BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INTELLIGENT SYSTEMS
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

ADVANCED METHODS FOR SYNTHESIS OF PROBABILISTIC PROGRAMS

POKROČILÉ METÓDY PRE SYNTÉZU PRAVDEPODOBNOSTNÝCH PROGRAMOV

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR **Bc. ŠIMON STUPINSKÝ**
AUTOR PRÁCE

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

BRNO 2021

Brno University of Technology
Faculty of Information Technology

Department of Intelligent Systems (DITS) Academic year 2020/2021

Master's Thesis Specification

Student: **Stupinský Šimon, Bc.**
Programme: Information Technology
Field of study: Software Verification and Testing
Title: **Advanced Methods for Synthesis of Probabilistic Programs**
Category: Formal Verification
Assignment:

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

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.**
Consultant: Andriushchenko Roman, Ing., UITS FIT VUT
Head of Department: Hanáček Petr, doc. Dr. Ing.
Beginning of work: November 1, 2020
Submission deadline: May 19, 2021
Approval date: November 11, 2020

Abstract

Probabilistic programs play a crucial role in various engineering domains, including computer networks, embedded systems, power management policies, or software product lines. PAYNT is a tool for the automatic synthesis of probabilistic programs satisfying the given specifications. In this thesis, we extend this tool primarily to support optimal synthesis and synthesis for multi-property specifications. Further, we have proposed and implemented a new method that can efficiently synthesise continuous parameters affecting the transition probabilities alongside the synthesis of a program topology, i.e., support of both topology and parameter synthesis at the same time. We demonstrate the usefulness and performance of PAYNT on a wide range of real-world case studies from various application domains. For challenging synthesis problems, PAYNT can significantly decrease the run-time from days to minutes while ensuring the completeness of the synthesis process.

Abstrakt

Pravdepodobnostné programy zohrávajú rozhodujúcu úlohu v rôznych technických doménach, ako napríklad počítačové siete, vstavané systémy, stratégie riadenia spotreby energie alebo softvérové produčkné linky. PAYNT je nástroj na automatizovanú syntézu pravdepodobnostných programov vyhovujúcich zadaným špecifikáciam. V tejto práci rozširujeme tento nástroj predovšetkým o podporu optimálnej syntézy a syntézy viacerých špecifikácií. Ďalej sme navrhli a implementovali novú metódu, ktorá dokáže efektívne syntetizovať parametre so spojitým definičným oborom ovplyvňujúce pravdepodobnostné prechody popri syntéze topológie programov, t.j., podporu pre syntézu topológie aj parametrov súčasne. Demonštrujeme užitočnosť a výkonnosť nástroja PAYNT na širokej škále prípadových štúdií z rôznych aplikačných domén ktoré majú uplatnenie v reálnom svete. Pri náročných problémoch syntézy môže PAYNT výrazne znížiť dobu behu až z dní na minúty a zároveň zaistiť úplnosť procesu syntézy.

Keywords

automated synthesis, probabilistic programs, Markov models, model checking

Kľúčové slová

automatizovaná syntéza, pravdepododnostné programy, Markovské modely, model checking

Reference

STUPINSKÝ, Šimon. *Advanced Methods for Synthesis of Probabilistic Programs.* Brno, 2021. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor RNDr. Milan Češka, Ph.D.

Rozšírený abstrakt

Náhodnosť je kľúčová pre výskumné oblasti, ako sú napríklad pravdepodobnostné programovanie, spoľahlivosť (systémové komponenty s neistotou), distribuované výpočty (porušenie symetrie), a plánovanie (neznáme a neisté prostredia). Pravdepodobnostné programy sú výkonným modelovacím nástrojom pre systémy obsahujúce pravdepodobnostnú neistotu. Systémy s nespoľahlivým a nepredvídateľným správaním si vyžadujú použitie takéhoto matematického aparátu založeného na teórii pravdepodobnosti. Návrh takýchto systémov vykazujúcich žiadúce správanie – napr. výber optimálnej stratégie riadenia spotreby energie alebo výber sieťového protokolu zvyšujúceho priepustnosť paketov – je náročný z hľadiska uvažovania o viacerých alternatívnych prevedeniach daného systému. Aplikácie takýchto systémov pokrývajú širokú škálu výskumných oblastí, vrátane analýzy (kvantitatívnych) softvérových produktových liniek [35, 17], syntézy stratégie pri plánovaní za čiastočnej pozorovateľnosti [23, 40] alebo návrh komunikačných protokolov [19, 24].

Sada deklaratívnych časových obmedzení často vyjadruje účinnosť a správnosť pravdepodobnostných programov. Nástroje pre kontrolu modelov pravdepodobnostných systémov, ako Storm [16] alebo Prism [25], vykonávajú automatizované overenie takýchto obmedzení. Tieto nástroje pre kontrolu pravdepodobnostných modelov však vyžadujú model alebo systém s pevnou topológiou, na rozdiel od požiadaviek na použitie týchto nástrojov pri modelovaní pravdepodobnostných programov. Vývojári by mali čo najskôr počas procesu vývoja systému overiť jeho návrhy, aby náklady na vývoj boli čím lacnejšie a udržateľnejšie. Návrh systému je v počiatočných fázach vývoja prevažne neúplný, pretože vo väčšine prípadov nie sú známe všetky špecifikácie systému alebo sú zámerne vynechané. Tieto nedefinované špecifikácie systému sa nazývajú diery a môžu napríklad odrážať nešpecifikovanú komponentu alebo čiastočne implementovaný radič. Primárnym účelom syntézy je odhaliť konkrétny (optimálny) subsystém s plne definovaným správaním. Zásadným aspektom návrhového cyklu je prieskum množiny návrhov, t.j. preskúmanie všetkých možných návrhov uvažovaného systému. Keď považujeme Markovský reťazec za matematický aparát pravdepodobnostného programu, potom návrhový priestor predstavuje rodinu takýchto reťazcov a úlohou syntézy je nájsť ten reťazec, ktorý spĺňa dané špecifikácie.

Typicky sa syntéza programu považuje za problém pri deduktívnom dokazovaní, ktoré odvodzuje program, ktorý vyhovuje zadaným špecifikáciam, od konštruktívneho dôkazu [27]. V poslednej dobe prichádza do popredia alternatívny prístup k syntéze, keď návrhári, okrem špecifikácie, poskytujú aj syntaktickú šablónu pre požadovaný program. Uvažujeme o prístupe využívajúcom náčrt systému, keď návrhár skonštruuje čiastkový program s neúplnými podrobnosťami a syntetizátor doplní chýbajúce podrobnosti oproti uvedeným špecifikáciám [36]. Ďalej môžeme problémy syntézy pre parametrické pravdepodobnostné systémy rozdeliť do dvoch kategórií. Najskôr zvážime úlohu syntézy topológie za predpokladu konečnej sady parametrov, ktoré ovplyvňujú topológiu modelu, kde jednotlivé parametre predstavujú nedefinované špecifikácie systému. Táto úloha sa zameriava na rodiny Markovských reťazcov, ktoré majú rozdielne topológie stavového priestoru a v dôsledku toho rôzne množiny dosiahnuteľných stavov. Druhá oblasť úloh syntézy uvažuje však o Markovskom reťazci s pevnou topológiou, ale nedefinovanými pravdepodobnosťami prechodu v rámci modelu.

Táto oblasť bola diskutovaná prístupmi pre opravu modeliov [4, 32] a technikami pre *syntézu parametrov* [10, 34]. Prístup „one-by-one“ môže naivne riešiť problém syntézy topológie analýzou všetkých jedinečných návrhov [11, 12]. Naopak, celý priestor návrhov je možné modelovať aj ako jediný Markovský rozhodovací proces typu „all-in-one“ [11, 13]. Vymenovanie všetkých členov návrhového priestoru (realizácie) je však nerealizovateľné kvôli

kombinatorickej explózii tohto priestoru a veľkosť uvažovaného Markovského rozhodovacieho procesu typu „all-in-one“ je úmerná počtu kandidátnych návrhov. Bohužiaľ, problém s týmto dvojitým výbuchom v stavovom priestore spôsobuje, že oba tieto prístupy sú pre veľké rodiny nerealizovateľné. Iné prístupy uvažujú o evolučných vyhľadávacích algoritmoch pre syntézu softvérových systémov [17]. Tieto metódy zostávajú však neúplné a nemôžu efektívne vyriešiť náročnejšie úlohy, napr. ktorý dizajn *optimálne* vyhovuje zadanej špecifikácii.

Pri modelovaní systémov v reálnom svete môže rýchlo nastať kombinácia problémov topológie aj syntézy parametrov, ale podpora riešenia takejto kombinovanej úlohy syntézy neexistuje. Táto diplomová práca sa zameriava na vývoj nástroja PAYNT, ktorý rozširuje možnosti syntézy, najmä o podporu kombinovanej syntézy. PAYNT je schopný efektívne syntetizovať topológiu programu aj parametre so spojitým definičným oborom ovplyvňujúce pravdepodobnosť prechodu – čo predstavuje jedinečná vlastnosť. Ďalej sa zameriavame na úplné najmodernejšie prístupy k syntéze pravdepodobnostných programov. Prvý prístup analyzuje každý návrh z danej podskupiny kandidatných riešení jednotlivo a konštruuje kritické podsystémy protipríkladov s cieľom vylúčiť čo najväčší počet možných návrhov, ktoré nespĺňajú danú špecifikáciu - tento prístup sa nazýva protipríkladmi riadená iduktívna syntéza (CEGIS) [8]. Druhý prístup, ktorý nazývame zjemňovanie abstrakcie (AR) [9], analyzuje celý návrhový priestor a pretvára ho do podrodín návrhov, ak analýza prinesie nepresvedčivé výsledky. Oba tieto prístupy preukázali presvedčivé výsledky, no aj napriek tomu každý z nich čelí určitým obmedzeniam. Navyše sú neporovnateľné, pretože jeden prístup môže byť vhodnejší pre istú skupinu pravdepodobnostných programov a naopak. Preto sa zameriavame na prístup predstavený v diplomovej práci Andriushchenka [1], po ktorom nasleduje jeho zdokonalenie v práci [2], ktorý kombinuje tieto sofistikované metódy poskytujúce analýzu celých podrodín návrhov naraz. Tento prístup dokáže výrazne prekonať prezentované metódy, niekedy až rádovo o niekoľko stupňov.

Advanced Methods for Synthesis of Probabilistic Programs

Declaration

Hereby I declare that this master's thesis was prepared as an original author's work under the supervision of RNDr. Milan Češka, Ph.D. The supplementary information was provided by consultant Ing. Roman Andriuschenko. All the relevant information sources, which were used during preparation of this thesis, are properly cited and included in the list of references.

.....
Šimon Stupinský
May 25, 2021

Acknowledgements

I would like to express my sincere gratitude to my supervisor, RNDr. Milan Češka, Ph.D., for his guidance and continuous support throughout my master's studies, and Ing. Roman Andriuschenko for his advices and cooperation on the development of our work. Last but not least, a sincere thanks goes to my girlfriend Dominika, family and friends for their never-ending support and encouragement during the inevitable hard times.

Contents

Chapter 1

Introduction

> *"The first rule of any technology used in a business is that automation applied to an efficient operation will magnify the efficiency. The second is that automation applied to an inefficient operation will magnify the inefficiency."*
>
> – Bill Gates

Randomisation is essential to research areas such as *probabilistic programming*, dependability (system components with uncertainty), distributed computing (symmetry breaking), and planning (unknown and noisy environments). Probabilistic programs are powerful modelling apparatus of systems containing probabilistic uncertainty. Systems with unreliable and unpredictable behaviour require the use of such a mathematical apparatus established on probability theory. Designing such systems exhibiting a desirable behaviour – e.g. selecting the optimal power management strategy or a network protocol increasing the packet throughput – is challenging for reasoning over multiple alternative designs. Their applications cover a broad range of research areas, including, e.g. analysis of (quantitative) software product lines [35, 17], strategy synthesis in planning under partial observability [23, 40], or design of communication protocols [19, 24].

A set of declarative temporal constraints often expresses the efficiency and correctness of the probabilistic programs. The model checkers for probabilistic systems, such as Storm [16] or Prism [25], provide automated verification of such constraints. However, these probabilistic model checkers require a fixed model or a fixed program, contrary to their usage requirements when modelling probabilistic programs. Developers need to verify the system designs as early as possible at the developing process to maintain its costs tractable and as best as possible. System designs are prevailingly incomplete at the initial development phases because, in most cases, there are no known all system specifications or intentionally left out potentially. These undefined system specifications are called *holes*, and they can, e.g., reflect an unspecified component or a partially implemented controller. The synthesis primary purpose is to reveal a concrete (optimal) subsystem with fully defined behaviour. A vital aspect of the design cycle is design space explorations, i.e. exploring all possible designs. When considering a Markov chain (MC) as the mathematical apparatus of a probabilistic program, then the design space represents a family of such chains, and the synthesis task is to find the one that satisfies a given specifications.

Classically, program synthesis is considered a problem in deductive theorem proving, which derives a program holding the desired specifications from the constructive proof of the theorem [27]. An alternative approach to synthesis has recently come to the fore where the designers, except the specification, provide a syntactic template for the desired

program. We consider an approach using the *sketch* of the system, where a designer constructs a partial program with incomplete details, and the synthesizer fills in the missing details against the given specifications [36]. Further, the synthesis problems for parametric probabilistic systems can be divided into two categories. First consider *topology synthesis* task assuming a finite set of parameters that affect the model topology, where the individual parameters represent the undefined system specifications. This task focuses on Markov chains families having different topologies of the state space and, as a consequence, different sets of reachable states. However, another area of synthesis tasks considers a Markov chain with fixed topology but undefined transition probabilities.

This area has been discussed by approaches of model repairing [4, 32] and techniques for *parameter synthesis* [10, 34]. A *one-by-one* approach can naively solve the topology synthesis problem by analysing all unique designs [11, 12]. On the contrary, the whole design space can also be modelled as a single *all-in-one* Markov decision process [11, 13]. However, enumerating all members of design space (realisations) is unfeasible due to its combinatorial explosion, and the size of such all-in-one MDP is proportional to the number of candidate designs. Unfortunately, the double state-space explosion problem renders both of these approaches infeasible for large families. Other approaches consider evolutionary search algorithms for the synthesis of software systems [17]. These methods remain incomplete and cannot efficiently solve more challenging systems, e.g., which design satisfies the specification *optimally*.

When modelling real-world systems, combining both topology and parameter synthesis problems can quickly occur, but the support to solve such a *combined synthesis* task does not exist. This thesis focus on the development of a tool PAYNT that expands the possibilities of synthesis, especially combined synthesis. PAYNT is able to efficiently synthesise the topology of the program as well as continuous parameters affecting the transition probabilities – this is a unique feature. In further, we will focus on complete state-of-the-art approaches for the synthesis of probabilistic programs.

The first approach analyses each design from a given sub-family individually and constructs critical sub-systems of counter-examples to prune all designs behaving incorrectly – the so-called *counter-example guided inductive synthesis* (CEGIS) [8]. The second approach, called abstraction refinement (AR) [9], immediately analyses the entire design space and refines it into design sub-families when the analysis yields inconclusive results. Both of these approaches have shown convincing results, although each faces certain limitations. They are incomparable because one approach can be more suitable for specific probabilistic programs classes and conversely. Therefore, we focus on approach introduced in *Andriushchenko* master's thesis [1], followed by its improvement in [2], which combines these sophisticated methods providing an analysis of whole design sub-families at once. It manages to significantly outperform them, sometimes by a margin of orders of magnitude.

Key Contributions.

This thesis considers a novel integrated method introduced in the previous works [1, 2] as a cornerstone for the topology synthesis. Initially, this method was designed only for feasibility synthesis task with one specification. However, probabilistic programs often have to satisfy specifications expressed as a conjunction of several temporal logic constraints. Therefore, we designed an extension of this method to support *multi-property specifications* and *optimal synthesis* task. The designed extension for multi-properties is performed in the same loop as the origin single-property synthesis, with necessary modifications of in-

dividual methods. Consequently, the novel integrated method inherits the benefits of *AR* and *CEGIS* in its favour also at multi-property synthesis. *Optimal synthesis* is a particular instance of *multi-property synthesis*, and it can find an application in various domains. In particular, specification set includes so-called violation property representing the currently optimal solution, and its threshold is updated whenever a new optimal solution is found. Moreover, we designed support for the relaxed variant of the optimal synthesis, so-called ε-optimal synthesis, which is in most cases even faster. We evaluate designed extensions on an extensive set of real-world case studies.

Furthermore, we design a method to perform a *combined synthesis* problem, which supports both *topology* and *parameter* synthesis. This method is based mainly on a conceptually simple technique for verifying parametric probabilistic models introduced in [34] and on a novel integrated method for topology synthesis. It uses advances of both approaches to create an innovative method, which is further complemented by new concepts. Thus, this method represents a non-trivial combination of these two approaches initially oriented specifically to one kind of synthesis problem. This method is unique among existing ones because it can efficiently synthesise the program's topology and continuous parameters affecting the transition probabilities. Our tool paper presenting PAYNT has been recently accepted at CAV'21, an A* conference. Moreover, we presented PAYNT also at the students' conference Excel@FIT'21, where we got the award from the expert panel to develop of the tool that significantly expands the possibilities of designing probabilistic systems. At this conference, we also won an award from the professional public.

Structure of this paper.

Chapter 2 summarises the necessary theory regarding parametric probabilistic models and formulates a probabilistic synthesis problem. Chapter 3 advocates the probabilistic synthesis in the broader frame of automated system development and introduces a state-of-the-art novel integrated method based on two modern approaches *CEGIS* and *AR*. Chapter 4 develops vital ideas associated with integrating the *multi-property* synthesis and *optimal* synthesis within the presented integrated method. Further, in this chapter, we develop critical ideas associated with integrating the combined synthesis – consists of topology and parameter synthesis – within the considered method. Chapter 5 introduces a new tool called PAYNT and its architecture, which implemented the presented methods. Chapter 6 evaluates our solutions on a broad range set of real-world case studies and compares them with the baseline enumeration approach. Finally, Chapter 7 closes this thesis with the notes and issues that can serve as a baseline point for the follow-up research and potential improvement of designed solutions.

Chapter 2

Probabilistic Programs

This section formalises necessary ingredients and the problem statement for probabilistic synthesis. We will first introduce a discrete-time Markov chain as the simplest operation model for probabilistic programs. Further, we will describe a Markov decision process representing a similar model as a Markov chain, but it contains additional non-determinism. These models play a crucial role when synthesising probabilistic programs, as we will see later in Chapter 3. The following definitions are inspired from the existing sources, mainly from [2, 34], where a more detailed description can also be found.

2.1 Probabilistic Models

We denote a finite set of *parameters* as V. We consider *parameters* over the domain $\mathbb{R}$ ranged over by x, y, z for the following definitions. Let $u : V \to \mathbb{R}$ denote a *valuation* for *parameters* V. We denote a set of *multi-affine multivariate polynomials* as $\mathbb{Q}_V$, where polynomial f is defined over parameters V and equal to $\sum_{i \leq M} \prod_{v \in V_i} v \cdot a_i$, for appropriate $\mathcal{M} \in \mathbb{N}_0$, $a_i \in \mathbb{Q}$, and $\forall\, 0 \leq i \leq m.\ V_i \subseteq V$. We note that this set includes only the polynomials where the variables have the maximal degree equal to one, i.e., $y^3 \notin \mathbb{Q}_V$, but $y \cdot z \in \mathbb{Q}_V$. When we apply the valuation u to polynomial $f \in \mathbb{Q}_V$, the resulting instance yields a real number $f[u] \in \mathbb{R}$, in which all occurrences of variable $x \in f$ are replaced by $u(x)$. We consider various types of (parametric) discrete probabilistic models. We can look at them as the transition systems with the given state space where the transitions are marked with polynomials in $\mathbb{Q}_V$ [34].

Definition 1 (pMDP)**.** A parametric Markov decision process (pMDP) $\mathcal{M}$ is a tuple $(S, V, s_0, Act, \mathcal{P})$, where S is a finite set of states, V is a finite set of parameters over $\mathbb{R}$, $s_0 \in S$ is an initial state, Act is a non-empty finite set of actions, and $\mathcal{P} : S \times Act \times S \nrightarrow \mathbb{Q}_V$ is a transition function.

A set $Act(s) = \{a \ \in Act \mid \exists s' \in S.\mathcal{P}(s, a, s') \neq \bot\}$ for state $s \in S$ represents the *available* actions. The model do not contains deadlock states, since for each state $s \in S$ holds that the action set $Act(s)$ is non-empty. When holds $|Act(s)| = 1$ for each state $s \in S$, such pMDP straightforward induces an parametric Markov chain (pMC), which has enabled only one action at each state.

Example 1 (Parametric models)**.** We depict (a) pMDP and (b) pMC with parameters $\{x, y\}$ in Figure 2.1. We mark an initial state $s_0 \in S$ with an arrow, and the target state $s_3 \in S$ with double lines. We label the transition from state $s \in S$ to $s' \in S$ by action α and corresponding $\mathcal{P}(s, \alpha, s')$, if such transition is executable.

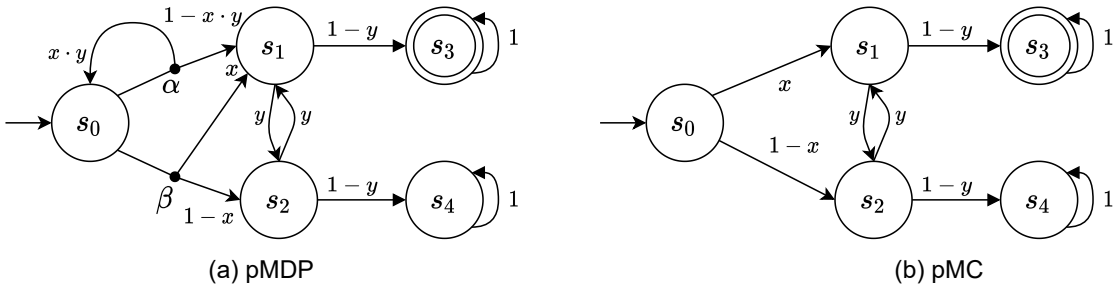


Figure 2.1: The introduced kinds of parametric probabilistic models – pMDP and pMC.

Definition 2 (Distribution)**.** [2] A *discrete* distribution over a finite or countably infinite set X is a function $\mu : X \to [0, 1]$ s.t. $\sum_{x \in X} \mu(x) = \mu(X) = 1$. The set of all distributions on X is denoted $Distr(X)$. The support of a distribution μ is $supp(\mu) = \{x \in X \mid \mu(x) > 0\}$.

Example 2 (Parametric probabilistic models)**.** Let $X = \{x_0, x_1, x_0\}$ be a finite set. Let function $\mu : X \to [0, 1]$ defined as $\mu : [x_0 \mapsto \frac{1}{2}, x_1 \mapsto 0, x_2 \mapsto \frac{1}{2}]$ be a *probability distribution* on X, i.e. $\mu \in Distr(X)$. The support of μ is $supp(\mu) = \{x_0, x_2\}$, and for simplification, we writes such distributions as $\mu = \frac{1}{2} : x_0 + \frac{1}{2} : x_2$.

Definition 3 (MDP)**.** A pMDP $\mathcal{M}$ is Markov decision process (MDP) $\mathcal{M}$ if $\mathcal{P} : S \times Act \times S \to Distr(S)$, so for each state $s \in S$ and each its action $\alpha \in Act(s)$ holds that $\sum_{s' \in S} \mathcal{P}(s, \alpha, s') = 1$.

We can define in the same way a MCs as the special instance of pMCs. We say, that the model is *parameter-free* when for each its transition probability holds, that it contains only constant values and not undefined parameters. When we apply the valuation u to the parametric model $\mathcal{M}$, then we obtain the instantiation of $\mathcal{M}$ at u ($\mathcal{M}[u]$), where each polynomial $f \in \mathcal{M}$ is replaced by $f[u]$. We use the valuation u typically to substitute the transition function f by the concrete probability $f[u]$. When the instantiated model $\mathcal{M}[u]$ yields an MDP or MC, we denote the valuation u as *well-defined* since it returns the probability distributions.

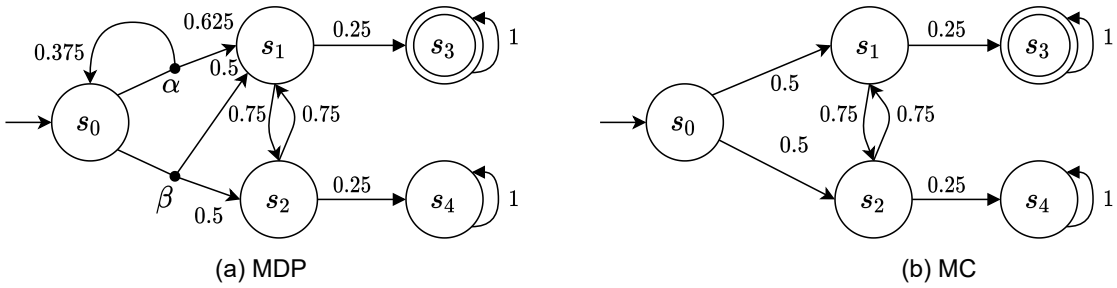


Figure 2.2: The introduced kinds of probabilistic models – MDP and MC.

Example 3 (Probabilistic models)**.** Figure 2.1 depicts an pMDP and an pMC. Let us consider the following valuation $u : \{x \mapsto 0.5, y \mapsto 0.75\}$. By applying this valuation to an pMDP results in parameter-free MDP depicted in Figure 2.2, and the same in the case of MC instantiated from relevant pMC.

Instinctively, we can imagine an MC as a state-transition system with the following semantics. A probability distribution for each state $\forall s \in S : \mathcal{P}(S)$ represents a stochastic choice of firing the transition from such state $s \in S$ to one of its successors states $s' \in supp(\mathcal{P}(\mathcal{S}))$. Consequently, an MC defined with such semantics has a *Markov property* (memorylessness), which is essential when modelling systems and for efficient analysis. This property declares that the probability of the transition from state $s \in S$ to state $s' \in supp(\mathcal{P}(\mathcal{S}))$ depends only on the current state, and it is independent of the taken path of chain to state s. We can see that each state of an MC disposes of a unique probability distribution over its possible successor states. MDPs define an extension of MCs introducing a non-deterministic choice between several probability distributions over each state.

A (in)finite sequence $\pi = s_0 \overset{a_0}{\to} s_1 \overset{a_1}{\to} \ldots$, where $s_i \in S$, $a_i \in Act(s_i)$, and $\mathbb{P}(s_i, a_i)(s_{i+1}) \neq 0$ for all $i \in \mathbb{N}$ represents the *path* of an MDP M. For finite π, $last(\pi)$ denotes the last state of π, and the set of (in)finite paths of M we denotes as $Paths^M_{fin}$ ($Paths^M$). When an MDP is currently in the state $s \in S$, it has a non-deterministic choice of an action $a \in Act(s)$ leading to the one possible probability distribution $\mathcal{P}(s)(a)$ over its successors' states. These actions cause the non-deterministic behaviour of an MDP. Still, this property can be suppressed by applying a *scheduler*, which selects one specific action in each state, i.e., transforms an MDP to an MC.

Definition 4 (Scheduler)**.** A scheduler for an pMDP $\mathcal{M} = (S, V, s_0, Act, \mathcal{P})$ is a function $\sigma : Paths^M_{fin} \to Act$ such that $\sigma(\pi) \in Act(last(\pi))$ for all $\pi \in Paths^M_{fin}$. Scheduler σ is memory-less if $last(\pi) = last(\pi') \implies \sigma(\pi) = \sigma(\pi')$ for all $\pi, \pi' \in Paths^M_{fin}$. The set of all schedulers of M is Σ^M.

When we apply a scheduler to an pMDP, then we obtain an induced parametric Markov chain, which does not contain any non-determinism. In other words, the transition probabilities are obtained concerning the choice of actions.

Definition 5 (Induced pMC)**.** [9] An pMC induced by pMDP $\mathcal{M} = (S, V, s_0, Act, \mathcal{P})$ and $\sigma \in \Sigma^M$ is defined by $\mathcal{M}_\sigma = (Paths^M_{fin}, s_0, \mathbf{P}^\sigma)$ where:

$$\mathbf{P}^\sigma(\pi, \pi') = \begin{cases} \mathcal{P}(last(\pi), \sigma(\pi))(s') & if\ \pi' = \pi \overset{\sigma(\pi)}{\to} s' \\ 0 & otherwise. \end{cases}$$

2.2 Region Model Checking

When considering the model checking of pMCs at parameter synthesis problem, we have to analyse the whole set of MCs representing the individual parameters substitution. For this purpose, we introduce the region model checking, a technique to practical analyses such sets of MCs. We consider valuation sets that mapping parameter to the value within a specified interval. We introduce a technique for approximate checking pMC instances concerning the valuation in such a set. Firstly, we introduce definitions for *regions* and the considered task of model checking. Subsequently, we define the construction of *over-approximation*, which will be subsequently *reduce* to an MDP problem. The definitions in this section are inspired mainly by [34].

2.2.1 Regions

Definition 6 (Region)**.** Let $V = \{v_1, v_2, \ldots, v_n\}$ be a set of parameters and $\forall\, v_i \in V.\ \mathcal{B}(v_i) = \{b_1, b_2\}$ rational bounds for each parameter v_i. These parameter bounds induce for parameter v_i its interval $I(v_i) = [b1, b2]$ where holds $b1 \leq b2$. Let $U = \{u \mid \forall\, v_i \in V.\ u(v_i) \in I(v_i)\}$ be a set of valuations, which we call a region for parameters set V.

Note that regions represent instances $\mathcal{M}[u]$ derived from the parametric model $\mathcal{M}$. As we presented above, these model instances are *well-defined* when they hold some assumptions, so we shift them also to regions.

Definition 7 (Well-defined region)**.** Let $\mathcal{M}$ be a parametric probabilistic model. Let r be a region for set of parameters V, we call the region v *well-defined* for $\mathcal{M}$, when holds for each *polynomial* $f \in \mathcal{M}$ either $f[u] > 0$ or f can be reduced to 0 $f = 0$, and for each *valuation* $u \in r$ holds that u is *well-defined* for model $\mathcal{M}$.

The first condition guarantees that both models $\mathcal{M}$ and $\mathcal{M}[u]$ have the same topology, and the second one says that $\mathcal{M}[u]$ represents a probabilistic model (MDP or MC).

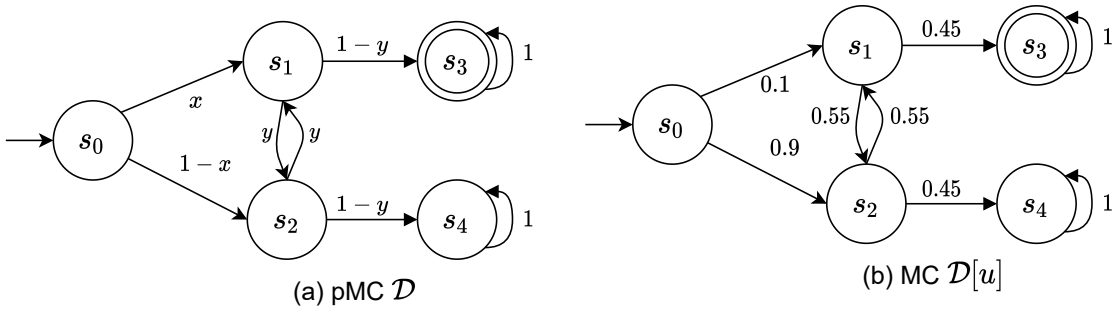


Figure 2.3: A *parametric* MC $\mathcal{D}$, and instantiated MC $\mathcal{D}[u]$ from it.

Example 4 (Region)**.** Let us consider the pMC $\mathcal{D}$ depicted in Figure 2.3(a), the region r defined as $r = [0.1, 0.5] \times [0.5, 0.6]$ and valuation derived from it $u = (0.1, 0.55)$. Applying a valuation u to parametric model $\mathcal{D}$, yields a parameter-free MC $\mathcal{D}[u]$ depicted in Figure 2.3(b). Note, that this instantiated model $\mathcal{D}[u]$ has the same topology as the original pMC. We can say that the considered region r is *well-defined*, since this holds for each possible valuation $u' \in R$, respectively for an instance $\mathcal{D}[u']$ derived from it. At the contrary, the region $r' = [0.0, 0.5] \times [0.75, 1.0]$ is not *well-defined*, because for example the valuation $v' = (0.0, 1.0)$ yields an MC which has no transition from s_0 to s_1, or from s_1 to s_3, so it has different topology as original pMC $\mathcal{D}$.

We will introduce a *satisfaction relation* for regions, which defines whether the given region r and parametric model $\mathcal{M}$ satisfies the specification φ. We denote $\mathcal{M}, r \models \varphi$ when each instance derived from the parametric model $\mathcal{M}$ described with a region r satisfies the analysed specification φ.

Definition 8 (Region satisfaction relation)**.** The relation $\models$ for a well-defined region r, a parametric model $\mathcal{M}$, and a given specification φ is defined as follows:

$$\mathcal{M}, r \models \varphi \iff \forall u \in r.\ \mathcal{M}[u] \models \varphi$$

When holds $\mathcal{M}, r \not\models \varphi$, then this statement implies $\exists u \in r.\ \mathcal{M}[u] \not\models \varphi$. On the contrary, when holds $\mathcal{M}, r \models \neg\varphi$ which implies $\forall u \in r.\ \mathcal{M}[u] \not\models \varphi$, this is different from the first case. Let $\mathcal{D} = (S, V, s_0, \mathcal{P})$ be an pMC, r a well-defined region for this pMC $\mathcal{D}$, and $\varphi = \mathbb{P}_{\geq\lambda}[\mathrm{F}\ T]$ a given reachability property. When we want to conclude whether the region r is *live* or not, we need to compute for each valuation $u \in r$ the corresponding (*minimal* or *maximal*) reachability probability. We define the following equivalences to detect whether given *region* r is *live* or not:

$$\mathcal{D},\ r \models \varphi \iff (\arg\min_{u \in r} \mathbb{P}[\mathcal{D}[u] \models F\ T]) \geq \lambda$$
$$\mathcal{D},\ r \models \neg\varphi \iff (\arg\max_{u \in r} \mathbb{P}[\mathcal{D}[u] \models F\ T]) < \lambda$$

Since the corresponding probability $\mathbb{P}[\mathcal{D}[u], r \models F\ T]$ can be represented as a rational function f, and the region r is well-defined, there exists the valuation inducing the minimal or maximal reachability probability on considered region r.

Example 5 (Region model checking)**.** Let $\mathcal{D}$ be an MC depicted in Figure 2.3(a), and we consider a region $r = [0.1, 0.5] \times [0.5, 0.6]$. Let us suppose, that we want to obtain a valuation $u \in r$ maximising $\mathbb{P}[\mathcal{D}[u], r \models F\ \{s_3\}]$, so the probability that from an initial state s_0 will be reached the state s_3. Let us observe, that the single state from which the state s_3 is not reachable is the state s_4, which is reachable only from state s_2. Therefore, to achieve the highest probability will be the best strategy minimising the probability to reach state s_2. We can conclude, that the parameter x should has its value $u(x)$ as high as possible, i.e., $u(x) = 0.5$. Now, we consider the state s_1, when we want to reach state s_3 from it, then the parameter y should has its value $u(x)$ as low as possible. However, at the same time we would need the highest possible value for the parameter y to reach state s_1 from state s_2. An exhaustive analysis is required to find an optimal value for parameter y, due to this trade-off.

2.2.2 Substitution of pMCs

We will introduce a technique based on the dropping of parameters dependencies in the different states. Thanks to the removal of these dependencies, we can then effectively compute an *optimal* solution. In order to compute minimal (or maximal) probability we have to make a discrete choice over valuations $v : V \to \mathbb{R}$ with $v(v_i) \in \mathcal{B}(v_i)$. The key idea consists of doing such a choice locally at each state, which allow us to construct the *parameter-free MDP* out of the pMC $\mathcal{D}$ and the region r. The non-deterministic choices in this constructed MDP represent each valuation which is required to consider.

Definition 9 (Substitution-pMC)**.** Let $\mathcal{D}^r_{sub} = (S, s_0, Act_{sub}, \mathcal{P}_{sub})$ be an MDP representing the parameter-substitution of a pMC $\mathcal{D} = (S, V, s_0), \mathcal{P})$ and a region r, where holds $Act_{sub} = \biguplus_{s \in S}\{v : V_s \to \mathbb{R} \mid v(v_i) \in \mathcal{B}(v_i)\}$ and

$$\mathcal{P}_{sub}(s, v, s') = \begin{cases} \mathcal{P}(s, s')[v] & if\ v \in Act_s \\ 0 & otherwise. \end{cases}$$

Intuitively, the parameter-substitution $\mathcal{D}^r_{sub}$ of a pMC $\mathcal{D}$ and a region r chooses an action v in state s which corresponds to the assigning of extremal values $\mathcal{B}(v_i)$ to the parameter v_i. Note that the non-deterministic choices introduced by this substitution depend only on the values $\mathcal{B}(v_i)$ of the parameter v_i in the region r.

Example 6 (Substitution-pMC)**.** Let us consider pMC $\mathcal{D}$ depicted in Figure 2.3(a) and also the introduced region $r = [0.1, 0.5] \times [0.5, 0.6]$ as above. Figure 2.4(a) depicts the *substitution* of MC $\mathcal{D}$, which includes non-deterministic choices instead of outgoing transition from states s_0, s_1, s_2 in the original pMC $\mathcal{D}$. In more precise, we choose the *upper* or *lower* bound for the corresponding interval of the parameter in each state. The green lines depict the transitions that belong to the action for the lower bound, and red lines for the upper bound. We note, that states s_3 and s_4 contains only constant outgoing transition and therefore they have not any choices. Figure 2.4(b) depicts an MC $(\mathcal{D}_{sub_r})^\sigma$ which is induced by the scheduler σ on MDP $\mathcal{D}_{sub_r}$.

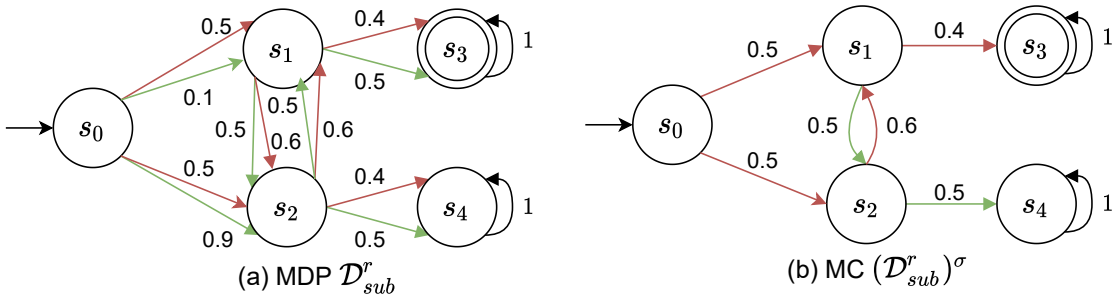


Figure 2.4: A *parameter-substitution* MDP $\mathcal{D}_{sub_r}$, and instantiated MC $(\mathcal{D}_{sub_r})^\sigma$.

According to Theorem 1 which was defined and proved in [34], we list the following:

Theorem 1. Let $\mathcal{D} = (S, V, s_0, \mathcal{P})$ be an pMC, r a region defined over this pMC $\mathcal{D}$, and $T \subseteq S$ a set of target states of $\mathcal{D}$, then holds:

$$\arg\min_{u \in r} \mathbb{P}[\mathcal{D}[u] \models F\ T] \geq \arg\min_{\sigma \in \Sigma^{\mathcal{D}^r_{sub}}} \mathbb{P}[(\mathcal{D}^r_{sub})^\sigma \models F\ T]$$
$$\arg\max_{u \in r} \mathbb{P}[\mathcal{D}[u] \models F\ T] \leq \arg\max_{\sigma \in \Sigma^{\mathcal{D}^r_{sub}}} \mathbb{P}[(\mathcal{D}^r_{sub})^\sigma \models F\ T]$$

We define the following consequence straightforward from this statement:

Theorem 2. Let $\mathcal{D} = (S, V, s_0, \mathcal{P})$ be an pMC, r a well-defined region defined over this pMC $\mathcal{D}$, then holds:

$$\mathcal{D}^r_{sub} \models \mathbb{P}_{\leq \lambda}[\mathrm{F}\ T] \Longrightarrow \mathcal{D}, r \models \mathbb{P}_{\leq \lambda}[\mathrm{F}\ T]$$
$$\mathcal{D}^r_{sub} \models \mathbb{P}_{> \lambda}[\mathrm{F}\ T] \Longrightarrow \mathcal{D}, r \models \neg\mathbb{P}_{\leq \lambda}[\mathrm{F}\ T]$$

Therefore, we can use the standard approaches for MDP model checking to check whether $\mathcal{D}, r \models \varphi$, by applying these techniques to model $\mathcal{D}^r_{sub}$. When the model checking of this considered over-approximation yield an inconclusive results, we refine the regions to sub-regions and repeat the control.

2.2.3 Substitution of pMDPs

In the previous section, we introduced a technique to apply the standard MDP model checking to the parametric MCs. This technique substitutes outgoing transitions with non-deterministic choices representing the relevant bounds of the parameter related to this transition. Now, we generalise this technique to parametric MDPs, or in general, systems

that include non-determinism. In addition to the choices over actions that the MDP implicitly contains, this technique also adds options over valuations concerning the considered parameters. This added level of non-determinism leads to a game with two players. The first player represents the original MDP non-determinism, and the second player the abstracted parameters, which results in a stochastic game.

Example 7 (MDP Substitution)**.** Let $\mathcal{M} = (S, V, s_0, Act, \mathcal{P})$ be an pMDP depicted in Figure 2.5(a). Note that an initial state s_0 can choice from two actions α and β. When we consider the scheduler $\sigma = \{s \mapsto \beta\}$, then by applying it to pMDP $\mathcal{M}$ we obtain pMC, which we can subsequently transform to parameter-substitution MC, as depicted in Figure 2.5(b). The parameter substitution of a MDP $\mathcal{M}$ returns an SG $\mathcal{G}$ which is depicted in Figure 2.5(d).

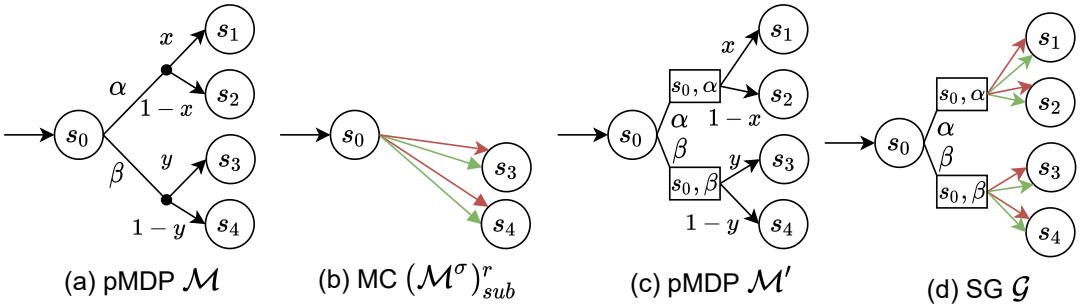


Figure 2.5: Illustration of the substitution of a pMDP.

When constructing the *parameter substitution* of a pMDP, we first need to construct its substitution, which separates probabilistic choices from non-deterministic actions. In the first step, we split each state $s \in S$ into set $\{s\} \uplus \{(s, \alpha) \mid \alpha \in Act(s)\}$, and then we add a transition with the probability of 1 from state s to state (s, α) and move the probabilistic choice concerning action α from s to (s, α). After applying of these operations, we obtain a pMDP $\mathcal{M}'$, depicted in Figure 2.5(c). We can see that this MDP has clear non-deterministic choices leading to the newly added states of the form (s, α), which on the contrary, have clear probabilistic choices. As we said above, by introducing parameter substitution on the probabilistic states, we obtain the stochastic game $\mathcal{G}$, depicted in Figure 2.5(d). The first player represents the original pMDP non-determinism, while the second one decides whether the lower or upper bounds of the parameters will be set.

Definition 10 (Substitution-pMDP)**.** Let $\mathcal{M} = (S, V, s_0, Act, \mathcal{P})$ be an pMDP, and r well-defined region. A *stochastic game* $\mathcal{M}^r_{sub} = (S_\circ \uplus S_\square, s_0, Act_{sub}, \mathcal{P}_{sub})$ is the parameter-substitution of $\mathcal{M}$ over r if $S_\circ = S$, $S_\square = \{(s, \alpha) \mid \alpha \in Act(s)\}$, $Act_{sub} = Act \uplus (\biguplus_{(s,\alpha)\in S_\square} Act^\alpha_s)$ where $Act^\alpha_s = \{v : V^\alpha_s \to \mathbb{R} \mid v(v_i) \in \mathcal{B}(v_i)\}$, and

$$\mathcal{P}_{sub}(t, \beta, t') = \begin{cases} 1 & if\ t \in S_\circ\ and\ t' = (t, \beta) \in S_\square, \\ \mathcal{P}(s, \alpha, t')[\beta] & if\ t = (s, \alpha) \in s_\square, \beta \in Act^\alpha_s, and\ t' \in S_\circ, \\ 0 & otherwise. \end{cases}$$

The constructed stochastic game $\mathcal{G} = \mathcal{M}^r_{sub}$ induced by various schedulers for the first player $\circ$ relates with the substitution in the pMCs induced by schedulers from $\mathcal{M}$. In more precise, the schedulers $\sigma \in \Sigma^M_\circ$ for the first player $\circ$ coincide with the schedulers in

$\mathcal{M}$. Therefore, the models $\mathcal{G}^\sigma$ and $(\mathcal{M}^\sigma)^r_{sub}$ induced by the scheduler σ induce the same reachability probabilities. Thus, we can substitute $\mathcal{M}$ directly and preserve the reachability probability, instead of performing the substitution on the pMC induced by $\mathcal{M}$ and σ.

Theorem 3. Let $\mathcal{M} = (S, V, s_0, Act, \mathcal{P})$ be an pMDP, r a well-defined region defined over this pMC $\mathcal{D}$, then holds:

$$\mathcal{M}^r_{sub} \models_\emptyset \mathbb{P}_{\leq\lambda}[\mathrm{F}\ T] \implies \mathcal{M}, r \models \mathbb{P}_{\leq\lambda}[\mathrm{F}\ T]$$
$$\mathcal{M}^r_{sub} \models_\circ \mathbb{P}_{>\lambda}[\mathrm{F}\ T] \implies \mathcal{M}, r \models \neg\mathbb{P}_{\leq\lambda}[\mathrm{F}\ T].$$

Therefore, similar as in the introduced pMC case, we can determine whether $\mathcal{M}^r_{sub} \models \varphi$ by analysing a stochastic game. For more details about proofs of listed definitions and theorems please refer to [34].

2.3 Families of Markov Chains

This thesis considers a parametric transition probability function as an explicit representation of an MCs family. Such explicit representation relieves the presentation and provides to describe exciting and practical problems for probabilistic synthesis. On the other hand, arbitrary probabilistic programs permit the modelling of more complex and independent parameter structures [9]. In this thesis and our implementation, we consider a more flexible high-level modelling language, see Section 5.2.

We want to focus on families of (parametric) Markov chains having different topologies of the state space and having undefined transition probabilities. Such families have different sets of reachable states and also different probabilities of reaching individual states. More specifically, we consider the family, which is parametrised by a finite set of discrete parameters K and a finite set of parameters V over the domain $\mathbb{R}$.

Definition 11 (Family of pMCs)**.** [9] A *family of pMCs* $\mathcal{F}$ is a quintuple $(S, s_0, K, V, \mathcal{B})$ where S is a finite set of *states*, $s_0 \in S$ is an *initial state*, K is a finite set of parameters with domains $T_k \subseteq S$ for each $k \in K$, V is a finite set of parameters over the domain $\mathbb{R}$, and $\mathcal{B} : S \times K \nrightarrow \mathbb{Q}_V$ is a family of parametric transition probability functions.

The transition probability function $\mathcal{B}$ of pMCs family $\mathcal{F}$ maps each state $s \in S$ to the probability distribution over parameters from K, where the transitions are labelled with polynomials in $\mathbb{Q}_V$. As we mentioned above, these parameters from both sets K and V represent undefined system specifications when the probabilistic synthesis is considered. This function $\mathcal{B}$ yields a specific pMC when we instantiate each parameter $k \in K$ with the specific value from its domain $T_k \subseteq S$. We call such instantiated pMC as *realisation*, and the following definition describes it.

Definition 12 (Realisation)**.** [9] A *realization* of a family $\mathcal{F} = (S, s_0, K, V, \mathcal{B})$ of pMCs is a function $r : K \to S$ s.t. $\forall k \in K : r(k) \in T_k$. A realization r yields a pMC $\mathcal{D}_r = (S, V, s_0, \mathcal{B}(r))$, where $\mathcal{B}(r) : S \times K \nrightarrow \mathbb{Q}_V$ represents the parametric transition probability matrix where each parameter $k \in K$ is substituted with $r(k)$. Let $\overline{\mathcal{R}}$ denote the set of all (parametric) realisations for $\mathcal{F}$.

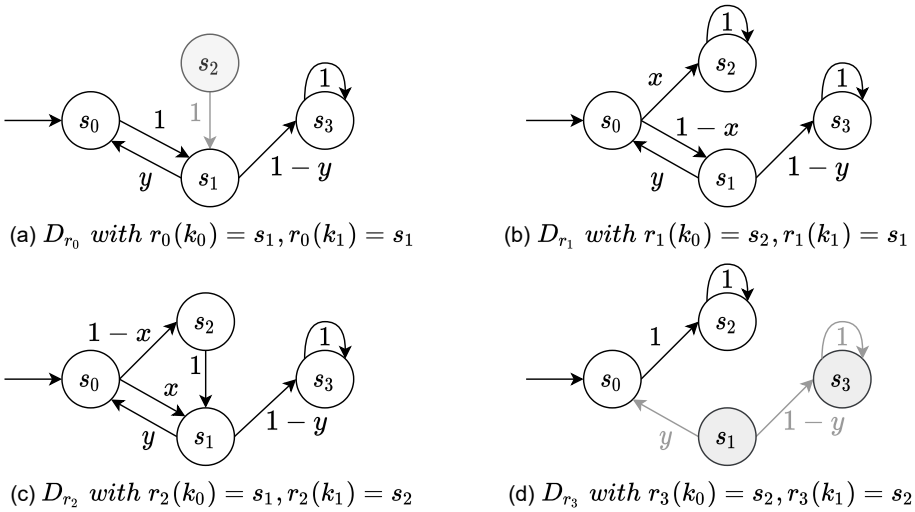
A set $\prod_{k\in K} T_k$ representing all possible parameter combinations values has the same semantics as a set $\overline{\mathcal{R}}$ representing all family realisations with different topology of the state space. In other words, we can define the *size of the family* of pMCs in the following way:

$|\mathcal{F}| := |\prod_{k\in K} T_k| = |\overline{\mathcal{R}}| = \prod_{k\in K} |T_k|$. The family size $|\mathcal{F}|$ is finite because of the finiteness of each parameter domain $T_k \subseteq S$, but it is exponential in the number of parameters $|K|$. The individual pMCs within the family share the same state space S, but their set of reachable states can vary. We emphasise that, from the point of view of program synthesis, such defined size of the family does not indicate the maximum number of realisations that must be examined. This size indicates the number of parametric realisations with different topologies, which, however, may still be inconclusive due to continuous parametric regions. We note $\mathcal{R}$ as the sub-families induced from the whole family $\overline{\mathcal{R}}$.

Example 8 (Family of MCs)**.** Let $\mathcal{F} = (S, s_0, K, V, \mathcal{B})$ be a family of pMCs, where $S = \{s_0, s_1, s_2, s_3\}$, $K = \{k_0, k_1, k_2, k_3\}$ with domains $T_{k_0} = T_{k_1} = \{s_1, s_2\}$, $T_{k_2} = \{s_0\}$, and $T_{k_3} = \{s_3\}$, and $V = \{x, y\}$. The parametric transition function $\mathcal{B}$ is defined as follows:

$$\mathcal{B}(s_0) = x : k_0 + (1-x) : k_1$$
$$\mathcal{B}(s_1) = y : k_2 + (1-y) : k_3$$
$$\mathcal{B}(s_2) = 1.0 : k_0$$
$$\mathcal{B}(s_3) = 1.0 : k_3$$

Figure 2.6 draws the pMCs family $\mathcal{F}$ that correspond to all possible realisations: $|T_{k_0}| \cdot |T_{k_1}| = 2 \cdot 2 = 4$. We note that these pMCs have a different topology of the underlying state space, resulting in different sets of reachable states.



(a) D_{r_0} *with* $r_0(k_0) = s_1, r_0(k_1) = s_1$ (b) D_{r_1} *with* $r_1(k_0) = s_2, r_1(k_1) = s_1$

(c) D_{r_2} *with* $r_2(k_0) = s_1, r_2(k_1) = s_2$ (d) D_{r_3} *with* $r_3(k_0) = s_2, r_3(k_1) = s_2$

Figure 2.6: A family $\mathcal{F}$ of four various realisations. Unreachable states are greyed out.

Definition 13 (Specification)**.** In our work, we focus on the conjunctions of specifications with *reachability* and *expected rewards*. Let T be a set of target states, then the reachability property $\varphi \equiv \mathbb{P}_{\bowtie\lambda}[\mathrm{F}\ T]$ where $\bowtie \in \{<, \leq, >, \geq\}$ and $0 \leq \lambda \leq 1$ expresses that the probability of reaching T refers to $\lambda \in [0, 1]$ in agreement with $\bowtie$. Expected reward property $\varphi \equiv \mathbb{E}_{\bowtie\lambda}[\mathrm{F}\ T]$ expresses that the expected reward accumulated before T is reached refers to $\lambda \in \mathbb{R}^+$ in accordance with $\bowtie \in \{<, \leq\}$. Let $\mathcal{P}[r]$ be a program induced by the realisation r, we denote $\mathcal{P}[r] \models \varphi$ when this program satisfies the specification φ. Let $\Phi = \{\varphi_i\}_{i\in I}$ be a finite set of specifications, when $\forall i \in I : \mathcal{P}[r] \models \varphi_i$ then we write $\mathcal{P}[r] \models \Phi$.

We aim to two kinds of synthesis tasks for a given probabilistic program described with a realisation set $\overline{\mathcal{R}}$, and a specification set Φ. The first task tries to identify just one realisation $r \in \overline{\mathcal{R}}$ that satisfy the given specification set Φ. This task represents a special instance of the threshold synthesis, which tries to divide the realisation set $\overline{\mathcal{R}}$ into two subsets based on their satisfiability. We do not address this synthesis task in our work. The second task on which we focus our attention is finding a realisation that minimises or maximises a given objective.

Definition 14 (Feasibility)**.** Find a realisation $r \in \overline{\mathcal{R}}$ such that $\mathcal{P}[r] \models \Phi$.

Definition 15 (Minimality)**.** For property $\varphi_{\min}$, find a realisation $r^* \in \overline{\mathcal{R}}$ such that:

$$r^* \in \arg\min_{r \in \overline{\mathcal{R}}} \{\mathbb{P}[\mathrm{P}[r] \models \varphi_{\max}] \mid \mathrm{P}[r] \models \Phi\}\,.$$

.

We defined only minimal synthesis task for probability property, but its variants for maximisation and expected rewards may be defined analogously. Moreover, we focus also to a relaxed variant of minimal synthesis, the so-called *ε-minimal synthesis*, defined as follows: $\mathcal{P}[r^*] \models \Phi$ *and* $\mathbb{P}[\mathcal{P}[r^*] \models \varphi_{min}] \leq (1-\varepsilon) \cdot \min_{r \in \overline{\mathcal{R}}}\{\mathbb{P}[\mathcal{P}[r] \models \varphi_{min}] \mid \mathcal{P}[r] \models \Phi\}$.

Example 9. (Synthesis problem) Assume an MCs family $\mathcal{F}$ from Example 8 and the specification $\varphi = \mathbb{P}_{\geq 0.1}[\mathrm{F}\ \{s_1\}]$. The solution to the feasibility synthesis problem is, for example, the realisation r_0, since $\mathcal{D}_{r_0}$ has a probability of $\frac{2}{3}$ to reach state s_1. For $\Phi = [F\ \{s1\}]$, the solution to the maximal synthesis problem on MCs family $\mathcal{F}$ is the realisation r_2, as MC $\mathcal{D}_{r_2}$ has a probability equal to one to reach state s_1.

Quotient pMDP $\mathcal{M}^{\mathcal{F}}$ simulates the behaviours of each member of the family $\mathcal{F}$ and can even pass between them during the execution. In more precise, when the path $s_0 s_1 s_2 \ldots$ is executable in some family member $\mathcal{D}_r$, then it is also executable in $\mathcal{M}^{\mathcal{F}}$ as $s_0 \xrightarrow{r} s_1 \xrightarrow{r} \ldots$. However, there may exist a path π that is executable in $\mathcal{M}^{\mathcal{F}}$, but it is not realisable in neither of the family members. We can conclude that *quotient* MDP over-approximates the behaviours of the members of family $\mathcal{F}$.

Definition 16 (Quotient pMDP)**.** [1] Let $\mathcal{F} = (S, s_0, K, V, \mathcal{B})$ be an pMCs family. A *quotient* pMDP $\mathcal{M}^{\mathcal{F}}$ of the family $\mathcal{F}$ is a tuple $(S, V, s_{init}, \mathcal{R}^{\mathcal{F}}, \mathcal{P})$, where $\mathcal{P}(\cdot)(r) \equiv \mathcal{B}_r$.

A restriction of a *quotient* pMDP concerning $\mathcal{R} \subseteq \mathcal{R}^{\mathcal{F}}$ induces an pMDP which takes into account only transitions associated with $\mathcal{R}$. We define the usage of *consistent* schedulers ensuring that execution of a *quotient* pMDP always selects the concrete realisation. We note that a *consistent* scheduler yields a specific family member for a *quotient* pMDP, which directly follows from Definition 16.

Definition 17 (Consistent scheduler)**.** [1] Let $\mathcal{F} = (S, s_0, K, V, \mathcal{B})$ be a pMCs family and let $\mathcal{M}^{\mathcal{F}} = (S, V, s_{init}, \mathcal{R}^{\mathcal{F}}, \mathcal{P})$ be a quotient pMDP of the family $\mathcal{F}$. For $r \in \mathcal{R}^{\mathcal{F}}$, a (memoryless) scheduler $\sigma_r \in \sigma^{M^{\mathcal{F}}}$ is called *r-consistent* iff $\forall s \in S : \sigma(s) = r$. A scheduler is called *consistent* iff it is consistent for some $r \in \mathcal{R}^{\mathcal{F}}$.

Chapter 3

Synthesis of Probabilistic Programs

3.1 Automated System Design

The dream of automating software development has been present since the beginning of the computer age. As early as 1945, as part of an ideal vision for an automatic computing machine, Alan Turing argued that the process of compiling instruction tables would be fascinating, free of errors and no effort, as the machine itself would perform all demanding operations. What if we could tell the computer what to do – or give it a specification of what we want as users – and let it work to figure out how to do it? This is the promise of program synthesis, where after entering a specification, the computer automatically generates a program that satisfies that specification.

Deductive Synthesis. The earliest examples of program synthesis used formal specifications to infer the correct program automatically. The aim is to take a formal specification and search backwards for a program that provably satisfies the specification. Some of the synthesisers in this area relied on the successes of automated theorem provers, which they used to compile proof of program accuracy. Others used a comprehensive set of rewriting rules in an attempt to transform a specification into an equivalent program. This approach is similar to a regular compiler transforming the input program into an output program using rewriting rules from specified grammar. However, the rewriting rules of the synthesiser should be complete – the synthesiser tries to find the optimal program – and should be able to transform the specification into every equivalent program. The landmarks in this area are works based on super-optimisation [28] and the Synthesis kernel [33].

Syntax-Guided Synthesis. The obstacle of deductive synthesis is that it requires both a complete formal specification and a complete axiomatisation of the program target language. The second wave of synthesis research focuses on the *syntax-driven* synthesis, trying to relax these requirements. The input to the *syntax-guided* synthesis problem consists of a semantic correctness specification for the desired program given by a logical formula and a syntactic set of candidate programs. This set is described by a sketch [37] (program template) containing undefined locations described as yet unknown system properties or maybe intentionally omitted. Approaches of this second wave have recently come to the fore, and our approach is also based on them.

3.2 Related Work

The synthesis tasks for parametric probabilistic programs can be divided into the following two categories.

Topology Synthesis. This synthesis task considers a finite set of parameters that affect the *topology* of Markov chains. Finding a family member that satisfies given specifications is *NP*-complete in the parameters' number and can be naively solved by analysing all individual family members [12]. An alternative approach models the family of MCs by an MDP and uses off-the-shelf algorithms for MDP model-checking [13]. Tools such as *QFLan* [39] and *ProFeat* [11] implements these approaches to quantitatively analyse alternative designs of software product lines [35]. These tools provide the complete methods to solve synthesis tasks, but they are limited strictly to small families, so they do not scale. Inductive techniques, motivated from these basic approaches, form on abstraction-refinement (*AR*) over the MDP representation [9], and counterexample guided inductive synthesis (*CEGIS*) for MCs [8] have been proposed to achieve better scalability. Experiments in previous papers [9, 8] have shown that the synthesis methods implemented in *ProFeat* or *QFLan* have evident deficits on the benchmarks investigated also in this thesis. This synthesis task is closely connected to controller synthesis for partially observed MDPs (POMPDs), representing, for instance, a famous Maze [30] model for AI planning under uncertainty, to which we will pay attention later. Other techniques to solve the controller synthesis for POMDPs also use neural network oracles to lead search [29] and adaptive learning schemes based on imitation learning [20].

Parameter Synthesis. It considers models with a fixed topology but with uncertain parameters associated with transition probabilities (or rates). It analyses how the parameter values affect the behaviour of Markov chains (or MDPs). *Parameter* synthesis problems targets finding parameter values for which the parametric model (optimal) satisfies the specification. The state-of-the-art probability model checkers Storm [16] and Prism [25] provide approximate techniques for parameter synthesis that solve the same parameters in various transitions independently. We note that these model checkers and others with the same focus are primarily determined to verify concrete MC and not to the whole MCs family. On the contrary, exact techniques construct rational functions for symbolic reachability probabilities have been proposed in [14] and further improved in [15]. Parametric probabilistic models have several applications in a wide range of areas. For instance, in [7] synthesised the rate parameters in stochastic biochemical networks, in [4] tuned the model parameters exploiting the parametric Markov chains. Moreover, parametric models have applications also when ranking the patches in the software repair [26] and for computing perturbation bounds [38].

Search-based approaches can handle both synthesis tasks, but they do not guarantee an exhaustive exploration of the parameter space. Genetic algorithms and evolutionary techniques belong to these approaches group, and the tool Rodes [6] implements their combination with parameter synthesis. However, these approaches are not complete and can only effectively solve a feasible instance within the feasibility synthesis task. In the case of an unfeasible instance or optimally synthesis task, they cannot be applied to them. Thus, we omit comparison with the incomplete methods that cannot treat these types of synthesis tasks.

3.3 Synthesis Methods

Existing methods for a probabilistic programs synthesis can be separated into two orthogonal classes. The first class involves complete methods that prove the non-existence, or potentially optimally, of the given probabilistic program. On the contrary, incomplete methods handling various evolutionary techniques and intelligent search strategies form the second one [17]. However, these incomplete methods cannot treat unfeasible and optimal synthesis problems compared to the complete methods. Therefore, we focus on the complete state-of-the-art methods even though the incomplete methods provide a valuable flexibility level. As a reference and baseline method, we consider a *one-by-one* approach that enumerates through each design space member [12]. An explosion of the design space renders this technique unfeasible for large families, necessitating using advanced approaches utilising the arbitrary structure of the Markov chain family.

This thesis focuses on the complete methods considered within the *oracle-guided* synthesis approach [21, 22]. The control unit called *learner* selects a realisation r from the designs family and passes it to the performing unit called *oracle*. This unit provides an answer to whether realisation r satisfies a given specification Φ. Whenever it is not this case, it provides supplementary information representing counterexample (CEGIS) or bounds from MC model checking (AR). For purposes of this thesis, two various orthogonal oracles can be considered:

(i) **Inductive Oracle** (CE): It tries to infer the declarations (counter-examples) about other family members by analysing individual realisation [8].

(ii) **Deductive Oracle** (AR): Abstraction refinement oracle considers more family members at once and then infers the consequences of these members' constructed aggregation [9].

Inductive Oracle. *Counter-examples* represents a concrete system execution violating a given specification φ. When considering the probabilistic model checking of Markov chains, the counter-examples represent paths set, which have the probabilities added to a quantity that violates $\bowtie \lambda$. When an MC D violates the given specification φ ($MC\ D\ \not\models \varphi$), a *counter-example* provides diagnostic information related to the states causing the violation. We consider the counter-examples related to critical sub-systems:

Definition 18 (Counter-example)**.** Let $D = (S, s_0, \boldsymbol{P})$ be an MC with $s_\perp \notin S$. An MC $D \downarrow C = (S \cup \{s_\perp\}, s_0, \mathbf{P}')$ induced by $C \subseteq S$ is sub-MC of MC D, where the transition probability matrix $\mathbf{P}'$ is defined as follows:

$$\mathbf{P}'(s) = \begin{cases} \mathbf{P}(S) & if\ s \in C \\ [s_\perp \mapsto 1] & otherwise. \end{cases}$$

For the given specification $\mathbb{P}_{\leq\lambda}[\mathrm{F}\ T]$, when holds $D{\downarrow}C \not\models \mathbb{P}_{\leq\lambda}[\mathrm{F}\ (T \cap (C \cup \{s_0\}))]$, then we call the sub-system $D{\downarrow}C$ as a *counter-example* (CE).

Let D_r be an MC that violates the given specification φ. The inductive oracle constructs the critical sub-system $D_r \downarrow C$ to compute other realisations that also violate φ. Subsequently, it uses this constructed sub-system to infer a set called *conflict* for D_r and φ.

Definition 19 (Conflict)**.** For MCs family $\mathcal{F} = (S, s_0, K, V, \mathcal{B})$, and the set $C \subseteq S$, the conflict set K_C is given by $\bigcup_{s \in C} supp(\mathcal{B}(s))$ and consist of relevant parameters.

Deductive Oracle. As we said above, a *quotient* MDP over-approximates the behaviours of each Markov chain in the currently analysed family $\mathcal{F}$. The *deductive* oracle performs the model checking of this MDP, and it can yields the following results. We consider the following specification against which is performed relevant model checking: $\mathbb{P}_{\geq\lambda}[\mathrm{F}\ T]$. The model checking of the quotient MDP returns the corresponding lower (x_{min}) and upper (x_{max}) bounds, as well as the minimising (σ_{min}) and maximising (σ_{max}) schedulers on the considered reachability probability. When $x_{max} < \lambda$, the synthesis task is unfeasible since for each family member $r \in \mathcal{R}^{\mathcal{F}}$ holds that $\mathcal{D}_r \not\models \varphi$. On the contrary, when $x_{min} >= \lambda$, the oracle can return an arbitrary family member as a solution to the synthesis task since each satisfies φ. Last but not least, when $x_{min} \leq \lambda \leq x_{max}$, we cannot decide this case, except when the scheduler σ_{max} is consistent. Such scheduler represents the valid member of the analysed family with $x_{max} \geq \lambda$, so it is a solution to the synthesis task. Otherwise, when scheduler σ_{max} is not consistent, the considered abstraction is too over-approximate, and the problem is undecided.

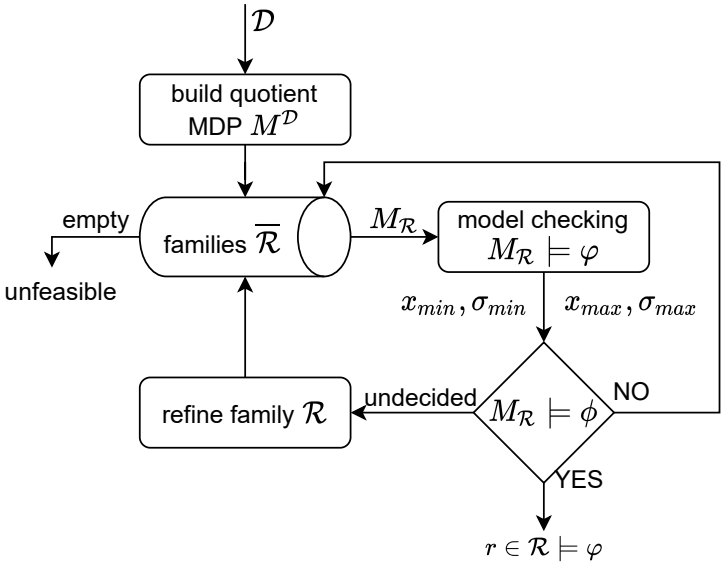


Figure 3.1: An *Abstraction Refinement* (AR) analysing loop.

In such a case, we split the undecidable family of Markov chains into two derived subfamilies. Then each of them represents the input for deductive oracle, which analyse them using the technique introduced above. When derived subfamilies are still undecidable, we refine them into smaller and smaller subfamilies until we find a feasible solution or explore the whole family. When the sub-family represents concrete realisation, so it induces an MC and not MDP ($x_{min} = x_{max}$), such family is necessarily decided and have not split. Moreover, since the family size is finite, the presented technique has guaranteed the termination for these two reasons. Figure 3.1 depicts described deductive approach.

Hybrid methods. The novel integrated approach, the so-called *hybrid* method, combines both these oracles and can thus take advantage of the benefits offered by both approaches [1, 2]. We illustrate the cooperation between the individual units within this method in Figure 3.2. The learner unit maintains the subfamilies queue $\mathcal{R}' \subseteq \overline{\mathcal{R}}$ for subsequent processing and decides which oracle is selected based on their previous efficiency. The *CE-Oracle* analyses the family member r and, when it satisfies the given specification

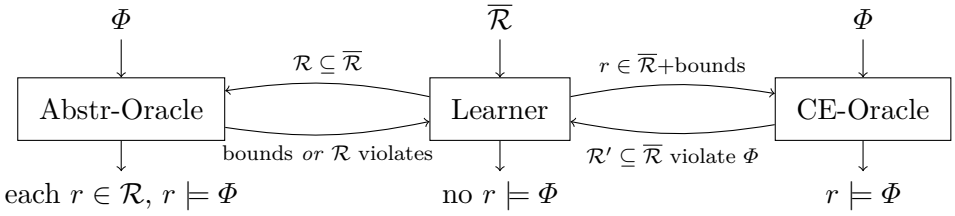


Figure 3.2: Oracle-guided synthesis (adapted from [2]).

.

Φ, then returns it as a solution. On the other hand, it can generalise the analysed realisation r into a subfamily $\mathcal{R}'$, and the learner unit can discard it from the whole design space $\overline{\mathcal{R}}$. Moreover, this oracle exploits the MDP bounds when constructing counterexamples, thanks to which it can generate more petite generalisations. The *Abstr-Oracle* analyses a given sub-family $\mathcal{R} \subseteq \overline{\mathcal{R}}$, and according to the result, it performs the subsequent action. When all realisations $r \in \mathcal{R}$ satisfy Φ, then it returns the overall synthesis result as feasible. In another case, when all realisations violate Φ, the whole analysed sub-family $\mathcal{R}$ will be discarded from the families queue $\overline{\mathcal{R}}$ by the learner. The last option returns safe bounds on the best- and worst-case behaviour of all realisations in $\mathcal{R}$ considered Φ when the analysis result is inconclusive.

Algorithm 1 Hybrid method: Feasibility synthesis with single property.

Input: A MCs family $\mathcal{F}$, a reachability property φ.
Output: *Realization* $r \in \overline{\mathcal{R}}$ *s.t.* $\mathcal{D}_r \models \varphi$, otherwise UNSAT.

1: $\overline{\mathcal{R}} \leftarrow \{\mathcal{R}^{\mathcal{F}}\}$ **// each analysed (sub)-family also holds bounds**
2: $\delta_{CEGIS} \leftarrow 1$ **// time allocation factor for CEGIS**
3: **while** $\overline{\mathcal{R}} \neq \emptyset$ **do**
4: result, $\overline{\mathcal{R}}'$, σ_{AR}, $t_{AR} \leftarrow$ AR($\overline{\mathcal{R}}$, φ)
5: **if satisfiable**(result) **then**
6: **return** result
7: **else**
8: result, $\overline{\mathcal{R}}''$, $\sigma_{CEGIS} \leftarrow$ CEGIS($\overline{\mathcal{R}}'$, φ)
9: **if satisfiable**(result) **then**
10: **return** result
11: **else**
12: $\delta_{CEGIS} \leftarrow \sigma_{CEGIS}$ / σ_{AR}
13: $\overline{\mathcal{R}} \leftarrow \overline{\mathcal{R}}''$
14: **end if**
15: **end if**
16: **end while**
17: **return** UNSAT

An extended synthesis approach was introduced in [1, 2] using the abstraction refinement to family prune and accelerating the construction of counter-examples by CE-oracle. Its main idea is to perform a limited number of abstraction refinement loops and then invoke CEGIS to one of the refined sub-families. It turned out that a moment of the switching can be crucial, and therefore, the method has to detect the rightest moment. One AR iteration is typically significantly slower than one CEGIS iteration since the AR iteration involves an MDP model-checking, which inspires whole method workflow. The advance of

the hybrid methods is also constructing counter-examples within the CEGIS loop, where it uses a fast greedy approach providing smaller generalisations.

As we said above, this *novel integrated* method combines inductive and deductive oracles, and we briefly summarise their operation. CEGIS iterates through all family members (realisations) until it reached the satisfying realisation, if such exists, or when it explores the whole design space. Moreover, it constructs counter-examples whenever the analysed realisation $r \in \mathcal{F}_r$ unsatisfying the given specification Φ. Realisations subset $\mathcal{R}' \subseteq \overline{\mathcal{R}}$ represents such counter-examples that CEGIS subsequently prunes from the analysed design space $\overline{\mathcal{R}}$. On the other hand, an AR loop builds MDP models from the sub-families queue $\mathcal{R} \subseteq \overline{\mathcal{R}}$ that model-checkers analyse in follows. When the analysis results are inconclusive, an AR refines the analysed sub-family and stores the obtained satisfiability bounds for further processing within CEGIS loop. Method allocates the time per both loop according to their performance, e.g., it can consider the number of pruned MCs per timed unit and estimates an efficiency from it. Namely, when it notices that AR prunes sub-families twice as slow as CEGIS, it increases time twice in the next round for CEGIS. The resulting algorithm is summarised in Algorithm 1, adapted from [2].

Chapter 4

Novel Methods for Probabilistic Synthesis

The developed framework for *integrated* synthesis has been designed for *feasibility* synthesis concerning a *single* property. However, probabilistic programs often have to satisfy specifications expressed as a *conjunction* of several temporal logic constraints, potentially including the *optimal* objective. Therefore, we design extensions to generalise the *hybrid* method to handle *multi-property* specifications and treat *optimal* synthesis. In the following, we introduce these extensions to adapt the integrated synthesis for both advanced methods. We design them individually for both *CEGIS* and *AR* loop, whereas the overall framework of the hybrid method is unchanged.

4.1 Multi-Property Synthesis

When considering *multi-property* specifications, the basic ideas of both oracles (*CEGIS* and *AR*) remain the same. When *AR* analyses the quotient MDP concerning multiple properties, it yields multiple probability bounds. *CEGIS* loop constructs a separate conflict for each unsatisfied property whenever it meets an unsatisfiable realisation. Moreover, it uses the corresponding probability bounds obtained at the *AR* loop to improve the quality of generated counter-examples.

CEGIS. The CEGIS performs the *multi-property* synthesis in the almost same manner as the *feasibility* synthesis for a single property, but there are a few differences. We decided to analyse each property $\varphi_i \in \Phi$ for each considered realisation $r \in \mathcal{R}$, even if we come across a property that the given realisation does not satisfy. We construct the counter-examples whenever the analysed realisation r does not satisfy the given specification φ_i. In this way, we prune the design space of the analysed family more efficiently because each constructed counter-example throws out a certain number of family members. The core of the loop stays the same. We pick the concrete realisation $r \in \mathcal{R}$, then construct the corresponding MC $\mathcal{D}_r$ and perform the model checking against to current specification φ_i. The synthesis terminates when a satisfying realisation against the whole specification set Φ is found, or the whole state space is exhausted, indicating that no feasible solution in the analysed family exists. This approach is summarised in Algorithm 2.

Algorithm 2 CEGIS loop: Multi-property synthesis.

Input: A family $\mathcal{F}$ of MCs with the set $\mathcal{R} \subseteq \overline{\mathcal{R}}$ of realisations, and a set of properties $\Phi = \{\varphi_0, \ldots, \varphi_{N-1}\}$.

Output: A realisation $r \in \mathcal{R}$ such that $\forall\ 0 \leq i < N.\ \mathcal{D}_r \models \varphi_i$, if such exists, otherwise `UNSAT`.

```
1: while R ≠ ∅ do
2:   allSat ← True
3:   r ← any(R)
4:   D_r ← buildDTMC(R, r)
5:   for φ_i ∈ Φ do
6:     sat ← solveDTMC(D_r, φ_i)
7:     if sat then
8:       if i = N − 1 ∧ allSat then
9:         return r
10:      else
11:        continue
12:      end if
13:    else
14:      R ← R \ constructCE(D_r, φ_i)
15:      allSat ← False
16:    end if
17:  end for
18: end while
19: return UNSAT
```

Abstraction Refinement. *Multi-properties* synthesis within the AR loop is performed in the same loop as the single-property synthesis, with some modifications. The algorithm's input represents the MCs family $\mathcal{F} = (S, s_0, K, V, \mathcal{B})$ with the corresponding set of realisations $\mathcal{R} \subseteq \overline{\mathcal{R}}$, and a set of properties Φ_D explicitly determined for this family $\mathcal{F}$. This specification set $\Phi_{\mathcal{F}}$ includes the specifications that need to be analyzed within the family, in other words, specifications that have not yet been satisfied.

The algorithm first constructs the *quotient* MDP $\mathcal{M}^{\mathcal{F}}$ concerning the given family $\mathcal{F}$ and realisation set R, and then iterates over sub-families of U that have not been yet analysed. For each selected sub-family $\mathcal{F}$ is then performed the MDP model checking on the relevant restricted MDP, which yields computed minimal (`min`) and maximal (`max`) probabilities, and corresponding schedulers (σ_{min} and σ_{max}) related to them, respectively. We analyse these obtained results concerning current specification, mainly whether it is *safety* or *liveness* specification, from their feasibility and decidability.

When holds $x_{min} > \lambda$ for safety property or $x_{max} < \lambda$ for liveness property ($\neg$`sat`) then all realisations $r \in \mathcal{R}$ violate specification φ_i, and the algorithm continues on the next sub-family or terminates with `UNSAT` result. On the contrary, when holds $x_{max} \leq \lambda$ for safety property or $x_{min} \geq \lambda$ for liveness property then each family member satisfy specification φ_i, and the algorithm continues on the next specification φ_{i+1} or yields any realisation $r \in \mathcal{R}$ as the solution to the synthesis task. Finally, when holds $x_{min} \leq \lambda \leq x_{max}$ for both kinds of properties, then we have to check whether the corresponding scheduler σ is consistent or not. When it is consistent, it represents a family member satisfying the specification φ_i, since it has $x_{min} \leq \lambda$ for safety and $x_{max} \geq \lambda$ for liveness property. Otherwise, the synthesis task

is still undecided, and the currently analysed sub-family $\mathcal{R}$ will be split. This procedure is summarised in Algorithm 3.

Algorithm 3 AR loop: Multi-property synthesis.

Input: A family $\mathcal{F}$ of MCs with the set $\mathcal{R} \subseteq \overline{\mathcal{R}}$ of realisations, and a properties set $\Phi_{\mathcal{F}} = \{\varphi_0, \ldots, \varphi_M\}$.

Output: A realisation $r \in \mathcal{R}$ s.t. $\forall\, 0 \leq i < M.\ \mathcal{D}_r \models \varphi_i$, if such exists, otherwise `UNSAT`.

$U \leftarrow \{\mathcal{R}\}$
$\mathcal{M}^{\mathcal{F}} \leftarrow$ `buildQuotientMDP`$(\mathcal{F}, \mathcal{R})$ // **Applying Definition 7 and 8 in [9]**
while $U \neq \emptyset$ **do**
  `select` $\mathcal{R} \in$ `U`, and $U \leftarrow U \setminus \{\mathcal{R}\}$
  $\mathcal{M}^{\mathcal{F}}[\mathcal{R}] \leftarrow$ `restrict`$(\mathtt{M}^{\mathcal{F}}, \mathcal{R})$ // **Applying Definition 12 in [9]**
  for $\varphi_i \in \Phi_{\mathcal{F}}$ **do**
    $\sigma_{min},\ min \leftarrow$ `solveMinMDP`$(\mathtt{M}^{\mathcal{F}}[\mathcal{R}], \varphi_{\mathtt{i}})$
    $\sigma_{max},\ max \leftarrow$ `solveMaxMDP`$(\mathtt{M}^{\mathcal{F}}[\mathcal{R}], \varphi_{\mathtt{i}})$
    `sat`, $\sigma \leftarrow$ `checkResult`(`min`, `max`, φ_i)
    if `sat` **then**
      if $i = M - 1$ **then return** `any`$(\mathcal{R})$ **else** `continue`
    end if
    if $\neg$`sat` **then**
      break
    end if
    if $\exists\, r \in \mathcal{R}.$ `isConsistentScheduler`(σ) **then**
      return r
    end if
    $U \leftarrow U \cup$ `split`$(\mathcal{R}, \mathtt{min}, \mathtt{max}, \sigma_{\mathtt{min}}, \sigma_{\mathtt{max}})$ // **A comprehensive info in [9]**
  end for
end while
return `UNSAT`

4.2 Optimal Synthesis

Optimal synthesis is designed similarly to the *feasibility* synthesis, with one significant difference. An *optimising* property represents the given *optimal* property, and its threshold is updated whenever satisfactory realisation is found. The goal is to exclude this solution from the searched state space. For instance, this update translates to decreasing the threshold of the minimising property when the *minimal* synthesis considered. The optimal synthesis yield the optimal solution when all family members were explored.

CEGIS. The algorithm to perform the *CEGIS* loop for minimal synthesis takes as the input a family of MCs $\mathcal{F} = (S, s_0, K, V, \mathcal{B})$, represented with realisations set $\mathcal{R} \subseteq \overline{\mathcal{R}}$, and the given minimisation specification φ_{min}. This approach synthesises a realisation $r^* \in \mathcal{R}$ minimising the satisfaction probability $\mathtt{min}^*$ of the given minimal objective. Initially, the algorithm sets the corresponding threshold of minimal objective concerning its settings and pick any realisation $r \in \mathcal{R}$. Subsequently, it builds the corresponding Markov chain $\mathcal{D}_r$ and performs the model checking against to given objective φ_{min}.

We then check whether $\mathcal{D}_r \models \varphi_{min}$ and when yes, we moreover check whether the obtained quantitative `result` is less than currently found minimum $\texttt{min}^*$. We update the current optimal realisation $\texttt{r}^*$ and corresponding minimal value $\texttt{min}^*$ according to the currently analysed realisation in such a situation. Moreover, we update the threshold of the *minimising* property represented the given minimal objective according to the newest one. We do this to not analyse for realisations worse than the current one in subsequent iterations but to prune them by constructing counter-examples.

Otherwise, when $\mathcal{D}_r \not\models \Phi_{min}$ or the current result is not better than the current minimum, we compute a critical set C for $\mathcal{D}_r$ and φ_{min}. This critical set C represents a fragment of the whole state space S of the analysed family $\mathcal{F}$. Subsequently, the algorithm constructs the sub-system $\mathcal{D}\downarrow C$ as a counter-example concerning computed critical set C. We then subtract these constructed counter-examples from the set $\mathcal{F}$ of candidate solutions. We repeat this procedure until either the whole state space is exhausted and at the end returns the found minimising realisation $\texttt{r}^*$ and corresponding value $\texttt{min}^*$. This approach is summarised in Algorithm 5.

Algorithm 4 CEGIS loop: Minimality synthesis.

Input: A family $\mathcal{F}$ of MCs with the set $\mathcal{R} \subseteq \overline{\mathcal{R}}$ of realisations, and a property φ_{min}.
Output: A realisation $r^* \in \mathcal{R}$ according to Definition 15.

$\texttt{min}^* \leftarrow 0$, $\texttt{r}^* \leftarrow \emptyset$
$\varphi_{min} \leftarrow \texttt{setThreshold}(\texttt{min}^*)$
while $\mathcal{R} \neq \emptyset$ **do**
  $r \leftarrow \texttt{any}(\mathcal{R})$
  $\mathcal{D}_r \leftarrow \texttt{buildDTMC}(\mathcal{R}, \texttt{r})$
  $result \leftarrow \texttt{solveDTMC}(\mathcal{D}_\texttt{r}, \varphi_\texttt{min})$
  if $result < min^*$ **then**
    $\texttt{r}^* \leftarrow r$, $\texttt{min}^* \leftarrow min$
    $\varphi_{min} \leftarrow \texttt{setThreshold}(\texttt{min}^* - \texttt{min}^* \cdot \varepsilon)$
  else
    $\mathcal{R} \leftarrow \mathcal{R} \setminus \texttt{constructCE}(\mathcal{D}_r, \varphi_{min})$
  end if
end while
return $r*$, min^*

Abstraction Refinement. We illustrate the *minimality* synthesis process within the AR loop by Algorithm 5. We remind that the synthesis target is to find realisation $r^* \in \overline{\mathcal{R}}$ that minimises the satisfaction probability $\texttt{min}^*$ of the minimal objective. A set U serves to store sub-families of the given family of realisations $\mathcal{R}$ that have not been yet analysed. The algorithm starts with building the quotient MDP for the whole analysed family $\mathcal{F}$ concerning the current realisations set $\mathcal{R} \subseteq \overline{\mathcal{R}}$. Subsequently, it restricts the set of realisations to obtain the corresponding sub-family for every $\mathcal{R} \in U$, concerning the quotient MDP $\mathcal{M}^{\mathcal{F}}$. Then, the algorithm continues with running the standard MDP model checking to compute the minimal and maximal probability and corresponding schedulers, respectively.

The main difference between the AR loop when feasibility synthesis is performed and now is the interpretation of the underlying MDP model checking results. The algorithm can discard $\mathcal{R}$ when the `min` probability for $\mathcal{R}$ is above $\texttt{min}^*$. Otherwise, it is required to check whether the corresponding scheduler σ_{min} is *consistent* or not. When it is consistent, we

can discard $\mathcal{R}$ and updated the current $\texttt{min}^*$ and σ_{min}, respectively. When the scheduler is not consistent but $\texttt{max} < \texttt{min}^*$ is valid, we can still update the current $\texttt{min}^*$ and enhance the pruning process. In such a situation, some realisations in $\mathcal{R}$ induce a lower probability than current $\texttt{min}^*$, but actually, we do not know which ones. However, when the scheduler is not consistent, regardless of whether $\texttt{min}^*$ has been updated, the algorithm has to split the analysed realisations set $\mathcal{R}$. Consequently, the algorithm subsequently has to analyse the derive sub-families since they can cover the minimising realisation.

Whenever the set U is empty, so all sub-families have been analysed, the algorithm terminates and yields the found optimal realisation r^*. This realisation is obtained by applying the function `applyScheduler` concerning Definition 5. The termination is guaranteed since only a finite number of subfamilies realisations has to be analysed. We summarise this procedure in Algorithm 5.

Algorithm 5 AR loop: Minimality synthesis.

Input: A family $\mathcal{F}$ of MCs with the set $\mathcal{R} \subseteq \overline{\mathcal{R}}$ of realisations, and a property φ_{min}.
Output: A realisation $r^* \in \mathcal{R}$ according to Definition 15.

$min^* \leftarrow 0$, $U \leftarrow \{\mathcal{R}\}$
$\mathcal{M}^{\mathcal{F}} \leftarrow \texttt{buildQuotientMDP}(\mathcal{F}, \mathcal{R})$ **// Applying Definition 7 and 8 in [9]**
while $U \neq \emptyset$ **do**
  `select` $\mathcal{R} \in \texttt{U}$, and $U \leftarrow U \setminus \{\mathcal{R}\}$
  $\mathcal{M}^{\mathcal{F}}[\mathcal{R}] \leftarrow \texttt{restrict}(\texttt{M}^{\mathcal{F}}, \mathcal{R})$ **// Applying Definition 12 in [9]**
  $\sigma_{min},\ min \leftarrow \texttt{solveMinMDP}(\texttt{M}^{\mathcal{F}}[\mathcal{R}], \varphi_{\texttt{min}})$
  $\sigma_{max},\ max \leftarrow \texttt{solveMaxMDP}(\texttt{M}^{\mathcal{F}}[\mathcal{R}], \varphi_{\texttt{min}})$
  if $min < min^*$ **then**
    if $\texttt{isConsistentScheduler}(\sigma_{\texttt{min}})$ **then**
      $\sigma^* \leftarrow \sigma_{min}$, $min^* \leftarrow min$
    else
      if $max < min^*$ **then**
        $min^* \leftarrow min$
      end if
      $U \leftarrow U \cup \texttt{split}(\mathcal{R}, \texttt{min}, \texttt{max}, \sigma_{\texttt{min}}, \sigma_{\texttt{max}})$ **// A comprehensive info in [9]**
    end if
  end if
end while
return $\texttt{applyScheduler}(\mathcal{R}, \sigma^*)$

4.3 Combined Probabilistic Synthesis

We present the main ideas behind a novel adaptation of the hybrid methods towards the *parameter* and *combined* synthesis. Recall that we need to operate with parameters having continuous domains and affecting the transition probabilities of Markov chains. We build on a *parameter substitution* [34] that replaces each parametric transition by two non-deterministic choices corresponding to the two extremes of its domain. Applying this operation to a parametric MC or parametric MDP, respectively, yields a parameter-free MDP, or a parameter-free stochastic game, respectively. These models can be subsequently verified using the off-the-shelf, efficient algorithms for model checking. Consequently, this technique boils down to verify an MDP and avoids computing of the rational functions and

SMT solving. The attractiveness of this technique is that it can be applied to parametric MC and parametric MDPs without much further trouble and exhausting analysis.

CEGIS. An inductive oracle adapts the computational environment in the following way. A family of pMCs $\mathcal{F} = (S, s_0, K, V, \mathcal{B})$ and the given set of specifications Φ represent the input to this method. The family is encoded as SAT formula concerning both kinds of parameters, where the parameters affecting topology from the set K are encoded as the list of possible options that they can have. On the contrary, the encoding of the parameters representing probability transition from the set V must be adapted and encoded as the individual interval for each such parameter. We remind each *CEGIS* iteration then analyses a concrete family member represented by an MC built concerning the assignment of the parameters yielded by the SAT solver.

When the analysed family member does not satisfy some specification $\varphi \in \Phi$, the construction of counter-examples is triggered, which yields a conflict set with generalised parameters. When the parameter is generalised, it does not matter its value within the whole sub-family, and all its options, or interval, respectively, can be excluded. Constructing counter-examples for families including continuous parameters requires additional reasoning as excluding the concrete value from the continuous interval is not feasible. Therefore, we design a strategy trying to exclude a specific part of the interval, which preserves the CE generalisation. For each such parameter, we construct an appropriate neighbourhood of the value in the current assignment – this leads to an MDP model. When the analysis of this MDP yields unsatisfiable results, these neighbourhoods can be safely excluded from the interval and thus from the family. Otherwise, the neighbourhoods are gradually refined until the unsatisfiable result is obtained.

Algorithm 6 CEGIS loop: CE generalisation for continuous parameters from a set V.

Input: An pMC $\mathcal{D}$ and a set of specifications $\Phi_\neg$ s.t. $\forall \varphi_\neg \in \Phi_\neg.\ \mathcal{D} \not\models \varphi_\neg$.
Output: A well-defined region r s.t. $\forall \varphi_\neg \in \Phi_\neg.\ \mathcal{D}, r \not\models \varphi_\neg$.

1: $r \leftarrow \{u(v_i) - \varepsilon, u(v_i) + \varepsilon \mid \forall\, v_i \in V.\ u(v_i) \in I(v_i)\ \wedge\ u(v_i) = \mathcal{D}[v_i]\}$
2: $\mathcal{D}^r_{sub} \leftarrow$ *substitution-pMC*$(\mathcal{D}, r)$ // **Applying Definition 9**
3: $\texttt{sat} \leftarrow \texttt{solveMDP}(\mathcal{D}^{\texttt{r}}_{\texttt{sub}}, \Phi_\neg)$ // **Applying Theorem 2**
4: **while** sat **do** // $\exists \varphi_\neg \in \Phi_\neg.\ \mathcal{D}^r_{sub} \models \varphi_\neg$
5: $\quad r \leftarrow \{u(v_i) - \varepsilon_\downarrow, u(v_i) + \varepsilon_\downarrow \mid \forall\, v_i \in V.\ u(v_i) \in I(v_i) \wedge u(v_i) = \mathcal{D}[v_i]\} \wedge i = \texttt{iter}\,\%\,n\}$
6: $\quad \mathcal{D}^r_{sub} \leftarrow$ *substitution-pMC*$(\mathcal{D}, r)$
7: $\quad \texttt{sat} \leftarrow \texttt{solveMDP}(\mathcal{D}^{\texttt{r}}_{\texttt{sub}}, \Phi_\neg)$
8: **end while**
9: **return** r

Algorithm 6 illustrates described strategy to exclude a specific part of the parameter interval. It takes a concrete family member from the currently analysed family of pMCs $\mathcal{F}$ represented as a pMC $\mathcal{D}$, and a set of specifications $\Phi_\neg$ that the given family member does not satisfy. First, we construct the region concerning the parameter assignment of the analysed family member, where each parameter $v_i \in V$ represents the neighbourhood of its valuation $u(v_i)$ created according to the current ε value. Subsequently, we construct an MDP $\mathcal{D}^r_{sub}$ representing a substitution of pMC concerning the original model $\mathcal{D}$ and constructed region r. Then we deduce whether $\mathcal{D}, r \not\models \varphi_\neg$ for each $\varphi_\neg \in \Phi_\neg$ by applying the standard technique for MDP model checking. When this does not hold for some specification, we refine region r by reducing the neighbourhood of one parameter v_i which

is selected concerning the current iteration number i. In this way, we may perform more iteration of MDP model checking, but we will avoid unnecessarily shrinking of regions and exclude the most extensive possible region.

Abstraction Refinement. We remind that the *abstraction refinement* loop analyses a whole family $\mathcal{R} \subseteq \overline{\mathcal{R}}$ represented as a pMDP model $\mathcal{M}$ against to given set of specifications Φ at once. In addition, a well-defined region r is given which serves as parameter space. Since the loop takes at the input a pMDP $\mathcal{M}$ and wants to analyse it, we need to construct a parameter substitution $\mathcal{M}^r_{sub}$ of the given MDP model $\mathcal{M}$ concerning the parameter region r (applying Definition 10). Since the topology of the substituted model $\mathcal{M}^r_{sub} \models \varphi$ is independent of the region, for checking multiple regions, we only substitute the probabilities according to the region of interest. This parameter substitution yields a stochastic game, and by analysing it, we can derive whether holds that $\mathcal{M}^r_{sub} \models \varphi$ for each $\varphi \in \Phi$ (see Theorem 3).

This regional model checking (see Section 2.2) of the constructed stochastic game yields the refinable lower and upper bounds on probabilities correspond to the current parameter space intervals. Abstraction refinement oracle works without the change in the following processing than presented loop for topology synthesis, except refinement strategy when the analysis result is inconclusive. A refinement strategy that mitigates the overhead of analysing sub-families, presented in [9], is not directly applicable to this type of task. Therefore, we design a *greedy splitting* strategy based on the domain size, which does not analyse the MDP structure, and performs as follows.

This strategy selects from the parameters set $K \cup V$, which includes both kinds of parameters,the parameter with the greatest number of possible options or parameter containing the longest interval, respectively. The domain of the selected parameter is splitting in half, and the analysed family is refined according to that. We note that parameter space partitioning is applicable only when parts of a region are well-defined, as the well-definedness of a region is effectively decidable. Other (sub-)regions can simply be tagged as not well-defined treated as being inconclusive. This naive strategy offers space for improvement in the future, and it is in our plans to improve it concerning the model of the analysed family.

Since the considered parameter spaces are continuous, they can be potentially parted infinitely many times in case of inconclusive results. Therefore, we need to introduce a termination condition that decides that the interval will no longer be divided. In our case, we construct the termination condition based on the results of MDP model checking. The MDP model checking is performed by Storm, which has been defined the accuracy with which it performs iterative methods or with what accuracy it returns results, respectively. We set a threshold whose value is reduced ten times concerning this accuracy. The family with the inconclusive result, but with an absolute difference of MDP results for some specification less than the threshold, will be not subsequently split and not further analysed. At the end of the synthesis, we will provide to user the information on how many such families have been discarded and, if necessary, can increase the accuracy and repeat the synthesis.

Chapter 5

PAYNT

PAYNT[1] (Probabilistic prograAm sYNThesiser) is a tool to synthesise probabilistic programs automatically. PAYNT supports the synthesis of finite-state programs and the specifications given as a conjunction of temporal logic constraints, possibly including an optimal objective. The input is a program sketch that concisely describes a finite family of (finite) Markov chains, and a specification. The tool can then identify a family member that (potentially optimally) satisfies given specifications. Figure 5.1 depicts the workflow of PAYNT.

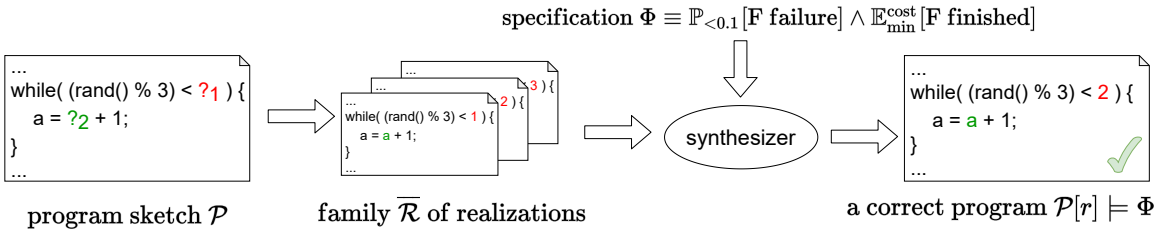


Figure 5.1: The workflow of the synthesis process within PAYNT.

5.1 Architecture

The design of PAYNT is based on an oracle-guided synthesis [2] enabling a flexible combination and integration of a variety of state-of-the-art synthesis and verification algorithms. In particular, it implements a hybrid synthesis approach leveraging both the *CEGIS* and the *AR* oracle. PAYNT is able to efficiently synthesise the program topology (*topology* synthesis) as well as continuous parameters affecting the transition probabilities (*parameter* synthesis). Moreover, it can handle sketches including both types of synthesis problems, so-called a *combined* synthesis, which is a unique feature compering to the existing tools.

To achieve a high-performance synthesis, PAYNT is implemented on top of the probabilistic model checker Storm [16], providing optimised verification procedures. It is implemented in a modular fashion on top of a python API to provide flexibility. PAYNT allows to define of the program sketches and provides all baseline synthesis algorithms under one roof. The tool finds application primarily for two groups of users. First, the analysis of

[1] https://github.com/gargantophob/synthesis

realisations sets is a useful back-end for automated system design. For instance, this can be used when synthesising finite-state controllers for partially observable Markov decision processes (POMDPs), synthesising a network protocol to increase the packet throughput, or selecting the optimal power management strategy. Second, PAYNT provides a modular development platform for automated design of probabilistic programs.

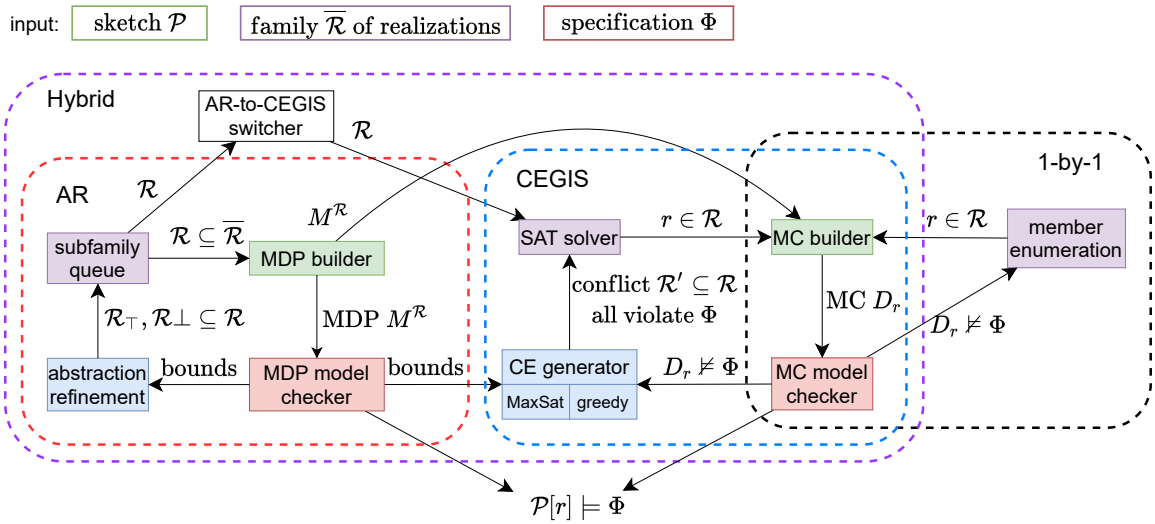


Figure 5.2: The architecture of tool PAYNT.

PAYNT's architecture (see Figure 5.2) consists of model checkers, modules to build models and components for family handling. The family handlers store the information about the already covered design space of the analysed family. As the name suggests, the member enumeration unit iterates over all family members. SAT solver maintains an SAT-formula describing undiscovered realisations, and subsequent, it uses Z3 SMT-solver to obtain the next candidate realisation. The queue with sub-families contains a collection of unexplored sub-families refined as hyper-rectangles when the analysis is inconclusive. The model builders take a specific input according to the active oracle and produce the relevant model representation: the CE-oracle sends to the builder a single realisation r while the AR-oracle sends a realisations set $\mathcal{R}' \subseteq \mathcal{R}$. The model checkers verify whether the constructed model (MCs or MDPs) satisfies the given specification. When analysing MDP, lower and upper bounds on satisfiability probabilities are provided. Last but not least, PAYNT provides a module for generating counterexamples. In particular, it implements two approaches: a greedy state-expansion and a MaxSat approach.

Implementation Frame. PAYNT takes as the input a sketch written in the Jani or Prism language and a set of temporal properties expressed using the PRISM syntax. PAYNT is implemented on top of a modern probabilistic model checker Storm [16] providing high-performance verification procedures implemented in C++. Further, it uses Z3 theorem prover for SMT-solving and a Python API for flexible implementation of the synthesis loop itself. The implementation of PAYNT is composed of *30* Python modules containing *7k* source lines of code. We consider only our implementation and do not include extensions contributed to Storm and its Python API, invoked by PAYNT. We tests the specific components with unit tests to maintain their correct functionality. Regression

tests verify the accuracy and correctness of the synthesis results and these tests currently cover more than *90%* of the source code lines.

5.2 Prism Sketch Language

As we said above, PAYNT takes as the input the sketch written in the PRISM [25] or JANI [5] language. These high-level programming languages to describe probabilistic systems are more advantageous than modelling the system as a Markov Chain. The state explosion problem, arising when using the Markov chains as an operational model, renders this approach unusable. STORM parses the high-level description (sketch) written in these languages and constructs the corresponding Markov chain, which the individual synthesis methods analyse. Now, we briefly introduce the PRISM sketching language proposed in [8].

A program written in PRISM language consists of modules that interact with between itself. We consider only programs with a single module since more modules can be transformed into one model program. A module state space is given by the set of (bounded) variables, with their initial values and the set of guarded commands describing the transitions between module states. The commands have the following form:

$$\texttt{[action] guard} \;\rightarrow\; p_1 : \texttt{update}_1 + \cdots + p_n : \texttt{update}_n$$

The *actions* ensure the synchronisation between two or more modules when they perform the command. The *guard* represents a boolean expression over the module variables. An update of the variables is selected concerning the probability distribution defined by expressions p_1 through p_n when guard evaluates to true. Essentially, the guard identifies states for which this command is applicable, and updates describe successor states and the probability distribution over these successors. Overlapping of guards is disallowed since it yields non-determinism.

Moreover, a *sketch* describes a program containing holes representing undefined program parts that should be filled with the value from the finite options set. These holes are declared as follows:

$$\texttt{type hole } h \texttt{ in } \{\texttt{expr}_1, \ldots, \texttt{expr}_k\}$$

The `type` represents the domain of `hole` h and it is an `integer` or `float`. The hole identifier is given by h, and the set of expressions $\texttt{expr}_i$ is defined over the program variables. Commands and variable declarations use a `hole` as a component of an `update expression` or a `guard`. Such a program sketch represents a system description with a specified general structure but with some concrete details left out. Instantiating all holes yields a specific program, and the synthesiser target is to resolve how to substitute all holes with their options to satisfy a given specification. PRISM sketching language also provides several extra functionalities: specification of the constraints to hole values and assigning costs to option holes, and many others. For more detailed information, please refer to [8].

Example 10 (PRISM Program Sketch)**.** For instance, we introduce the following PRISM program sketch P:

```
int hole X in {0,1}
int hole Y in {1,2,3}
module m
  int s: [0..4] init X+1
  [] s < Y → 0.75 : (s' = max(s-X, 0)) + 0.25 : (s' = s+X);
  [] s = Y → 0.50 : (s' = s-1) + 0.50 : (s' = s+1);
  [] s > Y → 0.25 : (s' = s-1) + 0.75 : (s' = min(s+1, 4));
end module
```

The module `m` can be in one of five states: $\{0, 1, 2, 3, 4\}$. When the module state `s` is equal to the value of hole `Y` (`s = Y`), it will move to the nearest previous (`s' = s-1`) or next (`s' = s+1`) state with the same probability of `0.50`. When the module state `s` is above the value of hole `Y` (`s > Y`), it will move to the left neighbour with the probability of `0.75` and with remaining to the right if it exists, or stay in the last state (`s=4`). Otherwise (`s < Y`), the model will move the same as in the previous case (`X=1`), or it stays in the current state without change (`X=0`). When we consider the following instantiating of the holes `X=1` and `Y=2`, the corresponding underlying Markov chain is depicted below. Figure 5.3 depicts described underlying MC.

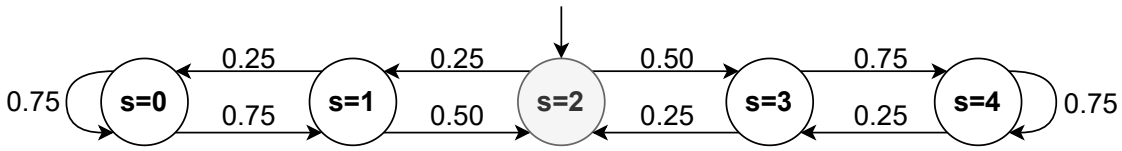


Figure 5.3: An underlying Markov Chain of a probabilistic program P from Example 10.

5.3 Usage of PAYNT

We demonstrate the usage of PAYNT on a synthesis problem considering a simple server for request processing depicted in Figure 5.4. Requests are produced by an external unit within random intervals and maintained in a request queue with capacity Q_{max}. If the queue is full arriving requests are lost. The server can operate in three modes having a different power consumption – *active*, *idle* and *sleeping*. The server process the requests only in the *active* state. When the server switches from a state with low energy into a state with higher, extra energy is consumed, and random latency is required. We note that the power consumption of request processing depends on the current size of the queue. The server operation time is finite but given as a random process.

The synthesis aims to construct a unit called *power manager* (PM), which controls the server. PM observes the current queue size and, according to it, then sets the relevant power profile. Precisely, it differentiates between four queue occupancy levels determined by the threshold levels T_1, T_2, and T_3. They indicate which fraction of the queue capacity

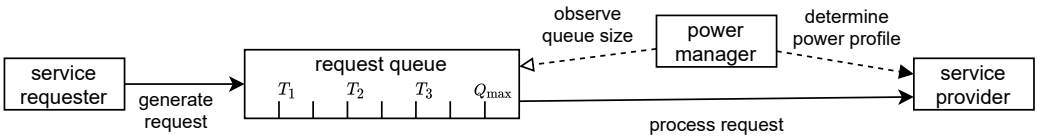


Figure 5.4: The server for request processing within *DPM* case study.

is currently occupied, and they are entered into the model as unknown parameters. Since the model considers three levels, then the power manager observers the queue occupancy on the following intervals: $[0, T_1]$, $(T_1, T_2]$, $(T_2, T_3]$, $(T_3, 1)$. Moreover, it considers a single power profile $P_1, \ldots, P_4 \in \{0, 1, 2\}$ for each occupancy level. The power profile's current value represents the server's mode, so the set $\{0, 1, 2\}$ encodes available modes sleeping, idle and active in the given order. The following sketch describes the module of *PM*:

```
module PM
    pm  :  [0..2] init 0; // 0 - sleep, 1 - idle, 2 - active
    [sync0] q <= T1*QMAX -> (pm'=P1);
    [sync0] q > T1*QMAX & q <= T2*QMAX -> (pm'=P2);
    [sync0] q > T2*QMAX & q <= T3*QMAX -> (pm'=P3);
    [sync0] q > T3*QMAX -> (pm'=P4);
endmodule
```

In this model, we consider the following parameters and their domains: the queue capacity $Q_{\max} \in \{1, 2, \ldots, 10\}$, the power profiles $P_1, \ldots, P_4 \in \{0, 1, 2\}$ and the thresholds $T_1 \in \{0.0, 0.1, 0.2, 0.3, 0.4\}$, $T_2 \in \{0.5\}$, $T_3 \in \{0.6, 0.7, 0.8, 0.9\}$. The final sketch describing this considered model forms a design space of $16,200$ various power managers. The average size of the Markov chains in this models is around 900 states. The synthesis target is to find the specific power manager, i.e., the holes instantiation, that minimises power consumption while the expected number of lost requests during the server's operation time is below 1. We can formalise these requirements as a pair of temporal logic formulae in the Prism language:

```
R{"lost"} <= 1 [ F "finished" ]
R{"power"}min=? [ F "finished" ]
```

PAYNT explores the design space and produces the following output, including the parameter assignment, and the quality value corresponds to Φ of the analysed program:

```
QMAX=5,T1=0,T3=0.7,P1=1,P2=2,P3=2,P4=2
R[exp]{"lost"}=0.6823 [F "finished"]
R[exp]{"power"}min=9100 [F "finished"]
```

The synthesised power manager, optimal wrt. to the specified properties, has the queue capacity set to 5 and individual thresholds at values $1 = 0.0$, $2 = \lfloor 5 \cdot 0.5 \rfloor$ and $3 = \lfloor 5 \cdot 0.7 \rfloor$. The synthesised power manager performs the following strategy. When the request queue is empty, then the power manager maintains an idle state. Otherwise, it always maintains an active state, regardless of the exact size of the queue, and is never in the sleeping state. The synthesised solution has a power consumption of $9,100$ units and the expected number of lost requests of $\approx 0.68 < 1$.

PAYNT computes an optimal solution in one minute. Although this synthesis problem is quite simple (recall it includes only 16k candidates), PAYNT is already 3× faster than

a naive enumeration of all realisations. Further, we explored a more complex variant of this problem inspired by a dynamical power manager's known model for complex electronic systems. The synthesis problem is described by the sketch, which covers a family around 43M realisations. PAYNT solves this synthesis problem within 10 hours, whereas the naive enumeration takes more than one month.

Combined Synthesis. Further, we have expanded this model to create a case study for combined synthesis as well. Because of this, we had to find some probabilistic transitions in the system template, which we omit and let them synthesize. Therefore, let us first consider the following sketch described the module of a service provider (SP):

```
module PM
    // waiting states: 3 - sleep to idle, 4 - idle to active
    sp  :  [0..4] init 0; // 0 - sleep, 1 - idle, 2 - active
    [tick1] sp <= 2 & pm <= sp -> (sp'=pm);
    [tick1] sp <= 2 & pm > sp -> (sp'=sp+3);
    [tick1] sp = 3 -> 0.8 : (sp'=sp-2) + 0.2 : true;
    [tick1] sp = 4 -> 0.6 : (sp'=sp-2) + 0.4 : true;
endmodule
```

We can see that there are states where SP has no control, so it contains transient states ($\texttt{sp} = 3$ and $\texttt{sp} = 4$). Such transient states have been introduced since a time resolution has been considered when modelling this system to correctly model transitions with delays longer than this time resolution. Thus, these states are used to model the non-unitary time transition between the states of SP. We decided to parameterise the probabilistic transitions of these transient states, which allow us to think about different time resolutions. Therefore, we consider the following changes to the SP template, including two parameters p_0 and p_1 that affect transition probabilities:

```
    [tick1] sp = 3 -> p_0 : (sp'=sp-2) + (1-p0) : true;
    [tick1] sp = 4 -> p_1 : (sp'=sp-2) + (1-p1) : true;
```

Note that larger values of these parameters represent a lower time resolution because the system will switch earlier between individual states. At the same time, a lower resolution time will also ensure lower power consumption. For these parameters, we consider intervals of length 0.2: $p_0 \in [0.7, 0.9]$ and $p_1 \in [0.6, 0.8]$. The synthesis target remained unchanged, and we are still looking for a specific power manager that minimises power consumption, and the expected number of lost requests is below one.

We recall that the considered family includes $16k$ parametric candidates compared to the previous task of topology synthesis, which represents a much more difficult task due to the continuous parameters. PAYNT synthesised an optimal solution in 2.8 hours, and the synthesised strategy remains unchanged compared to the presented above. For the sake of completeness, we will only add that the upper limits of the interval have been set behind the values of the parameters ($p_0 \approx 0.90$ and $p_1 \approx 0.8$), which ensure the lowest power consumption in the lowest time resolution. We emphasise that we have also demonstrated the correctness of the design of our implementation for combined synthesis with this result.

Chapter 6

Experimental Evaluation

This chapter will investigate the performance of the *hybrid* approach for integrated synthesis methods extensions – multi-property and optimal synthesis – for various case studies and properties. Moreover, we will also introduce a preliminary results of the performance of the designed approach for both *parameter* and *combined* synthesis. Further on, we demonstrate the applicability of PAYNT and interpret the synthesis results for three of these case studies. All experiments are run on a Debian GNU/Linux 10 machine with Intel(R) Core(TM) i7-3770K (8 cores at 3.50GHz) and using up to 32 GB RAM, with all the algorithms being executed single-threaded.

6.1 Performance Evaluation of Advanced Methods

This section aims to demonstrate the performance of advanced integrated methods for both *multi-property* and *optimal* synthesis on various synthesis problems from different application domains. In particular, we compare its performance with the *one-by-one* enumeration representing the baseline algorithm implemented in the existing synthesis tools such as QFLan [39] and ProFeat [11]. Experiments in previous papers [9, 8] have shown that the synthesis methods implemented in these tools have evident deficits on the investigated benchmarks. Moreover, experiments in [1] have shown that presented hybrid method significantly outperform both state-of-the art synthesis approaches – CEGIS and AR. These facts is supported by comparing the hybrid approach only with the one-by-one enumeration we present in this section. For each case study, we report the results for *unfeasibility* problems with one property and two properties, and *optimal* synthesis problem and its *relative* variant with $\varepsilon = 0.05$, which find the optimal value for a specific objective. In all cases, the synthesis methods have to explore the whole design space to prove unsatisfiability and optimality. The metrics marks with * represent the qualified estimates. We consider the following case studies:

Herman. This model represents distributed asynchronous protocol for rings with self-stabilisation [19, 24]. The protocol is extended with a choice for flipping several unfair coins, and each station in the ring includes one-bit memory. All stations in the ring have equivalent behaviour, i.e., they are anonymous, but they decide based on the own local events and value of the memory. The family maintains this anonymity and describes the different variations of coin choice and updates of memory. We are interested mainly in the expected time until *stabilisation*.

	1 property	2 properties	optimal	5%-optimal
1-by-1	32h	40h	32h	–
Hybrid	90s	105s	21m	8m

Table 6.1: Herman: Family size: $3.1M$, Number of parameters: 7, Average MC size: $1.1k$.

DPM. It represents a partial information scheduler for a disk power manager motivated by [31]. This model have been precise described in Section 5.3. We are interested primarily in the expected energy consumption and expected number of lost requests.

	1 property	2 properties	optimal	5%-optimal
1-by-1	31d	35d	31d	–
Hybrid	72m	84m	9.7h	6.2h

Table 6.2: DPM: Family size: $43M$, Number of parameters: 16, Average MC size: $3.6k$.

Maze. It represents a planning problem typically modelled as POMDP [30]. The family includes all deterministic strategies which are based on observation, and containing small memory with a fixed upper bound. We are interested primarily in the expected time to the *goal*.

	1 property	2 properties	optimal	5%-optimal
1-by-1	25h	31h	25h	–
Hybrid	63m	65m	78m	59m

Table 6.3: Maze: Family size: $9.4M$, Number of parameters: 22, Average MC size: $0.2k$.

Pole. It models balancing a pole in an unknown and noisy environment [3]. A model controller optimises an expected behaviour during the deployment without dependence on the current (hidden) environment since it is preferred before the finite set of environment behaviours. The family described schedulers that are independent of hidden information. We are interested mainly in the expected time until *failure*.

	1 property	2 properties	optimal	5%-optimal
1-by-1	23h	30h	23h	–
Hybrid	7s	8s	11m	60s

Table 6.4: Pole: Family size: $1.3M$, Number of parameters: 17, Average MC size: $5.6k$.

Grid. It represents a model describing again partially observable MDPs (POMDPs) [23]. There is an agent who tries to locate a target cell in a 4x4 grid. We are interested in lower- and upper- bounded properties on the expected number of steps.

	1 property	2 properties	optimal	5%-optimal
1-by-1	16m	19m	16m	–
Hybrid	31s	35s	47s	9s

Table 6.5: Grid: Family size: $65k$, Number of parameters: 8, Average MC size: $1.2k$.

Evaluation. As we have already said, we are based on what has already been shown in previous articles [1, 9, 8]. As a basis, we know that a hybrid oracle is orders of magnitude faster than one-by-one enumeration when analysing single property. Based on the results obtained in the framework of all considered case studies, we can draw the following conclusions, which confirm the tables above. An integrated *multi-property* synthesis slows down both approaches, although the hybrid slowdown is almost negligible. An *optimal* synthesis slows down only the hybrid approach, yet it is still incomparably faster than naive one-by-one enumeration. Moreover, a hybrid oracle supports ε-optimal synthesis, which is even faster.

6.2 Performance Evaluation of Combined Synthesis

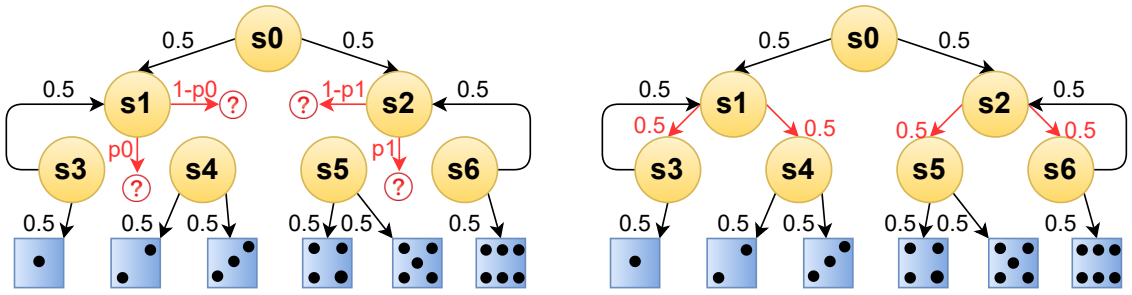
This section aims to demonstrate the performance of the designed and implemented method for combined probabilistic synthesis. We remind that this synthesis problem included both kinds of parameters, those that affect the topology of MCs and those that affect the transition probabilities of MCs. We only present the results achieved by our tool PAYNT, as there is no other tool that would support this combined synthesis of probabilistic programs. For each considered case study, we report the results for *optimal synthesis* and its *relative* variant with $\varepsilon = 0.05$, as in the previous section. Such a task representing the optimal synthesis is the most advanced and, therefore, most suitable for measuring the performance of the designed method because it has to explore the whole design space. We consider the following case studies:

$\lvert\mathbf{K}\rvert$	$\lvert\mathbf{V}\rvert$	$\lvert\Phi\rvert$	$\lvert\mathbf{F}\rvert$	$\overline{\lvert\mathbf{MC}\rvert}$	$\varepsilon = \mathbf{0.0}$	$\varepsilon = \mathbf{0.05}$
4	2	7	$4k \times 2 \cdot \lvert 0.50\rvert$	13	8m	50s

Table 6.6: Knuth-Yao Die: Performance evaluation of combined probabilistic synthesis.

Knuth & Yao Die. This case study considers a probabilistic program modelling a fair dice using only fair coins. the program starts at the initial state s_0 and repeatedly tosses a coin. Whenever tail appears, it takes the one branch, and when headers appear, another branch, and in this way it branching own state tree (see Figure 6.1). This continues until the value of the dice is decided. We consider six specifications representing the probability of reaching a final state s_7 (depicted as blue in Figure 6.1) with a specific dice value to prove that the program is correct. Further, we minimise the expected number of coin tosses to obtain the specific dice value. We consider a sketch with four parameters affecting the model topology (depicted as (?) in Figure 6.1(a)) that have simultaneously undefined transition probabilities p_0 and p_1. Since each parameter affecting topology can represent arbitrary model state $s_0, \ldots, s_7$, we obtain the sketch describing of $8^4 = 4096$ various pMCs. The probability parameters p_0 and p_1 are defined over the interval $[0.0, 0.5]$ since they

cover the transitions asymmetrically. PAYNT synthesised an optimal solution depicted in Figure 6.1(b) in 8 minutes with $\varepsilon = 0.0$ and 50 seconds when computes with $\varepsilon = 0.05$. We have found that the minimum expected number of coin tosses is $\frac{11}{3}$. The synthesised strategy chooses for the searched states always states $s_3, \ldots, s_6$ in arbitrary order, and the transition probabilities representing the fair coin (≈ 0.5). We emphasise that with this case study, we have also verified the correctness of our implementation, as a result corresponds to what is expected.



(a) Parametric model of Knuth & Yao dice

(b) Synthesised model of Knuth & Yao dice

Figure 6.1: Models represent the state space of the program for the Knuth & Yao case study.

Synthesis progress. Now, we briefly describe the combined synthesis process of this considered case study. The *hybrid* method takes $7.4k$ *AR* and $3.6k$ *CEGIS* iterations to synthesise the presented optimal strategy. During these iterations, *AR* executed $44k$ MDP model checks and *CEGIS* $14k$ model checks of MCs. We remind that when *AR* comes across an unsatisfiable candidate against one specification, it discards this candidate and does not proceed further with its analysis. On the contrary, when the analysed candidate holds for some specification, it will not need to be further analysed against to satisfiable specifications in derived sub-families. *CEGIS* analyses a whole set of specifications in each iteration because whereas it seeks to cut through space with as many counter-examples as possible. These facts influence the individual numbers of performed model checks.

As we presented in Section 4.3, *AR* is firstly splitting the parameters with options (affecting topology) because the designed method chooses the parameter to split according to the number of options, or interval length, respectively. *AR* split four option parameters during the 617 iterations, and in the remaining, it chooses the probability parameters. Both these parameters are split $2.5k$ times within the *AR* loop, because they alternate between iterations due to the halving of the intervals. We note that the remaining $1.7k$ iterations of AR ended up dropping the family due to an unsatisfiable specification. Further, the *AR* analyses in $3.3k$ iterations the family with unfilled probability parameters, where the remaining parameters have the concrete value. *CEGIS* constructs the counter-examples in $7k$ cases and $4k$ times were generalised all parameters with options, and the conflict set includes both probabilistic parameters. In other $1.1k$ cases, the conflict involving only the first probability parameter p and remaining parameters were generalised. In the remaining cases, the conflict set also involving parameters affecting the topology, and thus a smaller number of generalised parameters.

NAND multiplexing. This case study concerns NAND multiplexing, a technique for constructing reliable computation from unreliable devices. It describes fault-tolerant hardware by having copies of a NAND unit all doing the same work [18]. In the considered model, we assumed that the inputs X and Y are identically distributed (with probability $prob1$ of being stimulated), and the NAND gates suffer from a von Neumann fault (inverted output) with probability $perr$. We verify one optimal target describing that the system will reach the target state, and the resulting value of gate will be correct. Parameters are the probabilities of the faultiness of the units ($prob1$) and the probabilities of erroneous inputs ($perr$). The intervals for these parameters are taken from the following paper [18]. We consider instance $(20, 13)$, where $N = 20$ represents the number of inputs in each bundle and $K = 13$ the number of restorative stages. We chose such an instance to get large MCs, as the considered family contains two probability parameters, and thus we obtained case study with different properties.

$\lvert\mathbf{K}\rvert$	$\lvert\mathbf{V}\rvert$	$\lvert\Phi\rvert$	$\lvert\mathbf{F}\rvert$	$\lvert\overline{\mathbf{MC}}\rvert$	$\varepsilon = \mathbf{0.05}$	$\varepsilon = \mathbf{0.15}$
0	2	1	$2 \cdot \lvert 0.12 \rvert$	1M	22.6h	1h

Table 6.7: NAND multiplexing: Performance evaluation of combined probabilistic synthesis.

DPM. This case study has been precisely introduced in Section 5.3. We consider the domains parameters representing queue capacity, four power profiles and three thresholds, and two probabilistic parameters representing time resolutions. Analysed family describes $16k$ parametric candidates that including two parameters with an interval length of 0.2. We are interested in the expected power consumption and expected number of lost requests.

$\lvert\mathbf{K}\rvert$	$\lvert\mathbf{V}\rvert$	$\lvert\Phi\rvert$	$\lvert\mathbf{F}\rvert$	$\lvert\overline{\mathbf{MC}}\rvert$	$\varepsilon = \mathbf{0.0}$	$\varepsilon = \mathbf{0.05}$
7	2	2	$16k \times 2 \cdot \lvert 0.20 \rvert$	400	2.8h	2.1h

Table 6.8: DPM: Performance evaluation of combined probabilistic synthesis.

Herman. This case study will be in more detail introduced and described in Section 6.3.2. Domains parameters represent the memory updates, and the probability parameters represent different coin biases based on the variable and memory value. The synthesis target is to identify the protocol version that minimises stabilisation time.

$\lvert\mathbf{K}\rvert$	$\lvert\mathbf{V}\rvert$	$\lvert\Phi\rvert$	$\lvert\mathbf{F}\rvert$	$\lvert\overline{\mathbf{MC}}\rvert$	$\varepsilon = \mathbf{0.0}$	$\varepsilon = \mathbf{0.05}$
3	4	1	$8 \times 4 \cdot \lvert 0.25 \rvert$	1.5k	20.2h	52m

Table 6.9: Herman: Performance evaluation of combined probabilistic synthesis.

Evaluation. The obtained results cannot be generalised because the complexity of the synthesis process in combined synthesis depends on several factors. The effectivity of splitting the space of candidate solutions is mainly influenced by the density of solutions in the vicinity of the optimal solution. If the density is high, the parameter intervals must reach a small

grain size for the result to be decisive, and the synthesis thus requires many more iterations. Of course, this property also affects the correlation between the synthesis run-times of optimal and ε-optimal synthesis. Other factors influencing the efficiency of the combined synthesis process also include the influence of parameters influencing the topology, the number of various parametric candidates, or the size of the corresponding parametric models.

6.3 Applicability

This section demonstrates the applicability of PAYNT and interprets the synthesis results for two of the considered case studies. Firstly, we briefly introduce the model, then describe which sketch is considered for the synthesis problem and against which specifications it will be synthesised, and finally, we demonstrate the synthesised result.

6.3.1 Maze

This synthesis problem considers an instance of POMDP synthesis of the controller. Figure 6.2 depicts a *maze* in which operates an agent starting at a random location. The agent has simple sensor detecting walls, but it cannot recognise cells within the maze that have walls at identical directions – e.g. cells 5 and 10, or 1 and 3. We denote such cells as observation-equivalent, and they are depicted in Figure 6.2 by the same colour. The agent performs the actions representing the movements in the four available directions corresponds to the cardinals. However, these movements are performed with some level of non-determinism, i.e., they are subject to a random error. For instance, when the agent wants to move to the west, there is a small probability that it will move to the east.

We design a sketch for the agent's controller that helps it reach the target cell representing the maze's exit – cell 12. The controller has one bit of memory available which is initially set to value 0. The holes in this sketch represent the available actions concerning the current observation. The direction for steering and setting a new value of the memory represents actions, and observation is represented by the current memory value and detected walls in the current cell. This sketch describes a family with $9.4M$ various programs. Our target is to synthesise a program that minimises the expected number of steps to achieve the maze's exit.

PAYNT explores the set of candidate programs in *one* hour, whereas the naive one-by-one enumeration takes more than one day. Figure 6.2 depicts the synthesised strategy of the selected controller, which is optimal given the expected number of steps to reach the exit. Arrows depict the direction of steering concerning the current memory value (number at the start of an arrow) and the corresponding memory update (number at the end of an arrow). For instance, when an agent is currently in cell 3, it moves to the east when the memory value is equal to 0 and moves to the west when the memory contains 0, while the value of the memory remains unchanged in both cases.

When an agent is in the cell with enclosing walls (cells 0, 2 and 4), it can navigate to the exit since it knows its current position within the maze. Likewise, when the agent navigates from cells 11 or 13, it selects always to the north to reach cells 0 or 4 eventually. Otherwise, when the agent is located in purple or orange cells, it must first determine its current position within the maze by moving in one direction. For instance, the agent deployed at cell 1 or 3 moves first to the west (recall that the memory is initially set to 0), where it reaches the cell to determine the current position. Similarly, when the agent deployed at cell 5-10, it moves to the north. In this observation group, it is indeed more

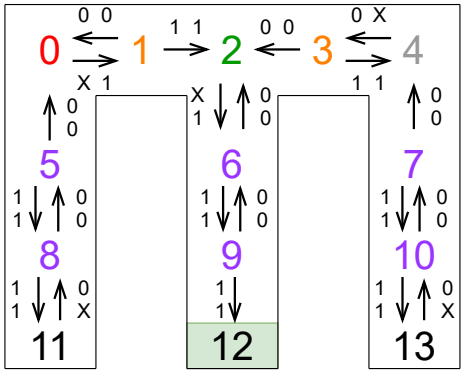


Figure 6.2: The state-space of the *Maze* model with a marked target state and the observation groups. A right side represents the found optimal strategy to reach the target location within the minimal expected time. The initial state is selected randomly with a memory value of 0.

beneficial to move to the north since the robot is twice as likely to be deployed when starting at locations 5/7/8/10 compared to locations 6 and 9.

The synthesised expected time to reach the maze's exit for obtained strategy is ≈ 9.80 steps. We note that this result cannot be improved by adding more bits to the controller's memory.

6.3.2 Herman

This model represents a unidirectional token ring containing an odd number of connected processes. Each process has a Boolean variable and access to its neighbour's variables. We say that a process holds a token if the value of its variable is equal to the neighbour. When only one token remains in the ring, such a situation is denoted a stable one, and all other situations are denoted as unstable. When the token ring passes from an unstable to stable configuration, we say a protocol is self-stabilising. An expected number of rounds to reach a stable configuration represents a stabilisation time, considered a performance metric.

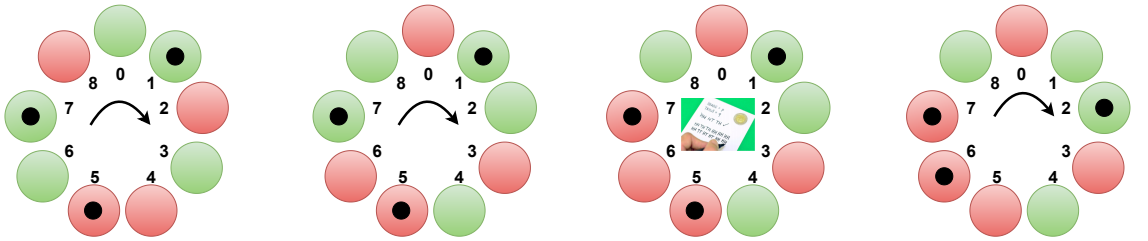


Figure 6.3: A round of Herman's protocol with nine processes within ring, where three processes holds a token in the current configuration.

We consider a version of *Herman's randomised self-stabilisation protocol* [19, 24]. All processes behave the same in this protocol, and therefore a stabilisation cannot be solved in a deterministic way in such anonymous networks. This protocol is synchronised, and in each round, a process that does not hold token flips the value of its variable (see Figure 6.3(b)). On the contrary, each process that has a token must decide whether to pass it or not

and, according to this decision, set the value of its variable. A token disappears from the circle if two tokens collide in one process. The original version of the protocol resolves this decision by a single biased coin flip. Figure 6.3(c) illustrates a situation when processes 1 and 5 remain the same value of their variable (they keep the token) and process 7 decided to flip the value of its variable (it pass the token). However, we are interested in the synthesis of alternatives.

We give each station an additional single bit of memory and the choice between multiple different coin biases. We consider a sketch of this protocol where the holes represent the choices of a coin based on the memory value and memory updates. Subsequently, we obtain the same protocol for each process by resolving the individual choices. The synthesiser aims to identify the protocol version that minimises stabilisation time from an initial configuration, where each process has a variable set to true. We consider a set of parameters to describe a family of $3.1M$ candidate programs.

PAYNT synthesised the optimal protocol in around 18 minutes, while the naive one-by-one enumeration takes more than one day when we consider a sketch containing a system with five processes. The synthesised optimal strategy uses the fairest coins (bias ≈ 0.25) and maintains the values of memory set at true, when starting from the initial configuration. Whenever a process eventually decides to keep the token, the memory is reset to 0, and the process starts using highly unfair coins (bias ≈ 0.07). Thus, the process is more likely to keep its token for a long time until it eventually passed it further. The system can, on average, stabilise in four rounds when using this synthesised strategy.

Combined Synthesis. Up to now, we presented the *topology* synthesis. Its constraints cause the sketch to contain four parameters with 25 values representing different coin biases combined variable and memory value. When we refine this topology synthesis by considering 250 coin biases from the same interval $[0.0, 0.25]$, the synthesis problem is intractable as the family size increases about the factor $10k$. Therefore, we consider an alternative formulation of the synthesis problem where we replace these discrete options with continuous transition parameters over the same interval. The combined synthesis method requires around 53 minutes and guarantees the optimal solution over the continuous parameters space compared to the first version, where were considered only discrete steps. We concluded that the appropriate use of both syntheses (topology and parameters) brings easier writing of the program sketch, coverage of the continuous space of parameters, and the potentially better performance of the synthesis itself.

Chapter 7

Final Considerations

7.1 Future Research

Our future work will focus on increasing the efficiency of our novel hybrid synthesis technique for all kinds of synthesis problems. First, it would be desirable to introduce an improved refinement strategy because the designed method for combined synthesis currently uses only the naive round-robin splitting strategy. In addition, the existing splitting strategy implemented within the AR loop does not scale and has problems with large MDP models. A new splitting strategy should consider the structure of the MDP and, based on the knowledge obtained from it, derive the most appropriate refinement of the currently analysed family. In addition, it should scale appropriately when analysing large models so as not to reduce the efficiency of the entire synthesis process. Last but not least, within the AR loop, we will investigate the construction of counter-examples for MDPs, which could bring more efficient pruning of the family space and higher efficiency of synthesis processes.

Furthermore, we would like to research also more interesting challenges outside the scope of the existing framework for probabilistic synthesis. Since a model checking of a Markov chain against to given specification represents a core stage of the entire synthesis process, we want to adapt the state-space aggregation approaches developed in [1]. Deciding the monotonicity of parameters represents another open challenge, which could help the designer synthesise probabilistic systems and provide valuable feedback. Other open problems include synthesising infinite families of infinite-state Markov chains.

7.2 Conclusions

In this thesis, we considered the problem of automated synthesis of probabilistic systems. We designed and implemented support for *multi-property* specifications and *optimal* synthesis within all oracles – CEGIS, AR and especially Hybrid. We evaluated the performance of these advanced methods to synthesise probabilistic programs on the various benchmarks. Performed experiments on practically relevant benchmarks demonstrated that a *novel integrated* synthesis algorithm retains proven performance comparing to state-of-the-art synthesis approaches. Further, we designed a method to perform *combined* probabilistic synthesis consisting of both *topology* and *parameter* synthesis. We demonstrated the performance of this method for various real-world case studies and proved its correct functionality concerning the requirements. All these designed methods supporting various synthesis tasks were

integrated within the tool PAYNT, which was also introduced. Additionally, PAYNT is the first tool that supports the combined synthesis, including unknown topology as well as transition probabilities.

Our tool paper presenting PAYNT has been recently accepted at CAV'21, an A* conference. Moreover, we presented PAYNT also at the students' conference Excel@FIT'21, where we got the award from the expert panel to develop of the tool that significantly expands the possibilities of designing probabilistic systems. At this conference, we also won an award from the professional public.

Bibliography

[1] Andriuschchenko, I. R. *Computed-Aided Synthesis of Probabilistic Models*. Brno, CZ, 2020. Master's thesis. BUT, FIT. Available at: https://www.vutbr.cz/studenti/zav-prace/detail/129318.

[2] Andriushchenko, R., Češka, M., Junges, S. and Katoen, J.-P. Inductive Synthesis for Probabilistic Programs Reaches New Horizons. In: *TACAS*. Springer, 2021. Lecture Notes in Computer Science. (to appear, see https://arxiv.org/abs/2101.12683).

[3] Aziz, A., Singhal, V., Balarin, F., Brayton, R. K. and Sangiovanni Vincentelli, A. L. It usually works: The temporal logic of stochastic systems. In: Wolper, P., ed. *Computer Aided Verification*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, p. 155–165. ISBN 978-3-540-49413-3.

[4] Bartocci, E., Grosu, R., Katsaros, P., Ramakrishnan, C. R. and Smolka, S. A. Model Repair for Probabilistic Systems. In: Abdulla, P. A. and Leino, K. R. M., ed. *Tools and Algorithms for the Construction and Analysis of Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, p. 326–340. ISBN 978-3-642-19835-9.

[5] Bornholt, J., Torlak, E., Grossman, D. and Ceze, L. Optimizing Synthesis with Metasketches. In: *POPL'16*. Association for Computing Machinery, 2016, p. 775–788. ISBN 9781450335492.

[6] Calinescu, R., Češka, M., Gerasimou, S., Kwiatkowska, M. and Paoletti, N. RODES: A Robust-Design Synthesis Tool for Probabilistic Systems. In: *QEST*. Springer, 2017, p. 304–308.

[7] Češka, M., Dannenberg, F., Kwiatkowska, M. and Paoletti, N. Precise Parameter Synthesis for Stochastic Biochemical Systems. In: Mendes, P., Dada, J. O. and Smallbone, K., ed. *Computational Methods in Systems Biology*. Cham: Springer International Publishing, 2014, p. 86–98. ISBN 978-3-319-12982-2.

[8] Češka, M., Hensel, C., Junges, S. and Katoen, J.-P. Counterexample-Driven Synthesis for Probabilistic Program Sketches. In: *FM*. Springer, 2019, vol. 11800, p. 101–120. LNCS.

[9] Češka, M., Jansen, N., Junges, S. and Katoen, J.-P. Shepherding Hordes of Markov Chains. In: *TACAS*. Springer, 2019, vol. 11428, p. 172–190. LNCS.

[10] 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.

[11] 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. DOI: 10.1007/s00165-017-0432-4.

[12] Classen, A., Cordy, M., Heymans, P., Legay, A. and Schobbens, P.-Y. Model checking software product lines with SNIP. *Int. J. on Softw. Tools for Technol. Transf.* 2012, vol. 14, p. 589–612.

[13] 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.

[14] Daws, C. Symbolic and Parametric Model Checking of Discrete-Time Markov Chains. In: *ICTAC*. Springer, 2004, vol. 3407, p. 280–294. LNCS.

[15] Dehnert, C., Junges, S., Jansen, N., Corzilius, F., Volk, M. et al. PROPhESY: A PRObabilistic ParamEter SYNnthesis Tool. In: *CAV'15*. Springer, 2015, vol. 9206, p. 214–231. LNCS.

[16] Dehnert, C., Junges, S., Katoen, J.-P. and Volk, M. A Storm is Coming: A Modern Probabilistic Model Checker. In: *CAV*. Springer, 2017, vol. 10427, p. 592–600. LNCS.

[17] 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.

[18] Han, J. and Jonker, P. A system architecture solution for unreliable nanoelectronic devices. *Nanotechnology, IEEE Transactions on*. january 2003, vol. 1, p. 201 – 208. DOI: 10.1109/TNANO.2002.807393.

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

[20] Inala, J. P., Bastani, O., Tavares, Z. and Solar Lezama, A. Synthesizing Programmatic Policies that Inductively Generalize. In: *International Conference on Learning Representations*. 2020.

[21] Jha, S., Gulwani, S., Seshia, S. A. and Tiwari, A. Oracle-Guided Component-Based Program Synthesis. In: *ICSE*. ACM, 2010, p. 215–224.

[22] Jha, S. and Seshia, S. A. A theory of formal synthesis via inductive learning. *Acta Informatica*. 2017, vol. 54, no. 7, p. 693–726.

[23] Kaelbling, L. P., Littman, M. L. and Cassandra, A. R. Planning and acting in partially observable stochastic domains. *Artificial intelligence*. Elsevier. 1998, vol. 101, 1-2, p. 99–134.

[24] 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.

[25] Kwiatkowska, M., Norman, G. and Parker, D. PRISM 4.0: Verification of probabilistic real-time systems. In: Springer. *CAV*. 2011, vol. 6806, p. 585–591. LNCS.

[26] Long, F. and Rinard, M. Automatic Patch Generation by Learning Correct Code. New York, NY, USA: Association for Computing Machinery. january 2016, vol. 51, no. 1, p. 298–312. DOI: 10.1145/2914770.2837617. ISSN 0362-1340. Available at: https://doi.org/10.1145/2914770.2837617.

[27] Manna, Z. and Waldinger, R. A Deductive Approach to Program Synthesis. New York, NY, USA: Association for Computing Machinery. january 1980, vol. 2, no. 1, p. 90–121. DOI: 10.1145/357084.357090. ISSN 0164-0925. Available at: https://doi.org/10.1145/357084.357090.

[28] Massalin, H. Superoptimizer: A Look at the Smallest Program. In:. Washington, DC, USA: IEEE Computer Society Press, 1987, p. 122–126. ASPLOS II. DOI: 10.1145/36206.36194. ISBN 0818608056. Available at: https://doi.org/10.1145/36206.36194.

[29] 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.

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

[31] Paleologo, G. A., Benini, L., Bogliolo, A. and De Micheli, G. Policy Optimization for Dynamic Power Management. In: *Proceedings of the 35th Annual Design Automation Conference*. New York, NY, USA: Association for Computing Machinery, 1998, p. 182–187. DAC '98. DOI: 10.1145/277044.277094. ISBN 0897919645. Available at: https://doi.org/10.1145/277044.277094.

[32] Pathak, S., Ábrahám, E., Jansen, N., Tacchella, A. and Katoen, J.-P. A Greedy Approach for the Efficient Repair of Stochastic Models. In: *NFM'15*. Springer, 2015, vol. 9058, p. 295–309. LNCS.

[33] Pu, C., Massalin, H. and Ioannidis, J. The Synthesis Kernel. *Comput. Syst.* february 1970, vol. 1.

[34] Quatmann, T., Dehnert, C., Jansen, N., Junges, S. and Katoen, J.-P. Parameter Synthesis for Markov Models: Faster Than Ever. In: *ATVA'16*. 2016, vol. 9938, p. 50–67. LNCS.

[35] Rodrigues, G., Alves, V., Nunes, V., Lanna, A., Cordy, M. et al. Modeling and Verification for Probabilistic Properties in Software Product Lines. In: *Proceedings of IEEE International Symposium on High Assurance Systems Engineering*. 16th ed. IEEE Computer Society Press, vol. 2015, p. 173–180. DOI: 10.1109/HASE.2015.34.

[36] Solar Lezama, A. *Program Synthesis By Sketching*. 2008. Dissertation. EECS Department, University of California, Berkeley. Available at: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-177.html.

[37] Solar Lezama, A., Tancau, L., Bodik, R., Seshia, S. and Saraswat, V. Combinatorial Sketching for Finite Programs. In:. New York, NY, USA: Association for Computing Machinery, 2006, p. 404–415. ASPLOS XII. DOI: 10.1145/1168857.1168907. ISBN 1595934510. Available at: https://doi.org/10.1145/1168857.1168907.

[38] Su, G. and Rosenblum, D. S. Nested Reachability Approximation for Discrete-Time Markov Chains with Univariate Parameters. In: Cassez, F. and Raskin, J.-F., ed. *Automated Technology for Verification and Analysis*. Cham: Springer International Publishing, 2014, p. 364–379. ISBN 978-3-319-11936-6.

[39] Vandin, A., Beek, M. H. ter, Legay, A. and Lluch Lafuente, A. QFLan: A Tool for the Quantitative Analysis of Highly Reconfigurable Systems. In: Havelund, K., Peleska, J., Roscoe, B. and Vink, E. de, ed. *Formal Methods*. Cham: Springer International Publishing, 2018, p. 329–337. ISBN 978-3-319-95582-7.

[40] Verma, A., Murali, V., Singh, R., Kohli, P. and Chaudhuri, S. Programmatically interpretable reinforcement learning. In: PMLR. *International Conference on Machine Learning*. 2018, p. 5045–5054.

Appendix A

Storage Medium

`/synthesis/*` — source code of PAYNT from date May 25, 2021

`/README.txt` — useful information about the storage medium content

`/text/*` — source code of this thesis

`/xstupi00.pdf` — final version of this thesis