^{\mathcal{D}}$ and invoke an (analog of) MDP model checking Algorithm 2 to compute a lower bound $\boldsymbol{x}_{\min}$ on the reachability probability for all chains in family $\mathcal{D}$, and then supply $\boldsymbol{x}_{\min}$ as the rerouting vector $\boldsymbol{\gamma}$ for subsequent counterexample generation. In cases where the lower estimate $\boldsymbol{x}_{\min}$ is not too conservative ($\boldsymbol{x}_{\min} \not\approx \mathbf{0}$) then we have a chance for obtaining smaller counterexamples (relevant to not all MCs, but at least to those that are considered in the family $\mathcal{D}$), thus pruning the state space of candidate solutions faster. Let us test this conjecture in practice.

Example 14. We return again to the family $\mathcal{D}$ of MCs and the safety property $\varphi \equiv \mathbb{P}_{\leq 0.3}[\mathrm{F}\ \{t\}]$ from Example 12. Model checking a quotient MDP $M^{\mathcal{D}}$ yields the following lower bound on the reachability probability (can be easily read from Figure 5.2): $\boldsymbol{x}_{\min} : [s_{init} \mapsto 0.2, s_1 \mapsto 0.6, s_2 \mapsto 0.2, t \mapsto 1, f \mapsto 0]$. Let us now follow the steps of CEGIS and consider a realization r_0, which, as we already know, is unsatisfiable. We now invoke Algorithm 6 with $\boldsymbol{\gamma} = \boldsymbol{x}_{\min}$ to construct the corresponding counterexample. The execution is illustrated in Figure 5.4. We begin iteration 0 with an empty set $C^{(0)}$ of active states. Model checking $D^{(0)}$ (see 5.4a) yields $\boldsymbol{x}^{(0)}(s_{init}) = 0.2 \leq 0.3$ and thus $D^{(0)}$ is not a counterexample. We then activate s_{init}, $C^{(1)} = \{s_{init}\}$, and now model checking $D^{(1)}$ (depicted in Figure 5.4b) yields $\boldsymbol{x}^{(1)}(s_{init}) = 0.6 > 0.3$, completing the counterexample generation. The only relevant parameters are those used in activated transitions from state s_{init}, i.e. $\overline{K} = \{X\}$, improving upon the previously computed conflict $\{X, Y\}$.

One can view transition $\boldsymbol{\gamma}(s) = \boldsymbol{x}_{\min}(s)$ from inactive s to a target state $s_\top$ as a 'shortcut' representing the least possible set of transitions connecting s and T in all MCs in the considered family $\mathcal{D}$. In other words, by ignoring exact transitions of inactive s, we retain information that the reachability probability in s is not zero (as considered in pure CEGIS) but at least $\boldsymbol{x}_{\min}(s)$. Furthermore, if in the obtained counterexample the state s remains inactive, it means that, in order to refute φ, the system might have to use paths connecting s and T, but the contributed probability is enough to exceed threshold λ, regardless of the exact values of the parameters that are considered on these paths. In our example, during iteration 0 we have e.g. a shortcut with value 0.2 from state s_{init} to a virtual target state $s_\top$, illustrating the fact that it is possible to reach T with probability at least 0.2, regardless of concrete transitions available in state s_{init} and beyond. In some

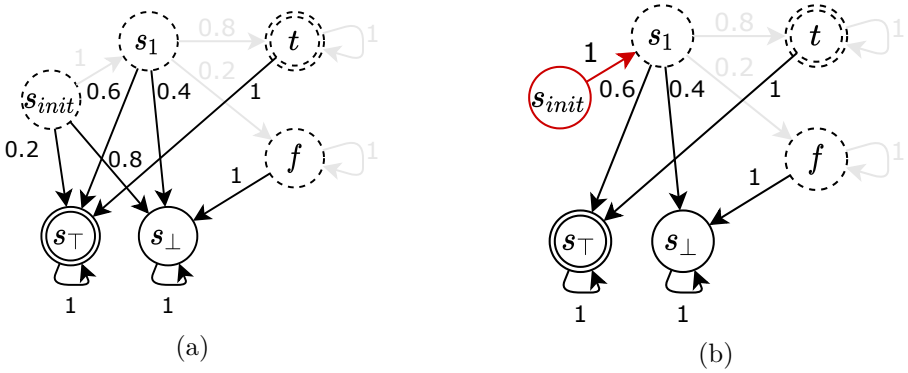


Figure 5.4: Constructing a counterexample to $\mathcal{D}_{r_0}$ and φ from Example 12 using Algorithm 6 and rerouting vector $\boldsymbol{\gamma} = \boldsymbol{x}_{\min}$.

cases, the value of $\boldsymbol{x}_{\min}(s)$ might represent impossible reachability value since it is based on inconsistent schedulers – for consistent ones the value is even higher – but we are satisfied with a lower bound. Unfortunately, during iteration 0 the value 0.2 is not enough to refute φ, and therefore we activate s_{init}. By doing so, we introduce to subsystem $D^{(0)}$ behavior that is specific to the family member $\mathcal{D}_{r_0}$, and continue to particularize this chain until it represents few enough family members, all of them violating φ. In our case, by activating s_{init}, we declare that the subsystem must at least enter state s_1 (since $X = s_1$), and therefore the considered counterexample will be applicable to those family members that also assume $X = s_1$. From state s_1 having a minimum probability of 0.6 of reaching T (as reflected in transition 0.6 to $s_\top$), it is now sufficient to violate threshold λ and thus complete the construction of the counterexample. The obtained conflict set involves only parameters associated with the activated transitions, i.e. $\{X\}$.

Before we describe in more detail how to properly use this improved counterexample generation approach in the context of probabilistic synthesis, let us complete our improved counterexample generation by considering liveness properties as well.

Proposition 5. Let $\varphi \equiv \mathbb{P}_{\geq\lambda}[\mathrm{F}\ T]$ be a liveness property and $D = (S, s_{init}, \boldsymbol{P})$ be an MC such that $D \not\models \varphi$.Let $\boldsymbol{\chi} = \mathrm{reachabilityMC}(D, T)$ be a vector of exact reachability probabilities for D. Similarly, let $\boldsymbol{x}^{(i)} = \mathrm{reachabilityMC}(D^{(i)}, T)$ for $0 \leq i \leq |\ S\ |$, where chains $D^{(i)}$ are consistent with Algorithm 6 executed with the rerouting vector $\boldsymbol{\gamma} \geq \boldsymbol{\chi}$. Then $\boldsymbol{\gamma} \geq \boldsymbol{x}^{(0)} \geq \boldsymbol{x}^{(1)} \geq \cdots \geq \boldsymbol{x}^{(|S|)} = \boldsymbol{\chi}$.

Justification of Proposition 5 is analogous to that of Proposition 4 concerning safety formulae. To construct a counterexample for a liveness property, we begin with a pessimistic estimate that each state can reach T with probability $\boldsymbol{\gamma}(\cdot)$ and then gradually add behavior inherent to D, until we manage to trap enough probability outside of T. Basically, every conclusion we have reached regarding the safety properties can be applied to liveness formulae as well, up to the interchange of operators. In particular, for constructing a counterexample for a family $\mathcal{D}$ of MCs, one must compute an upper bound $\boldsymbol{x}_{\max}$ on the reachability probability and then use it for rerouting in Algorithm 6. Furthermore, Algorithm 6 can be used for synthesizing chains wrt. reward properties as well, since, numerically, these work on similar principles as reachability properties. Hence, we obtain a unified algorithm capable of handling a huge variety of specifications, as opposed to ex-

isting algorithms for counterexample generation, where handling liveness specifications or even reward properties requires additional manipulations with model and/or specification.

5.3 The Advanced Approach

Although we have already seen how to properly facilitate MDP model checking for counterexample generation – compute lower/upper bound on the reachability probability and then run a CEGIS loop, where counterexamples are computed using Algorithm 6 – a couple of implementation details are still worth mentioning. First of all, although we might use lower/upper bound for the quotient MDP $M^{\mathcal{D}}$ directly, this bound might be too conservative ($\boldsymbol{x}_{\min} \approx \mathbf{0}$ and $\boldsymbol{x}_{\max} \approx \mathbf{1}$) and thus offer little to no help during counterexample generation. Therefore, it makes sense to run CEGAR loop for some time and let it refine $\mathcal{D}$ into a set of subfamilies for which the corresponding bounds are usable. In our algorithm, we use a parameter $mdpMClimit$ to limit the number of MDP model checking calls to a constant value. If, after reaching this limit, CEGAR still has undecidable (but analyzed) subfamilies, we stop CEGAR phase and start analyzing each of these subfamilies separately in a CEGIS loop, where we use bounds associated with the corresponding subfamily for rerouting. Surprisingly, during the experimental evaluation, it will be shown that, in the majority of cases, even the value $mdpMClimit = 1$ (implying no splitting into subfamilies) can be optimal, and the value $mdpMClimit = 7$ (implying the depth two of recursive splitting) was a safe pick for all observed cases.

Second, during counterexample construction in Algorithm 6, it is unlikely that, for a family member with thousands of states, activating one state at a time is reasonable: recall that, during each iteration, we have to invoke MC model checker. Instead, we propose to activate multiple states at a time to avoid this computational overhead. In our implementation, this is again dictated by a parameter $activatePerIter$ representing a number of states activated before subsequent model checks. Intuitively, small values of $activatePerIter$ require frequent model checks but yield smaller counterexamples, since exceeding the threshold λ is detected sooner. Conversely, large values of $activatePerIter$ save model checking time but might result in poor counterexamples, in the extreme cases even yielding conflict sets $\overline{K} = K$, thus discarding the advantage of MDP model checking results. In practice, however, we noticed little to no performance fluctuation between similar values of $activatePerIter$: provided that the chosen value is not extreme (i.e. not 1 and not $|S|/2$), the designed algorithm provides sound results. Usually, we recommend choosing the value of $|S| \cdot 0.05$, i.e. activating 5% of the state space at a time.

Finally, we have not yet discussed how the method *next* used on Line 10 of Algorithm 6 is implemented. Until now it was irrelevant: any order of activation of reachable states suffices, including e.g breadth-first or depth-first search. However, for real-world applications, where families have millions of members and each member has thousands of states, a 'good' order can provide ten times smaller counterexamples than a 'bad' one. In particular, one might get inspired by some informed search heuristics discussed in [3, 4]. In our research, we considered the simpler approaches discussed below. As will be shown later, they will prove to be efficient enough, although they still leave some space for subsequent research. Assume a safety property $\varphi \equiv \mathbb{P}_{\leq\lambda}[\mathrm{F}\ T]$ and an MC D violating φ. Let $\boldsymbol{\chi} = \mathrm{reachabilityMC}(D, T)$ denote the exact reachability probabilities for D and that the procedure is executed with the rerouting vector $\boldsymbol{\gamma}$ containing lower bound $\boldsymbol{x}_{\min}$ on the reachability probability. Assume we have already activated i states in the set $C^{(i)}$, computed the corresponding $\boldsymbol{x}^{(i)}$ (reachability probabilities for subsystem $D^{(i)}$), and still do not have a counterexample for

φ. Let $B^{(i)} := Succ(C^{(i)}) \setminus C^{(i)} \subseteq S$ denote a set of states that are reachable (in one step) from the activate subspace, yet not have been activated yet. We store states ready for activation from $B^{(i)}$ in a priority queue, where we activate a state $s \in \arg\max_{s \in B^{(i)}} \boldsymbol{\upsilon}(s)$ having the maximum value of the key $\boldsymbol{\upsilon}(s)$. For possible candidates for key $\boldsymbol{\upsilon}$, we considered vectors $\boldsymbol{x}_{\min}$, $\boldsymbol{x}_{\max}$, $\boldsymbol{\chi}$, $\boldsymbol{x}^{(i)}$ or their combinations. Experimental evaluation associated with these strategies will be discussed in Chapter 6, here we will mention that the most suitable key vector was $\boldsymbol{\chi}$. Ultimately, activating a state with the maximum reachability probability for a given MC D (for which we, essentially, construct a counterexample) will prioritize states that are closer (in a probabilistic sense) to T, and therefore satisfying paths contributing to a reachability probability exceeding λ are constructed much faster. Similarly, if we analyze a liveness property, we choose a state $s \in B^{(i)}$ with a minimum reachability probability $\boldsymbol{\chi}(s)$. To summarize, the proposed synthesis technique for family $\mathcal{D}$ and a reachability property φ consists of two phases.

- *[CEGAR phase]* Analyze family $\mathcal{D}$ in a CEGAR loop (see Algorithm 4) with the use of at most $mdpMClimit$ $(= 7)$ MDP model checker calls. Whenever a feasible solution is found, return it. Whenever all analyzed subfamilies were shown to contain no solutions, return UNSAT. Otherwise, let $\{(\mathcal{D}_k, \boldsymbol{\gamma}_k)\}_k$ be a set of undecidable families $\mathcal{D}_k$, with $\boldsymbol{\gamma}_k$ being the corresponding bound on the reachability probability for each $s \in S$ (lower bound for safety, upper bound for liveness).
- *[CEGIS phase]* For each subfamily $\mathcal{D}_k$, initiate a CEGIS loop (see Algorithm 3). For constructing counterexamples, use Algorithm 6 with $\boldsymbol{\gamma}_k$ as a rerouting vector and activate $activatePerIter$ $(= 5\%)$ states before subsequent MC model checks. Whenever a feasible solution if found, return it. If none of the subfamilies contains a solution, return UNSAT.

Chapter 6

Experimental Evaluation

Both integrated methods discussed in Chapter 5 were implemented in Storm [27] – a state-of-the-art probabilistic model checker. Auxiliary procedures (building and checking models, CEGIS and CEGAR adapters) were implemented as a Python module using the Storm Python API. For SMT solving, we use Z3 [35]. The toolchain takes a PRISM [33] or JANI [11] sketch and a PCTL specification, and returns a satisfying hole assignment, if such exists. In the following set of experiments, we will investigate the performance of the synthesis methods for various models and properties. We will be interested in how our integrated methods compare to two state-of-the-art synthesis approaches for probabilistic synthesis: program-level CEGIS [13] and CEGAR [14]. All of the experiments are run on the Ubuntu 19.04 machine with Intel i5-8300H (4 cores at 2.3 GHz) and using up to 8 GB RAM, with all the algorithms being executed sequentially (1 thread). The benchmark consists of five different models from various domains. In no particular order, these include:

- *Grid* represents a benchmark for solving partially observable MDPs (POMDPs) [29], where an agent attempts to locate a target in a 4x4 grid. Specifications include a lower- and upper- bounded properties on the number of steps. The number of holes: 6; family size: 1.7k members; the size of the quotient MDP: 9k states; the average size of a family member: from 0.6k (safety) to 1.2k (liveness) states.

- *Pole* considers a balancing pole in a noisy environment (motivated by [7, 20]). A specification is either lower- or upper-bounded probability of balancing the pole for at least a given amount of time. The number of holes: 17; family size: 1.3M members; the size of the quotient MDP: 17k states; the average size of a family member: from 11k (liveness) to 14k (safety) states.

- *Maze* is inspired by the infamous cheese-maze example [37]: a simple maze where an agent (similarly to *Grid*) attempts to reach a target location. A specification is either lower- or upper-bounded target state reachability. The number of holes: 20; family size: 1M members; the size of the quotient MDP: 9k states; the average size of a family member: from 5k (liveness) to 7k (safety) states.

- *DPM* considers a scheduler for a disk power manager (motivated by [9, 23]). A specification is a safety property on a service queue overflow. The number of holes: 16; family size: 43M members; the size of the quotient MDP: 27k states; the average size of a family member: 4k states.

- *Herman* considers a distributed Herman protocol for self-stabilizing rings [28, 34]. A specification is a liveness property asserting that stabilization takes place after one round. The number of holes: 8; family size: 0.5k members; the size of the quotient MDP: 54k states; the average size of a family member: 2k states.

6.1 Evaluating the Naive Approach

First, let us briefly evaluate the naive integration: generalizing conflicts using MDP model checking. We consider a (slightly smaller version of) *Grid* benchmark with a liveness property $\varphi \equiv \mathbb{P}_{\geq\lambda}[\mathrm{F}\ T]$ and investigate how the three methods: CEGIS, CEGAR, and their naive integration, handle property φ with varying threshold λ. The results are reported in Table 6.1, where for each of the four different values of the threshold we report the computation time (time, in milliseconds) as well as the number of iterations (iters): MC model checking calls for CEGIS, MDP model checking calls for CEGAR and both (MC+MDP) in the case of the integrated approach.

λ	CEGIS		CEGAR		Naive	
	time	iters	time	iters	time	iters
0.10	69	3	249	8	218	3+6
0.40	413	17	402	24	465	5+14
0.75	5062	191	821	72	1894	16+60
0.90	12113	452	44	2	84	1+2

Table 6.1: Experimental evaluation of three methods for model synthesis.

First, let us investigate how CEGIS and CEGAR alone handle this family of Markov chains. Notice that, with the increasing value of the threshold λ, we decrease the chance that any particular model satisfies the liveness property: larger values of λ correspond to stricter formulae. This is reflected in the behavior of CEGIS, where, for instance, for the value 0.1 of λ, it was lucky enough to find a satisfiable assignment on the third try. On the other hand, for the value of 0.9, it takes CEGIS twelve seconds and hundreds of iterations to exhaust the state space and thus show that the satisfiable assignment does not exist. On the other hand, CEGAR handles the limiting cases quite well: if the threshold is too small or too large, then, after a couple of refinements, imprecise under- and over-approximations provided by MDP model checking are accurate enough to reach a conclusive result. However, for the values of the threshold that evenly splits the state space of candidates into satisfiable and unsatisfiable ones (around 0.75 in this case), CEGAR struggles for a while until it obtains a refined partitioning.

Finally, let us turn our attention to the integrated method. For the value 0.1 of the threshold λ, where CEGIS is slightly favorable, the integrated method performs the exact same three MC model checking calls and tries (unsuccessfully) during each iteration to generalize relevant parameters, hence six additional MDP model checking calls. The resulting time is worse than for pure CEGIS, but it still beats the abstraction refinement scheme. In the case when $\lambda = 0.4$, where there is no clear winner between CEGIS and CEGAR, the computational overhead in the integrated scheme comes into play and the resulting time is slightly worse than for pure methods. However, notice how in this case MDP model checking starts to help our integrated method: during the first four iterations

it managed to successfully detect and generalize some irrelevant parameters, which lead it to the satisfiable assignment already in the fifth iteration, instead of 17th in the case of CEGIS. With the increasing value of the threshold λ, now being 0.75, the CEGIS approach becomes less and less favorable compared to CEGAR. Nonetheless, the integrated method manages to leverage the situation with the help of the MDP model checking and the resulting time is much better as compared to pure CEGIS, although it still loses to the CEGAR approach. Finally, when the threshold λ goes to the extreme values and CEGIS bleaks in comparison to CEGAR, which manages to prune the state space in two calls, the integrated method again represents a middle ground between the two methods and is capable of pruning the state space almost as efficiently as pure CEGAR: here the value 1+2 in the table cell represents one MC model checking call, where it managed to provide a counterexample where only two parameters are relevant, and then two subsequent MDP model checking calls where the integrated method successfully generalized both of these parameters and hereby completed the synthesis.

From the experiment, we can see the integrated method confirms our expectations in the sense that it is perfectly capable of applying MDP model checking in order to improve the quality of generated conflicts and is capable, in some cases, of pruning the state space more efficiently. Performance-wise, it works fine: only in one case ($\lambda = 0.4$) it is slightly outperformed by both methods, otherwise, it tends to emulate a superior approach (CEGIS for $\lambda = 0.1$, CEGAR for $\lambda \in \{0.75, 0.9\}$), although never outperforming both of the existing methods. To summarize, the designed method is an adequate approach for probabilistic synthesis, although it does not represent anything special. As will be shown shortly, the advanced approach offers much more from the point of view of performance. Therefore, we abandon the naive approach completely, and, from now on, we reserve the term 'integrated method' exclusively for the synthesis approach that uses estimates of reachability probability for the construction of better counterexamples.

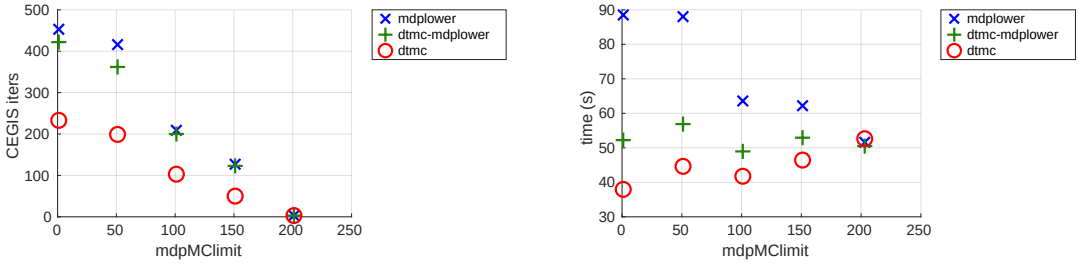
6.2 Tuning the Integrated Method

Before comparing the integrated method with existing approaches, let us first report some interesting observations regarding the strategy design for this method. In particular, we wish to investigate (1) different strategies for state activation and (2) how the value of parameters *mdpMClimit* and activatedPerIter determines the behavior of model synthesis. Regarding the first tangent, recall that we chose to use a priority queue for reached but not yet activated states, where we activate a state s having the maximum (with safety properties) or minimum (with liveness properties) value of $\boldsymbol{v}(s)$. Here a choice of vector $\boldsymbol{v}$ represents our strategy for state activation. Assume we deal with a safety property. After investigating several candidates, the following three strategies were shown to be the most interesting.

1. $\boldsymbol{v} = \boldsymbol{x}_{\min}$: this strategy represents activating a state with a maximum lower bound on the reachability probability, potentially representing a beginning of a path that would lead in all members of a family to the set T of target states.

2. $\boldsymbol{v} = \boldsymbol{\chi}$: this strategy, compared to the previous one, tries instead to activate a state having a maximum reachability probability for this particular chain.

3. $\boldsymbol{v} = \boldsymbol{\chi} - \boldsymbol{x}_{\min}$: this strategy takes a different approach and, instead of looking for maximum probabilities, either for a whole family of chains of this particular chain only,



(a) number of iterations

(b) execution time (s)

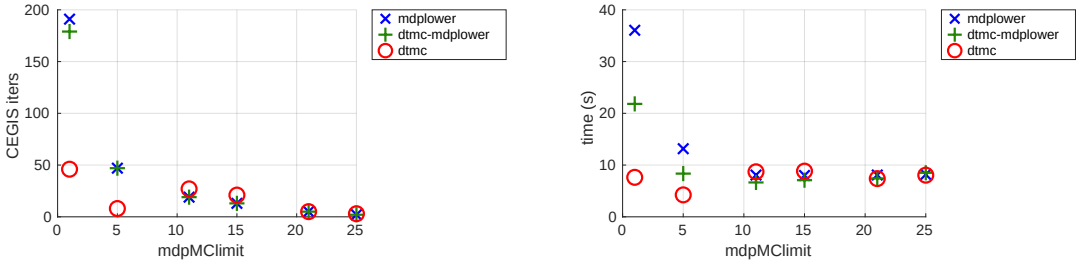
Figure 6.1: Investigating activation strategies for Grid model, safety property, threshold is $\lambda_1 = 0.28$ (unfeasible).

it takes a difference between the two. The idea here is to pick a state in the direction where we maximize the difference between the pessimistic estimate $\boldsymbol{x}_{\min}$ and actual probabilities $\boldsymbol{\chi}$ that we are suspecting to achieve if we continue to expand more and more states.

As a quick note, observe that using upper bound $\boldsymbol{x}_{\max}$ or the intermediate model checking result $\boldsymbol{x}^{(i)}$ does not make sense. The first vector $\boldsymbol{x}_{\max} \geq \boldsymbol{\chi}$ represents probabilities that are most likely unachievable in the given member; in the second case, since we evaluate these vectors for inactive state s, one can see that $\boldsymbol{x}^{(i)}(s) = \boldsymbol{\gamma}(s) = \boldsymbol{x}_{\min}(s)$, i.e. $\boldsymbol{x}^{(i)}$ has the same semantics as $\boldsymbol{x}_{\min}$. As for the three strategies mentioned above, we do not have sufficient justification to prefer any of the three, so it is better to allow empirical evidence to speak for itself. In the following experiment, we consider the *Grid* benchmark and a safety specification, where the threshold λ first takes value $\lambda_1 = 0.28$, for which the corresponding synthesis problem is unfeasible. Additionally, we execute the algorithm with different values of *mdpMClimit*. The results are reported in Figure 6.1, where, for each of the three strategies discussed above – $\boldsymbol{\upsilon} = \boldsymbol{x}_{\min}$ (mdplower), $\boldsymbol{\upsilon} = \boldsymbol{\chi}$ (dtmc) and $\boldsymbol{\upsilon} = \boldsymbol{\chi} - \boldsymbol{x}_{\min}$ (mdplower-dtmc), we first report the number of CEGIS iterations (i.e. constructed counterexamples) required for the synthesis, and then the synthesis time (in seconds). For reference, pure CEGIS requires 453 iterations, whereas pure CEGAR requires 203 iterations.

Regarding the number of CEGIS iterations wrt. the allowed number *mdpMClimit* of CEGAR iterations, we, as expected, observe the inverse dependency: the more we refine the subfamilies in the CEGAR phase, the easier it is to prune the state space in the CEGIS phase. As we can see, the expansion strategy $\boldsymbol{\upsilon} = \boldsymbol{\chi}$, by means of constructing better (that is, smaller) counterexamples, and therefore by pruning the state space faster, requires the least amount of CEGIS iterations, as compared to other strategies. This is immediately reflected in the performance of the corresponding strategies, where choosing $\boldsymbol{\upsilon} = \boldsymbol{\chi}$ displays the best behavior. In Figure 6.2, the experiment is repeated with threshold $\lambda_2 = 0.30$ (for which the synthesis problem is feasible) with the same results. For reference, in this case, pure CEGIS requires 191 iterations, whereas pure CEGAR requires only 20 iterations.

Although this experiment answers which expansion strategy is the most efficient one (from now on, we will be using exclusively $\boldsymbol{\upsilon} = \boldsymbol{\chi}$, i.e. exact values of reachability probalities), it remains unclear how to tune the parameter *mdpMClimit*. For $\lambda_2 = 0.30$, the optimal solution seems to be around *mdpMClimit* $= 5$, although, for $\lambda_1 = 0.30$, even the value *mdpMClimit* $= 1$ seems enough: here we model check the quotient MDP and, even without refining, obtain the lower bound on the reachability probability adequate enough to allow,



(a) number of iterations (b) execution time (s)

Figure 6.2: Investigating activation strategies for Grid model, safety property, threshold is $\lambda_2 = 0.30$ (feasible).

in the CEGIS phase, to prune the state space in as low as 46 iterations (as compared to 191 for pure CEGIS), and therefore refining subfamilies even further is, in fact, excessive. Of course, the exact behavior will depend on the model and on the property. However, as was mentioned earlier, we rarely encountered cases where picking $mdpMClimit = 1$ was not optimal, although, even in such cases, picking the value of $mdpMClimit$ larger than 10 is unreasonable: we recommend to choose the value of $mdpMClimit$ around 7.

Finally, let us investigate how the value of parameter $activatePerIter$ influences the execution of the integrated algorithm. Intuitively, we expect, with the increasing values of $activatePerIter$, 1) spend less time model checking candidate counterexamples, thus avoiding the computational overhead, and 2) provide larger counterexamples, thus pruning the state space of candidate solutions slower and require more iterations in the CEGIS phase. In Figure 6.3 we report the results of the corresponding experiment, confirming our expectations. Here we pick the same *Grid* model and a safety property (i.e. average family member has approx. 600 states) and execute integrated algorithm with $mdpMClimit = 1$ and $activatePerIter$ ranging from 1 to 65. For each case, we report a number of iterations in the CEGIS phase and execution time, in seconds. We confirm that larger values of parameter $activatePerIter$ correspond to larger counterexamples, requiring more iterations. We also confirm that both extremely small and extremely large values of $activatePerIter$ result in rather poor behavior: either because activating one to two states at a time is unreasonable, or because activating a hundred of states at a time yields large counterexamples. To put it into perspective, for this model and property, it takes pure CEGIS around 20 seconds and 453 iterations to exhaust the state space – the same amount of iterations as for integrated method with $activatePerIter \geq 65$. As was mentioned previously, we suggest picking the value of $activatePerIter$ to be around 5% of the state space of a Markov chain. In such cases, we see that the integrated approach is capable of constructing smaller counterexamples and requires much fewer iterations, in some cases achieving a threefold performance increase.

6.3 Performance Evaluation of the Synthesis Methods

Having decided how to set the parameters of the integrated algorithm properly, we are now ready to compare how the method performs compared to pure CEGIS and CEGAR. We evaluate all three methods on a set of five models mentioned at the beginning of this chapter, some of which having different properties of interest, resulting in a total of eight different experiments. For each model and property $\mathbb{P}_{\bowtie\lambda}[\mathrm{F}\ T]$, we investigate how the per-

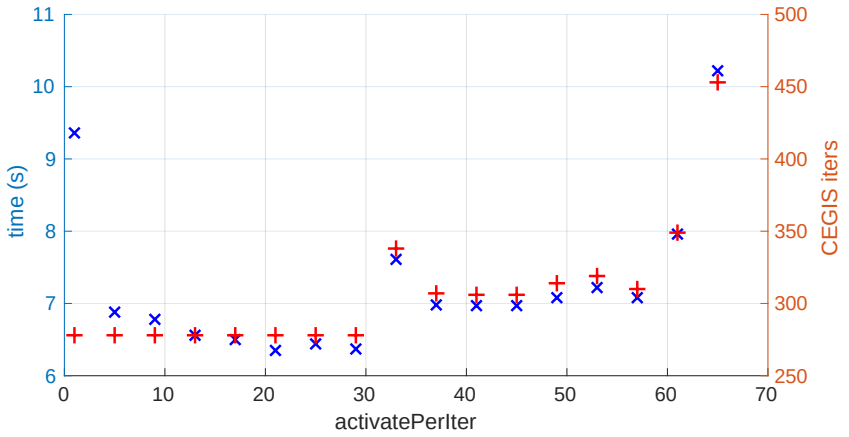


Figure 6.3: Caption

λ	CEGIS	CEGAR	Integrated
0.01	19.09	111.44	6.7
0.03	7.87	9.6	4.31
0.05	7.7	9.59	4.4
0.07	5.79	6.75	3.21
0.09	<1	5.22	<1
0.11	<1	5.34	<1

Grid model, safety property, $\lambda_f = 0.02$.

λ	CEGIS	CEGAR	Integrated
0.89	<1	16.81	<1
0.91	<1	16.69	<1
0.93	<1	16.66	<1
0.95	9.09	56.92	5.1
0.97	8.65	54.39	5.14
0.99	21.1	298.95	10.64

Grid model, liveness property, $\lambda_f = 0.98$.

Figure 6.4: Performance evaluation (execution time, s) for *Grid* model.

formance of the methods depends on the bound λ for values in the vicinity of the feasibility threshold λ_f, that is, we consider the hardest feasibility problems (problems with λ too low or too high are usually easily solvable). For each method, we report the execution time, in seconds. Measurements under one second are displayed as '<1', measurements with ten-minutes timeout are displayed as '>600'. The integrated method is executed with 'optimal' (wrt. a given model and a given property) values. For the exact values of obtained measurements, as well as for the values of parameters of the integrated method (for the purposes of reproducibility), please refer to the report file in the attached medium. The results are reported in Tables 6.4 through 6.7. Let us try to process the presented numbers and interpret them in a meaningful way.

- *Grid.* In our case, safety and liveness properties for the *Grid* model are dual ($\mathbb{P}_{\leq\lambda}[\mathrm{F}\ T] \equiv \mathbb{P}_{>1-\lambda}[\mathrm{F}\ S \backslash T]$), which is reflected in the behavior of synthesis methods (discrepancy in the absolute values between safety and liveness properties are due to the difference in the state space of family members, see the description at the beginning of this chapter). This experiment is designed to showcase that the integrated method handles liveness properties with the same ease as safety properties, also confirming that our intuition to choose $\min \boldsymbol{\upsilon}$ during activation with liveness properties is correct. For all values of λ, the integrated method is at least as good CEGIS; in hard cases ($\lambda = 0.01$ for safety or $\lambda = 0.99$ for liveness), it is at least twice as good as CEGIS and at least 15x as good as CEGAR.

λ	CEGIS	CEGAR	Integrated
0.73	>600	<1	<1
0.75	>600	9.77	5.1
0.77	>600	9.16	5.15
0.79	<1	7.38	<1
0.81	<1	5.16	<1
0.83	<1	<1	<1

Pole model, safety property, $\lambda_f = 0.74$.

λ	CEGIS	CEGAR	Integrated
0.25	<1	<1	<1
0.27	<1	3.97	<1
0.29	<1	5.2	<1
0.31	>600	7.55	3.99
0.33	>600	8.38	2.92
0.35	>600	<1	<1

Pole model, liveness property, $\lambda_f = 0.34$.

Figure 6.5: Performance evaluation (execution time, s) for *Pole* model.

λ	CEGIS	CEGAR	Integrated
0.16	<1	<1	<1
0.18	25.48	10.79	2.09
0.20	25.5	10.8	1.14
0.22	43.09	10.91	1.15
0.24	39.65	10.82	1.17
0.26	1.54	10.87	1.16

Maze model, safety property, $\lambda_f = 0.17$.

λ	CEGIS	CEGAR	Integrated
0.74	1.49	94.86	1.67
0.76	79.27	10.2	1.63
0.78	106.13	10.42	4.65
0.80	82.9	10.33	1.5
0.82	566.82	10.52	2.27
0.84	5.75	<1	<1

Maze model, liveness property, $\lambda_f = 0.83$.

Figure 6.6: Performance evaluation (execution time, s) for *Maze* model.

λ	CEGIS	CEGAR	Integrated
0.003	1.28	<1	<1
0.005	52.74	<1	<1
0.007	>600	<1	<1
0.009	>600	32.14	1.08
0.011	2.1	31.67	1.02
0.013	2.09	31.85	1.01

DPM model, safety property, $\lambda_f = 0.008$.

λ	CEGIS	CEGAR	Integrated
0.52	46.01	77.77	2.34
0.54	45.23	77.53	2.37
0.56	45.4	79.76	2.3
0.58	>600	263.23	3.36
0.60	>600	260.27	3.33
0.62	>600	276.14	3.31

Herman model, liveness property, $\lambda_f = 0.57$.

Figure 6.7: Performance evaluation (execution time, s) for *DPM* and *Herman* models.

- *Pole.* This is an example where the program contains a considerable amount of commands (as well as holes) and, additionally, the average size of a family member is quite high (around 14k states). Therefore, high-level CEGIS does not perform too good unless it is lucky to find a feasible solution early; meanwhile, CEGAR performs fine. The integrated method does not have any issues the CEGIS approach experiences, and in all cases performs almost twice as good as CEGAR.

- *Maze.* This is a model with a moderate number of family members of moderate size. CEGAR clearly outperforms CEGIS here. Meanwhile, in non-trivial cases, the integrated approach performs five (liveness) to ten (safety) times better than CEGAR, and around 80 (liveness) to 25 (safety) better than CEGIS.

- *DPM.* The last two models address the scalability of the synthesis methods. Recall, from the beginning of this chapter, that *DPM* is a family with 43M members, with the corresponding quotient MDP having 27k states. We see that CEGIS easily handles cases with high or low values of threshold λ, but really struggles near the feasibility threshold λ_f. On the other hand, CEGAR is able to reject all candidates in one MDP model checking for unfeasible cases $\lambda < \lambda_f$, but has serious problems handling feasible cases $\lambda > \lambda_f$ due to the size of the quotient MDPs. Meanwhile, the integrated method has absolutely no problems and is able to synthesize even the hardest cases in around one second. Here, the integrated method is making use of one MDP model checking in order to quickly prune the state space in about 3 iterations, as compared to 11 for pure CEGIS.

- *Herman.* Finally, *Herman* is a model with an even larger quotient MDP (around 54k states), although the total number of members is not that high. Nevertheless, both CEGIS and CEGAR struggle with it, especially for unfeasible properties. However, the integrated method computes all of the cases without particular effort, in a matter of a couple of seconds, demonstrating relative acceleration around 20x (feasible properties, compared to CEGIS) and up to 80x (unfeasible properties, compared to CEGAR).

Several conclusions are in order. First, we have never encountered a case study where the integrated method would perform worse than either of the existing methods. The highlights of our experiment evaluation were cases where one of the methods – CEGIS or CEGAR – is preferred to solve feasible problems and the other performs better for unfeasible cases: we have witnessed that the integrated technique inherits the strong sides of both methods and is capable of handling any model and any property without any performance decrease compared to pure approaches. In fact, in most cases, it performs *significantly* better than both CEGIS and CEGAR. Finally, if we scale the model one or two orders of magnitude up, we have witnessed that, while state-of-the-art methods struggle to compute the solution within a meaningful time frame, the novel integrated approach manages to sail through any of the problems it is faced with and manages to produce a sound result in a matter of seconds.

Chapter 7

Final Considerations

7.1 Future Research

Although the designed integrated synthesis method demonstrates remarkable results, it still offers a considerable space for improvement. First, it would be desirable to introduce a self-tuning loop where a method would adapt its parameters $mdpMClimit$ and $activatePerIter$ to a given model. The main idea is simple: we begin with some small values for both parameters and start a CEGAR loop. After refining the subfamilies for some time – as dictated by $mdpMClimit$ – we initiate CEGIS phase, where we monitor the quality of computed counterexamples: if provided counterexamples are too large (indicated by the number of states, number of conflicting commands or, preferably, number of conflicting holes) then we now that the provided bound on the reachability probability is too conservative, thus, we continue refining the subfamilies. Once we start constructing adequate counterexamples, we can try to increase the activation step $activatePerIter$, thus reducing the computational overhead, but probably leading to larger counterexamples, necessitating the execution of CEGAR phase again. Speaking of activation, it makes sense to look more rigorously into the activation strategy, i.e. in which order to activate states during counterexample construction. Although we brushed off this issue and picked a simple uninformed search based on the best criterium, we think it is still worth adopting ideas that gave rise to some advanced approaches, as in e.g. previously mentioned eXtended Best-First search [4] and the like. Finally, the method can significantly benefit from coarse-grained parallelization.

Regarding the functionality of the proposed method, we would like to introduce support for multiple properties, i.e. synthesizing a Markov chain wrt. multiple specifications. This is a feature that even CEGAR, in its current version, does not support, although the challenge is more implementational than theoretical. Another such implementational challenge is the support for parameter restrictions: given the domains for each of the parameters (holes), we wish to introduce additional constraints on mutually exclusive parameter combinations. Next, we wish to introduce support for reward properties or other PCTL specifications: this is again a purely implementational task, since we have already established the necessary theoretical background. Afterward, we wish to apply the novel approach for other synthesis problems, including threshold synthesis or optimal synthesis.

Finally, we would like to investigate more challenging questions beyond the scope of the existing synthesis framework. This includes, among others, adapting the ideas for state-space aggregation developed in [6], since model checking a Markov chain against reachability specifications remains a core stage of the synthesis pipeline. Other open problems include synthesizing infinite families of (infinite-state) Markov chains or deciding monotonicity of

parameters, which could not only assist in the process of synthesis, but could also provide sound feedback to the designer of a probabilistic system.

7.2 Conclusions

In this thesis, we considered the problem of automated synthesis of probabilistic systems. After recognizing the importance of having an efficient solution to this problem and identifying its key challenges: explosion of the underlying state space and the exponential growth of the family of candidate solutions, we first tackled the former problem and extended existing frameworks for aggregation-based analysis of Markov models by allowing them to handle chains with an arbitrary structure of the underlying state space and improve upon existing bounds on the approximation error. The designed framework was proved to be superior to existing approaches, providing the speedup of the analysis up to a factor of five with the corresponding approximation error bounded within 0.1%. Afterward, we considered the synthesis problem itself and, by adopting previously developed principles of counterexample-guided inductive synthesis (CEGIS) and abstraction refinement (CEGAR) we introduced a novel integrated synthesis algorithm. Experiments on practically relevant case studies demonstrated that the designed technique is not only comparable to state-of-the-art synthesis approaches, in most cases it manages to significantly outperform existing methods, sometimes by a margin of orders of magnitude.

Bibliography

[1] Abate, A., Brim, L., Češka, M. and Kwiatkowska, M. Adaptive aggregation of Markov chains: Quantitative analysis of chemical reaction networks. In: *Computer Aided Verification (CAV)*. Springer, 2015, p. 195–213.

[2] Ábrahám, E., Becker, B., Dehnert, C., Jansen, N., Katoen, J.-P. et al. Counterexample Generation for Discrete-Time Markov Models: An Introductory Survey. In: *Formal Methods for Executable Software Models: 14th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2014, Bertinoro, Italy, June 16-20, 2014, Advanced Lectures*. Cham: Springer International Publishing, 2014, p. 65–121. ISBN 978-3-319-07317-0.

[3] Aljazzar, H. and Leue, S. Directed Explicit State-Space Search in the Generation of Counterexamples for Stochastic Model Checking. *IEEE Transactions on Software Engineering*. 2010, vol. 36, no. 1, p. 37–60.

[4] Aljazzar, H., Leitner Fischer, F., Leue, S. and Simeonov, D. DiPro - A Tool for Probabilistic Counterexample Generation. In: Groce, A. and Musuvathi, M., ed. *Model Checking Software*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, p. 183–187. ISBN 978-3-642-22306-8.

[5] Alur, R., Bodik, R., Juniwal, G., Martin, M. M. K., Raghothaman, M. et al. Syntax-Guided Synthesis. In: *Proceedings of the IEEE International Conference on Formal Methods in Computer-Aided Design (FMCAD)*. October 2013, p. 1–17.

[6] Andriushchenko, R. *Approximate Techniques for Markov Models*. Brno, CZ, 2018. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Available at: https://www.fit.vut.cz/study/thesis/20512/.

[7] Arming, S., Bartocci, E., Chatterjee, K., Katoen, J.-P. and Sokolova, A. Parameter-Independent Strategies for pMDPs via POMDPs. In: McIver, A. and Horvath, A., ed. *Quantitative Evaluation of Systems*. Cham: Springer International Publishing, 2018, p. 53–70. ISBN 978-3-319-99154-2.

[8] Bartocci, E., Grosu, R., Katsaros, P., Ramakrishnan, C. R. and Smolka, S. A. Model Repair for Probabilistic Systems. In: *TACAS*. 2011.

[9] Benini, L., Bogliolo, A., Paleologo, G. A. and De Micheli, G. Policy optimization for dynamic power management. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*. 1999, vol. 18, no. 6, p. 813–833.

[10] Biere, A., Heule, M., Maaren, H. and Walsh, T. Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications. january 2009.

[11] Bornholt, J., Torlak, E., Grossman, D. and Ceze, L. Optimizing Synthesis with Metasketches. In: *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. New York, NY, USA: Association for Computing Machinery, 2016, p. 775–788. POPL '16. ISBN 9781450335492.

[12] Ceska, M., Dannenberg, F., Paoletti, N., Kwiatkowska, M. and Brim, L. Precise Parameter Synthesis for Stochastic Biochemical Systems. *Acta Informatica*. march 2016.

[13] Ceska, M., Hensel, C., Junges, S. and Katoen, J. Counterexample-Driven Synthesis for Probabilistic Program Sketches. *CoRR*. 2019, abs/1904.12371.

[14] Ceska, M., Jansen, N., Junges, S. and Katoen, J. Shepherding Hordes of Markov Chains. *CoRR*. 2019, abs/1902.05727.

[15] Češka, M., Šafránek, D., Dražan, S. and Brim, L. Robustness Analysis of Stochastic Biochemical Systems. *PLOS ONE*. Public Library of Science. april 2014, vol. 9, no. 4, p. 1–23.

[16] Chrszon, P., Dubslaff, C., Klüppelholz, S. and Baier, C. ProFeat: feature-oriented engineering for family-based probabilistic model checking. *Formal Aspects of Computing*. august 2017, vol. 30.

[17] Classen, A., Cordy, M., Heymans, P., Legay, A. and Schobbens, P.-Y. Model checking software product lines with SNIP. *International Journal on Software Tools for Technology Transfer*. 2012, vol. 14, p. 589–612.

[18] Classen, A., Cordy, M., Heymans, P., Legay, A. and Schobbens, P. Y. Formal semantics, modular specification, and symbolic verification of product-line behaviour. *Science of Computer Programming*. february 2014, vol. 80, p. 416–439.

[19] Didier, F., Henzinger, T. A., Mateescu, M. and Wolf, V. Fast Adaptive Uniformization of the Chemical Master Equation. In: *2009 International Workshop on High Performance Computational Systems Biology*. Oct 2009, p. 118–127.

[20] Es, I., Carwardine, J., Martin, T., Nicol, S., Sabbadin, R. et al. MOMDPs: a Solution for Modelling Adaptive Management Problems. *Twenty-Sixth AAAI Conference on Artificial Intelligence*. july 2012.

[21] Forejt, V., Kwiatkowska, M., Norman, G. and Parker, D. Automated Verification Techniques for Probabilistic Systems. In: *Formal Methods for Eternal Networked Software Systems: 11th International School on Formal Methods for the Design of Computer, Communication and Software Systems, SFM 2011, Bertinoro, Italy, June 13-18, 2011. Advanced Lectures*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, p. 53–113. ISBN 978-3-642-21455-4.

[22] Gerasimou, S., Tamburrelli, G. and Calinescu, R. Search-Based Synthesis of Probabilistic Models for Quality-of-Service Software Engineering (T). In: *2015 30th*

IEEE/ACM International Conference on Automated Software Engineering (ASE). Nov 2015, p. 319–330. ISSN null.

[23] Gerasimou, S., Calinescu, R. and Tamburrelli, G. Synthesis of Probabilistic Models for Quality-of-Service Software Engineering. *Automated Software Engineering*. april 2018.

[24] Gillespie, D. T. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of Computational Physics*. 1976, vol. 22, no. 4, p. 403 – 434. ISSN 0021-9991.

[25] Gillespie, D. T. Exact stochastic simulation of coupled chemical reactions. *The journal of Physical Chemistry*. ACS Publications. 1977, vol. 81, no. 25, p. 2340–2361.

[26] Haverkort, B., Hermanns, H. and Katoen, J.-P. On the Use of Model Checking Techniques for Dependability Evaluation. In: *Reliable Distributed Systems*. IEEE, 2000, p. 228–237.

[27] Hensel, C., Junges, S., Katoen, J.-P., Quatmann, T. and Volk, M. *The Probabilistic Model Checker Storm*. 2020.

[28] Herman, T. Probabilistic Self-Stabilization. *Inf. Process. Lett.* USA: Elsevier North-Holland, Inc. june 1990, vol. 35, no. 2, p. 63–67. ISSN 0020-0190.

[29] Junges, S., Jansen, N., Dehnert, C., Topcu, U. and Katoen, J. Safety-Constrained Reinforcement Learning for MDPs. *CoRR*. 2015, abs/1510.05880.

[30] Kaelbling, L. P., Littman, M. L. and Cassandra, A. R. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*. 1998, vol. 101, no. 1, p. 99 – 134. ISSN 0004-3702.

[31] Kierzek, A. M., Zaim, J. and Zielenkiewicz, P. The Effect of Transcription and Translation Initiation Frequencies on the Stochastic Fluctuations in Prokaryotic Gene Expression. *The Journal of Biological Chemistry*. 2001, vol. 276, no. 11, p. 8165–8172.

[32] Kwiatkowska, M., Norman, G. and Parker, D. Stochastic Model Checking. In: Bernardo, M. and Hillston, J., ed. *Formal Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation (SFM'07)*. Springer, 2007, vol. 4486, p. 220–270. LNCS (Tutorial Volume).

[33] Kwiatkowska, M., Norman, G. and Parker, D. PRISM 4.0: Verification of Probabilistic Real-time Systems. In: Gopalakrishnan, G. and Qadeer, S., ed. *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*. Springer, 2011, vol. 6806, p. 585–591. LNCS.

[34] Kwiatkowska, M., Norman, G. and Parker, D. Probabilistic Verification of Herman's Self-Stabilisation Algorithm. *Formal Aspects of Computing*. Springer. 2012, vol. 24, no. 4, p. 661–670.

[35] Lindemann, C. Performance Modelling with Deterministic and Stochastic Petri Nets. *SIGMETRICS Perform. Eval. Rev.* New York, NY, USA: Association for Computing Machinery. august 1998, vol. 26, no. 2, p. 3. ISSN 0163-5999.

[36] Meuleau, N., Kim, K., Kaelbling, L. P. and Cassandra, A. R. Solving POMDPs by Searching the Space of Finite Policies. *CoRR*. 2013, abs/1301.6720.

[37] Norman, G., Parker, D. and Zou, X. Verification and Control of Partially Observable Probabilistic Real-Time Systems. *CoRR*. 2015, abs/1506.06419.

[38] Pathak, S., Ábrahám, E., Jansen, N., Tacchella, A. and Katoen, J.-P. A Greedy Approach for the Efficient Repair of Stochastic Models. In:. April 2015.

[39] Quatmann, T., Dehnert, C., Jansen, N., Junges, S. and Katoen, J. Parameter Synthesis for Markov Models: Faster Than Ever. *CoRR*. 2016, abs/1602.05113.

[40] Rodrigues, G. N., Alves, V., Nunes, V., Lanna, A., Cordy, M. et al. Modeling and Verification for Probabilistic Properties in Software Product Lines. In: *2015 IEEE 16th International Symposium on High Assurance Systems Engineering*. Jan 2015, p. 173–180. ISSN 1530-2059.

[41] Solar Lezama, A. *Program Synthesis by Sketching*. USA, 2008. Dissertation. ISBN 9781109097450.

[42] T Gillespie, D. Stochastic Simulation of Chemical Kinetics. february 2007, vol. 58, p. 35–55.

[43] Wimmer, R., Jansen, N., Vorpahl, A., Ábrahám, E., Katoen, J. et al. High-level Counterexamples for Probabilistic Automata. *Logical Methods in Computer Science*. 2015, vol. 11, no. 1.