Saarland University

Faculty of Mathematics and Computer Science

Department of Computer Science

Bachelor's Thesis

Bounded Model Checking for PHL

submitted by

Simon Engel

on 6 September 2021

Reviewed by:

Prof. Bernd Finkbeiner, Ph.D.

Dr. Rayna Dimitrova

Supervisor:

Prof. Bernd Finkbeiner, Ph.D.

Advisor:

Norine Coenen, M.Sc.

Abstract

In the context of formal verification, model checking is the automated process of proving or disproving that a formal model of a system satisfies a given property. This property may be specified in a specialized specification language such as a temporal logic.

PHL (Probabilistic Hyper Logic) is a relatively new temporal logic for specifying probabilistic hyperproperties of Markov decision processes (MDPs). PHL was first proposed a year ago by Dimitrova, Finkbeiner, and Torfah, who also developed two model checking algorithms for a fragment of PHL. One of these algorithms involves synthesizing a so-called scheduler for an MDP, applying this scheduler to the MDP, and then checking whether the resulting system satisfies a given probabilistic constraint. This process is iterated until either an adequate scheduler is found, or all schedulers up to a certain size have been checked.

This thesis presents an approach to eliminating this guess-and-check loop. The key to this approach is an improved scheduler synthesis procedure that is able to immediately find an adequate scheduler, if such a scheduler exists, and that can even synthesize an optimal scheduler that maximizes or minimizes a given probability expression. A first feasibility study shows that scheduler synthesis for a simple property and small MDPs can be performed on ordinary hardware, but suggests that further optimization will be necessary to make this approach viable for real-world use cases.

Acknowledgments

Firstly, I would like to thank my supervisor Prof. Bernd Finkbeiner. Thank you for giving me the opportunity of writing this thesis, even after I failed to pass your core course on verification. Working on this exciting topic has been an enriching experience for me, and I am particularly grateful for the insights I have gained into formal methods, as well as for the opportunity to play around with SMT for the first time in my life. I know that this thesis has taken me very long to finish, and I cannot thank you enough for your patience and for supporting me every single time I messed something up. I hope you will not be too disappointed with the result.

Secondly, and above all, I want to thank my advisor Norine Coenen for just about everything there is to be thankful for. This journey has not always been easy, but even when I was close to giving up, you did not, and you pulled me through. I have learned a lot during this past year, though probably not half as much as you wanted me to, and I think it is safe to say that I owe most of it to you. Thank you for being awesome!

Lastly, many thanks are due to Dr. Rayna Dimitrova for immediately agreeing to review this thesis, and to my friends and family for their support, moral and otherwise. In particular, I would like to thank Maribelle Paul for proofreading parts of this thesis. I really appreciate your feedback!

Contents

1 Introduction

Formal verification has the potential to prevent severe damage that may result from the failure of security-critical computing systems. Therefore, formal verification is especially desirable in the context of security-critical systems. However, formally verifying security policies can pose great theoretical challenges. This is due to the fact that security-critical systems often combine internal randomization with interaction with a potentially adversarial environment, and many security policies relate multiple possible executions of the system to one another.

A good example is a database management system that is supposed to implement differential privacy [Dwo11]. Obviously, this system must handle queries from an unpredictable environment. In addition, differential privacy demands that the system must randomize its outputs in such a way that for similar databases, the probabilities of all possible replies to a given query are almost the same. This means that the system must use internal randomization, and that its correctness can only be expressed as a relation between multiple executions on different databases.

In order to formally verify the correctness of such a system, we first need an appropriate type of formal model. In particular, the model must allow for nondeterminism, which is needed to capture the unpredictability of the environment, as well as probabilistic choice to model the system's internal randomization. Secondly, we need a specification language for this type of model that can express so-called probabilistic hyperproperties: properties that combine probabilistic constraints with the defining characteristic of hyperproperties, namely correctness as a relation between multiple executions of a system.

One framework that fulfills these requirements is Markov decision processes (MDPs) plus the relatively new temporal logic PHL (Probabilistic Hyper Logic). MDPs are a type of system model that combines nondeterminism with probabilistic choice, and that can be viewed as a generalization of both Kripke structures and discrete-time Markov chains (DTMCs). Every transition from one state of an MDP to another comprises two steps: first the nondeterministic choice of a so-called action; and then the probabilistic choice of a successor state, where the transition probabilities depend on the action that was chosen in the first step. An MDP can be reduced to a DTMC by fixing a so-called scheduler, which resolves the nondeterminism in the MDP by assigning a probability to each action choice.

PHL was proposed a year ago by Dimitrova, Finkbeiner, and Torfah [DFT20] as a temporal logic for expressing probabilistic hyperproperties of MDPs. It allows for quantification over multiple possible schedulers for an MDP and can express Boolean combinations of hyperproperties and probabilistic constraints on the resulting DTMCs. While this makes PHL a very expressive logic, it also makes the model checking problem for PHL generally undecidable. Dimitrova, Finkbeiner, and Torfah were however able to develop two model checking algorithms for a fragment of PHL which can express many properties of interest, including differential privacy. One of these algorithms is a sound but incomplete approximate model checking algorithm for proving PHL formulas from this fragment, while the other one is a bounded model checking algorithm for finding counterexamples to disprove them.

Intuitively, the goal of the latter algorithm is, given an MDP and a PHL formula that must begin with a universal scheduler quantifier, to find a scheduler for the MDP such that it falsifies the formula. This is equivalent to finding a scheduler that satisfies the negation of the given formula, which is itself a PHL formula that begins with an existential scheduler quantifier. We will be using this latter formulation of the problem throughout this chapter.

Since even for the simplest MDPs, there exist infinitely many schedulers that, when applied to the MDP, result in infinite DTMCs, the algorithm focuses on deterministic finite-memory schedulers of bounded size. Deterministic schedulers can only assign probabilities of either 1 or 0 to action choices, which makes these choices essentially deterministic. A finite-memory scheduler consists of a finite-state machine that serves as its memory and is updated on every step of the MDP, in combination with an action choice function that chooses the next action based on the scheduler's memory state and the current state of the MPD. The size of a finite-memory scheduler is its number of memory states. Finite-memory schedulers have the advantage that they induce only finite DTMCs. Furthermore, for a given size bound, there exist only finitely many deterministic finite-memory schedulers up to that size. Therefore, by considering only deterministic finite-memory schedulers of bounded size, the problem of determining whether there exists a scheduler with certain properties becomes decidable.

The bounded model checking algorithm works as follows. It first synthesizes a scheduler for the MDP under consideration such that the hyperproperty from the PHL formula is guaranteed to be satisfied. Then it uses a probabilistic model checker to determine whether the resulting DTMC also satisfies the probabilistic constraint from the formula. If this is not the case, the process is iterated until either an adequate scheduler is found, or all schedulers up to a certain size have been checked.

Clearly, the running time of this algorithm depends heavily on the number of iterations until an adequate scheduler is found. Worse still, after every unsuccessful try, an additional constraint is added to the hyperproperty in order to make sure that the same scheduler will not be synthesized again. We can therefore expect the running time to increase faster than linear in the number of iterations. Dimitrova, Finkbeiner, and Torfah also conducted some experiments with a proof-of-concept implementation, and their results are in line with this hypothesis. This observation naturally raises the question whether it might be possible to modify the scheduler synthesis procedure such that it immediately produces an adequate scheduler, and thus to eliminate the iteration altogether.

In this thesis, we develop such an improved scheduler synthesis procedure. We present a general construction for encoding the existence of an adequate scheduler as a satisfiability modulo theories (SMT) constraint system. Furthermore, by adding an optimization objective to this constraint system, it is possible to find optimal schedulers that maximize or minimize a given probability expression. We also briefly discuss experiments with a proof-of-concept implementation of this scheduler synthesis procedure for a specific case study, which consists of a very simple probabilistic hyperproperty for a parameterized and therefore scalable MDP. This first feasibility study shows that it is possible to solve the constraint system, for small instances, with an off-the-shelf SMT solver on ordinary hardware. However, the results suggest that further optimization will be necessary in order to make this approach scale up to real-world use cases such as the differential privacy example from the beginning of this chapter.

The rest of this thesis is structured as follows. Chapter 2 provides the necessary background information and definitions that we will later need to present the SMT constraint system and to argue for its correctness: Section 2.1 presents the relevant temporal logics and related algorithms, while Section 2.2 briefly introduces the SMT framework from a user's perspective. In Chapter 3, we formally introduce the bounded model checking problem that we want to solve and describe a general procedure for encoding the corresponding scheduler synthesis problem as an SMT constraint system. Chapter 4 presents first experimental results with a proof-of-concept implementation of our new scheduler synthesis procedure for a simple case study. Finally, we discuss related work in Chapter 5, and we summarize the results from this thesis and outline possible future work in Chapter 6.

2 Background

This chapter provides background information on the relevant temporal logics as well as the existing model checking and synthesis algorithms that we will later build upon. It also gives a brief introduction to Satisfiability Modulo Theories (SMT) and Optimization Modulo Theories (OMT) which will form the basis for our new bounded model checking algorithm.

2.1 Temporal Logics

The goal of this thesis is to develop an improved model checking algorithm for a fragment of PHL. Every formula from this fragment contains a HyperLTL subformula and at least one LTL subformula. Our new model checking algorithm will combine ideas from existing algorithms for HyperLTL synthesis and for probabilistic LTL model checking. This section gives an overview of the relevant temporal logics, including LTL, HyperLTL, and PHL, and briefly describes the aforementioned model checking and synthesis algorithms.

2.1.1 Linear Temporal Logic

We want to specify the correctness of a system in terms of its behavior over time. The behavior of a system can be described through a set of so-called execution traces that record for every time step a set of so-called atomic propositions that hold at that point in time. Kripke structures are a type of transition system that produces such execution traces.

Definition 1 (Kripke Structure)**.** A *Kripke structure* is a tuple $K = (S, \rightarrow, I, \mathsf{AP}, L)$ where

- S is a finite, nonempty set of *states*,
- $\rightarrow \subseteq S \times S$ is the *transition relation*,
- $I \subseteq S$ is the set of *initial states*,
- AP is a finite set of *atomic propositions*, and
- $L : S \rightarrow 2^{\mathsf{AP}}$ is the *labeling function*

such that

i) for all $s \in S$ there exists an $s' \in S$ such that $s \rightarrow s'$.

Remark. Requirement i) ensures that every state has at least one outgoing transition.

Definition 2 (Execution Trace)**.** Let $K = (S, \to, I, \mathsf{AP}, L)$ be a Kripke structure. A *path* of K is a sequence $\pi = s_0 s_1 s_2 \cdots \in S^\omega$ of states such that $s_0 \in I$, and for all $i \geq 0$ we have $s_i \to s_{i+1}$. The *trace* of path π is the sequence

$$\mathit{trace}(\pi) = L(s_0)L(s_1)L(s_2)\cdots \in (2^{\mathsf{AP}})^\omega.$$

We denote the set of all traces of K by $\mathit{Traces}(K)$.

LTL is a specification language that can express properties of execution traces. It can be thought of as propositional logic over atomic propositions plus temporal operators. The unary temporal operator $\bigcirc$ ("next") expresses that the following subformula must hold in the next time step, while the binary temporal operator $\mathcal{U}$ ("until") expresses that its right-hand side must hold at some point, and in every time step until then, its left-hand side must hold.

Definition 3 (LTL Syntax)**.** Let AP be a finite set of atomic propositions. *LTL formulas* over AP are defined by the grammar

$$\varphi \quad ::= \quad a \;\mid\; \varphi \wedge \varphi \;\mid\; \neg\varphi \;\mid\; \bigcirc\varphi \;\mid\; \varphi \,\mathcal{U}\, \varphi$$

where $a \in \mathsf{AP}$.

There exist a number of derived temporal operators, including $\Diamond$ ("eventually") and $\Box$ ("globally"), which are defined by

$$\Diamond\varphi = \top \,\mathcal{U}\, \varphi \qquad \text{and} \qquad \Box\varphi = \neg\Diamond\neg\varphi.$$

Definition 4 (LTL Semantics)**.** Let AP be a finite set of atomic propositions. The *satisfaction relation* $\models$ between infinite words over 2^{AP} and LTL formulas over AP is defined inductively as follows:

$$\begin{array}{lll}
\lambda \models a & \text{iff} & a \in \lambda_0 \\
\lambda \models \varphi_1 \wedge \varphi_2 & \text{iff} & \lambda \models \varphi_1 \quad \text{and} \quad \lambda \models \varphi_2 \\
\lambda \models \neg\varphi & \text{iff} & \lambda \not\models \varphi \\
\lambda \models \bigcirc\varphi & \text{iff} & \lambda[1,\infty] \models \varphi \\
\lambda \models \varphi_1 \,\mathcal{U}\, \varphi_2 & \text{iff} & \text{there exists an } i \geq 0 \text{ such that } \lambda[i,\infty] \models \varphi_2 \\
 & & \text{and for all } 0 \leq j < i \text{ we have } \lambda[j,\infty] \models \varphi_1
\end{array}$$

where $\lambda = \lambda_0\lambda_1\lambda_2 \cdots \in (2^{\mathsf{AP}})^\omega$.

2.1.2 LTL Model Checking

The so-called automata-theoretic approach makes use of the fact that for every LTL formula φ, there exists an ω-automaton that accepts exactly those traces that satisfy φ. One of the simplest LTL model checking algorithms for Kripke structures involves constructing nondeterministic Büchi automata (NBAs) from LTL forumlas.

Definition 5. Let $s = s_0 s_1 s_2 \cdots$ be an infinite sequence, and let $S = \bigcup_{i=0}^{\infty}\{s_i\}$ be the set of all terms in s. We denote the set of terms that occur infinitely often in s by

$$\mathrm{Inf}(s) = \{t \in S : \text{there exist infinitely many } i \in \mathbb{N} \text{ such that } s_i = t\}.$$

Definition 6 (Nondeterministic Büchi Automaton)**.** A *nondeterministic Büchi automaton* (NBA) is a tuple $\mathcal{A} = (A, \Lambda, \delta, a_0, F)$ where

- A is a finite, nonempty set of *states*,
- Λ is a finite, nonempty *alphabet*,
- $\delta : A \times \Lambda \to 2^A$ is the *transition function*,
- $a_0 \in A$ is the *initial state*, and
- $F \subseteq A$ is the set of *accepting states*.

Let $\lambda = \lambda_0\lambda_1\lambda_2 \cdots \in \Lambda^\omega$ be an infinite word over Λ. A *run* of $\mathcal{A}$ on λ is an infinite sequence $a = a_0a_1a_2 \cdots \in A^\omega$ of states such that a_0 is the initial state, and for all $i \geq 0$ we have $a_{i+1} \in \delta(a_i, \lambda_i)$. Run a is *accepting* if $\mathrm{Inf}(a) \cap F \neq \emptyset$, i.e. it contains infinitely many accepting states. $\mathcal{A}$ *accepts* λ if there exists an accepting run of $\mathcal{A}$ on λ. The *language* of $\mathcal{A}$ is the set $\mathcal{L}(\mathcal{A}) = \{\lambda \in \Lambda^\omega : \mathcal{A} \text{ accepts } \lambda\}$.

Given a Kripke structure K and an LTL formula φ, we can determine whether K satisfies φ by constructing an NBA $\mathcal{A} = (A, 2^{\mathsf{AP}}, \delta, a_0, F)$ for $\neg\varphi$ and checking whether in the product Kripke structure $K \otimes \mathcal{A}$, a loop is reachable that contains a state with a component from F.

2.1.3 Probabilistic LTL Model Checking

DTMCs are a type of system model that basically works like Kripke structures, except that the next state is always determined by probabilistic choice.

Definition 7 (Discrete-Time Markov Chain)**.** A *discrete-time Markov chain* (DTMC) is a tuple $M = (S, \mathrm{P}, \iota, \mathsf{AP}, L)$ where

- S is a finite, nonempty set of *states*,
- $\mathrm{P} : S \times S \to [0, 1]$ is the *transition probability function*,
- $\iota : S \to [0, 1]$ is the *initial distribution*,
- AP is a finite set of *atomic propositions*, and
- $L : S \to 2^{\mathsf{AP}}$ is the *labeling function*

such that

i) for all $s \in S$ we have $\sum_{s' \in S} \mathrm{P}(s, s') = 1$, and

ii) $\sum_{s \in S} \iota(s) = 1$.

Remark. Requirement i) ensures that the transition probabilities from every state s define a probability measure on S. Requirement ii) ensures that the initial distribution defines a probability measure on S.

Given a DTMC M and an LTL formula φ, we can compute with what probability M will produce a trace that satisfies φ. However, this requires a deterministic type of ω-automaton, such as deterministic Rabin automata (DRAs).

Definition 8 (Deterministic Rabin Automaton)**.** A *deterministic Rabin automaton* (DRA) is a tuple $\mathcal{R} = (R, \Lambda, \delta, r_0, Acc)$ where

- R is a finite, nonempty set of *states*,
- Λ is a finite, nonempty *alphabet*,
- $\delta : R \times \Lambda \to R$ is the *transition function*,
- $r_0 \in R$ is the *initial state*, and
- $Acc = (B_j, G_j)_{j=1}^{m} \subseteq 2^R \times 2^R$ is the *acceptance condition*.

Let $\lambda = \lambda_0\lambda_1\lambda_2 \cdots \in \Lambda^\omega$ be an infinite word over Λ. The *run* of $\mathcal{R}$ on λ is the unique infinite sequence $r = r_0r_1r_2 \cdots \in R^\omega$ of states such that r_0 is the initial state, and for all $i \geq 0$ we have $r_{i+1} = \delta(r_i, \lambda_i)$. Run r is *accepting* if there exists a $j \in \{1, \ldots, m\}$ such that $\mathrm{Inf}(r) \cap B_j = \emptyset$ and $\mathrm{Inf}(r) \cap G_j \neq \emptyset$, i.e. the run contains only finitely many states from B_j and infinitely many states from G_j. $\mathcal{R}$ *accepts* λ if the run of $\mathcal{R}$ on λ is accepting. The *language* of $\mathcal{R}$ is the set $\mathcal{L}(\mathcal{R}) = \{\lambda \in \Lambda^\omega : \mathcal{R} \text{ accepts } \lambda\}$.

NBAs can be translated into equivalent DRAs, and thus there exists a DRA for every LTL formula. Probabilistic LTL model checking for DTMCs can be performed as follows. First, the LTL formula is translated into a DRA, and the product DTMC of the input DTMC and the DRA is constructed.

Definition 9 (Product of DTMC and DRA)**.** Let $M = (S, \mathrm{P}, \iota, \mathsf{AP}, L)$ be a DTMC and $\mathcal{R} = (R, 2^{\mathsf{AP}}, \delta, r_0, Acc)$ be a DRA. The *product* of M and $\mathcal{R}$ is the DTMC $M \otimes \mathcal{R} = (S \times R, \widehat{\mathrm{P}}, \widehat{\iota}, \mathsf{AP}, \widehat{L})$ with

- $\forall s, s' \in S, r, r' \in R : \widehat{\mathrm{P}}\big((s, r), (s', r')\big) = \begin{cases} \mathrm{P}(s, s') & \text{if } r' = \delta\big(r, L(s)\big) \\ 0 & \text{otherwise} \end{cases}$,
- $\forall s \in S, r \in R : \widehat{\iota}\big((s, r)\big) = \begin{cases} \iota(s) & \text{if } r = r_0 \\ 0 & \text{otherwise} \end{cases}$, and
- $\forall s \in S, r \in R : \widehat{L}\big((s, r)\big) = L(s)$.

The model checking algorithm then determines the union of the accepting bottom strongly connected components (BSCCs) of the product DTMC. Finally, a system of linear equations is solved to obtain the probability of reaching an accepting BSCC from the initial states. A detailed description of this algorithm can be found in Section 10.3 of [BK08].

2.1.4 HyperLTL

While trace properties, as expressible by LTL, only require every single execution of a system to satisfy some property, hyperproperties can also relate multiple executions to one another. The logic HyperLTL extends LTL with quantification over traces and can thus express hyperproperties.

Definition 10 (HyperLTL Syntax)**.** Let AP be a finite set of atomic propositions and $\mathcal{V}$ be a countably infinite set of trace variables. *HyperLTL formulas* over AP are defined by the grammar

$$\begin{aligned} \psi &::= \forall \pi .\, \psi \mid \exists \pi .\, \psi \mid \varphi \\ \varphi &::= a_\pi \mid \varphi \wedge \varphi \mid \neg \varphi \mid \bigcirc \varphi \mid \varphi \,\mathcal{U}\, \varphi \end{aligned}$$

where $\pi \in \mathcal{V}$ and $a \in \mathsf{AP}$.

Definition 11 (HyperLTL Semantics)**.** Let AP be a finite set of atomic propositions. The *satisfaction relation* $\models$ between sets of infinite words over 2^{AP} and HyperLTL formulas over AP is defined inductively as follows:

$$\begin{array}{lll} \Pi \models_T \forall \pi .\, \psi & \text{iff} & \text{for all } \lambda \in T \text{ we have } \Pi[\pi \mapsto \lambda] \models_T \psi \\ \Pi \models_T \exists \pi .\, \psi & \text{iff} & \text{there exists a } \lambda \in T \text{ such that } \Pi[\pi \mapsto \lambda] \models_T \psi \\ \Pi \models_T a_\pi & \text{iff} & a \in \Pi(\pi)[0] \\ \Pi \models_T \varphi_1 \wedge \varphi_2 & \text{iff} & \Pi \models_T \varphi_1 \quad \text{and} \quad \Pi \models_T \varphi_2 \\ \Pi \models_T \neg \varphi & \text{iff} & \Pi \not\models_T \varphi \\ \Pi \models_T \bigcirc \varphi & \text{iff} & \Pi[1,\infty] \models_T \varphi \\ \Pi \models_T \varphi_1 \,\mathcal{U}\, \varphi_2 & \text{iff} & \text{there exists an } i \geq 0 \text{ such that } \Pi[i,\infty] \models_T \varphi_2 \\ & & \text{and for all } 0 \leq j < i \text{ we have } \Pi[j,\infty] \models_T \varphi_1 \end{array}$$

where $T \subseteq (2^{\mathsf{AP}})^\omega$.

2.1.5 HyperLTL Synthesis

HyperLTL synthesis is the automated process of constructing a system that satisfies a given HyperLTL formula. The HyperLTL synthesis procedure from [Fin+20] involves constructing a universal co-Büchi automaton for the LTL suffix of the given HyperLTL formula.

Definition 12 (Universal Co-Büchi Automaton)**.** A *universal co-Büchi automaton* is a tuple $\mathcal{B} = (B, \Lambda, \delta, b_0, F)$ where

- B is a finite, nonempty set of *states*,
- Λ is a finite, nonempty *alphabet*,
- $\delta : B \times \Lambda \to 2^B$ is the *transition function*,
- $b_0 \in B$ is the *initial state*, and
- $F \subseteq B$ is the set of *rejecting states*.

Let $\lambda = \lambda_0\lambda_1\lambda_2 \cdots \in \Lambda^\omega$ be an infinite word over Λ. A *run* of $\mathcal{B}$ on λ is an infinite sequence $b = b_0b_1b_2 \cdots \in B^\omega$ of states such that b_0 is the initial state, and for all $i \geq 0$ we have $b_{i+1} \in \delta(b_i, \lambda_i)$. Run b is *accepting* if $\mathrm{Inf}(b) \cap F = \emptyset$, i.e. it contains only finitely many rejecting states. $\mathcal{B}$ *accepts* λ if all runs of $\mathcal{B}$ on λ are accepting. The *language* of $\mathcal{B}$ is the set $\mathcal{L}(\mathcal{B}) = \{\lambda \in \Lambda^\omega : \mathcal{B} \text{ accepts } \lambda\}$.

Given an NBA $\mathcal{A} = (A, \Lambda, \delta, a_0, F)$, the universal co-Büchi automaton $\mathcal{B} = (A, \Lambda, \delta, a_0, F)$ accepts the language $\Lambda^\omega \setminus \mathcal{L}(\mathcal{A})$. Thus, there exists a universal co-Büchi automaton for every LTL formula.

The HyperLTL synthesis procedure from [Fin+20] is SMT-based, and we will use exactly the same idea as part of our SMT-based PHL model checking procedure.

2.1.6 Probabilistic Hyper Logic

PHL is a temporal logic for specifying probabilistic hyperproperties of MDPs. MDPs combine nondeterminism with probabilistic choice and can be viewed as a generalization of both Kripke structures and DTMCs.

Definition 13 (Markov Decision Process)**.** A *Markov decision process* (MDP) is a tuple $M = (S, \mathit{Act}, \mathrm{P}, \iota, \mathsf{AP}, L)$ where

- S is a finite, nonempty set of *states*,
- Act is a finite, nonempty set of *actions*,
- $\mathrm{P} : S \times \mathit{Act} \times S \to [0, 1]$ is the *transition probability function*,
- $\iota : S \to [0, 1]$ is the *initial distribution*,
- AP is a finite set of *atomic propositions*, and
- $L : S \to 2^{\mathsf{AP}}$ is the *labeling function*

such that

i) for all $s \in S$ and $a \in \mathit{Act}$ we have $\sum_{s' \in S} \mathrm{P}(s, a, s') \in \{0, 1\}$,

ii) for all $s \in S$ there exists an $a \in \mathit{Act}$ such that $\sum_{s' \in S} \mathrm{P}(s, a, s') = 1$, and

iii) $\sum_{s \in S} \iota(s) = 1$.

Remark. Requirement i) ensures that the transition probabilities from every state s via every action a are either all zero, or they define a probability measure on S. Requirement ii) ensures that every state has at least one outgoing transition. Requirement iii) ensures that the initial distribution defines a probability measure on S.

By fixing a scheduler which resolves the nondeterminism, an MDP can be reduced to a DTMC.

Definition 14 (Scheduler)**.** Let $M = (S, \mathit{Act}, \mathrm{P}, \iota, \mathsf{AP}, L)$ be an MDP. A *scheduler* for M is a function

$$\mathfrak{S} : (S \cdot \mathit{Act})^* S \to (\mathit{Act} \to [0, 1])$$

such that for all $t = s_0 a_0 \cdots s_{n-1} a_{n-1} s_n \in (S \cdot \mathit{Act})^* S$ it holds that

i) $\sum_{a \in \mathit{Act}} \mathfrak{S}(t)(a) = 1$, and

ii) for all $a \in \mathit{Act}$ with $\mathfrak{S}(t)(a) > 0$ we have $\sum_{s \in S} \mathrm{P}(s_n, a, s) > 0$.

Remark. Requirement i) ensures that $\mathfrak{S}(t)$ defines a probability distribution on *Act*. Requirement ii) ensures that $\mathfrak{S}$ only chooses actions that are enabled in the current state s_n.

Note that the above definition is far more general than is required for this thesis, since we are only going to deal with deterministic finite-memory schedulers. A definition of how deterministic finite-memory schedulers are applied to MDPs is given in Section 3.2.

Definition 15 (PHL Syntax)**.** Let AP be a finite set of atomic propositions, and let $\mathcal{V}_{sched}$ resp. $\mathcal{V}_{path}$ be countably infinite sets of scheduler variables resp. path variables. *PHL formulas* over AP are defined by the grammar

$$\begin{array}{rcl} \Phi & ::= & \forall\sigma.\,\Phi \mid \Phi\wedge\Phi \mid \neg\Phi \mid \chi \mid P\bowtie c \\ \chi & ::= & a_\pi \mid \chi\wedge\chi \mid \neg\chi \mid \bigcirc\chi \mid \chi\,\mathcal{U}\,\chi \mid \forall\pi:\sigma.\,\chi \\ P & ::= & \mathbb{P}(\varphi) \mid P+P \mid c\cdot P \\ \varphi & ::= & a_\sigma \mid \varphi\wedge\varphi \mid \neg\varphi \mid \bigcirc\varphi \mid \varphi\,\mathcal{U}\,\varphi \end{array}$$

where $\sigma\in\mathcal{V}_{sched}$ and $\bowtie\,\in\{<,\le,\ge,>\}$ and $c\in\mathbb{Q}$ and $\pi\in\mathcal{V}_{path}$ and $a\in\mathsf{AP}$.

2.2 Satisfiability Modulo Theories

The basic idea behind our new model checking algorithm is to encode a scheduler synthesis problem as an SMT or OMT constraint system. SMT is a generalization of the Boolean satisfiability problem (SAT), and the maximum satisfiability problem (MaxSAT) and OMT are optimization problems that extend SAT and SMT, respectively. This section gives a brief overview of SAT, SMT, MaxSAT, and OMT.

2.2.1 SAT and SMT

Intuitively, the SAT is the following decision problem: given a propositional formula $\mathcal{F}$, decide whether or not there exists a truth assignment for the variables in $\mathcal{F}$ such that $\mathcal{F}$ evaluates to true. SMT is a generalization of SAT to first-order formulas, where the goal is to decide whether a given formula is satisfiable with respect to some background theory. Background theories may prescribe the interpretation of certain predicate and function symbols, such as numeric constants or arithmetic operators. There are a number of background theories commonly implemented by SMT tools, such as the theory of Equality with Uninterpreted Functions (EUF), the theories of real arithmetic, of linear integer arithmetic, of mixed integer and real arithmetic, or the theory of arrays. Using an appropriate background theory, or a combination of multiple background theories, many constraints can be specified in a natural and concise way. An introduction to SAT can be found in [MM18], and both [Bar+09] and [BT18] provide comprehensive introductions to SMT.

2.2.2 MaxSAT and OMT

MaxSAT is an optimization problem that extends SAT [ABL13; Ven11]. The goal of MaxSAT is, given a set of clauses, to find a variable assignment that satisfies as many of these clauses as possible. Weighted MaxSAT is a variant of MaxSAT where each clause is assigned a weight, and the goal is no longer to maximize the number of satisfied clauses, but the sum of their weights. Partial Weighted MaxSAT generalizes Weighted MaxSAT by splitting the set of clauses into so-called hard and soft clauses: hard clauses are not weighted and must be satisfied, while soft clauses may be falsified. In this setting, the goal becomes to satisfy all hard clauses while minimizing the weight of falsified soft clauses.

Analogous optimization problems exist for SMT, namely MaxSMT and the weighted and partial weighted variants thereof. Additionally, in the context of the background theories of real or integer arithmetic, it may be desirable to find an interpretation that not only satisfies an SMT constraint system, but also maximizes or minimizes a linear objective function over numeric variables [BP14; Tom14; Ven11]. This optimization problem is known as OptSMT. The umbrella term OMT refers to all types of optimization problems that extend SMT, including MaxSMT, OptSMT, and combinations of the two.

2.2.3 Tools

Z3 Theorem Prover (Z3) [MB08] is an SMT solver developed at Microsoft Research. Z3 is an industrial-strength tool that supports, among other background theories, EUF and the theories of integer arithmetic, real arithmetic, arrays, and bit-vectors [Web+19]. A few years ago, an extension called νZ [BP14] was added to Z3, offering optimization capabilities on top of Z3's SMT functionality through a weighted MaxSMT module and an OptSMT module. The νZ extension allows the user to specify multiple optimization objectives, and to either combine them lexicographically or as Pareto fronts, or to optimize them independently.

MathSAT5 [Cim+13] is another SMT solver that also supports a wide range of background theories, including EUF and linear arithmetic on integers and rationals. The OMT tool OptiMathSAT [ST20] is based on MathSAT5. Like νZ, OptiMathSAT is capable of solving MaxSMT and OptSMT problems with different types of combinations of optimization objectives.

3 Bounded Model Checking for PHL

In this chapter, we define the bounded model checking problem for a fragment of PHL. This bounded model checking problem is essentially equivalent to the problem of synthesizing a scheduler for an MDP that either satisfies a probabilistic constraint, or that maximizes or minimizes a probability expression. Both variants of this scheduler synthesis problem can be encoded as an SMT constraint system. The better part of this chapter describes a general procedure for constructing this constaint system.

3.1 Two Fragments of PHL

We will start by defining two fragments of PHL. The first fragment of interest contains exactly the PHL formulas of the form

$$\forall\sigma_1 \dots \forall\sigma_n.\, \big(\chi \to c_1 \cdot \mathbb{P}(\varphi_1) + \dots + c_k \cdot \mathbb{P}(\varphi_k) \bowtie c\big) \tag{3.1}$$

where

$$\chi = \forall\pi_1 : \sigma_1 \dots \forall\pi_n : \sigma_n.\, \psi$$

for some quantifier-free formula ψ. The second fragment of PHL that we are interested in contains exactly the formulas of the form

$$\exists\sigma_1 \dots \exists\sigma_n.\, \big(\chi \wedge c_1 \cdot \mathbb{P}(\varphi_1) + \dots + c_k \cdot \mathbb{P}(\varphi_k) \bowtie c\big) \tag{3.2}$$

where χ is of the same form as above. By negating a PHL formula Φ of the form (3.1), we obtain the formula

$$\neg\Phi = \exists\sigma_1 \dots \exists\sigma_n.\, \big(\chi \wedge c_1 \cdot \mathbb{P}(\varphi_1) + \dots + c_k \cdot \mathbb{P}(\varphi_k) \not\bowtie c\big).$$

of the form (3.2), and vice versa. We will therefore refer to PHL formulas of the form (3.1) as *positive* PHL formulas, and to formulas of the form (3.2) as *negative* PHL formulas. The set of all positive PHL formulas will be referred to as the *positive* PHL fragment, the set of all negative PHL formulas as the *negative* PHL fragment.

3.2 Bounded Model Checking Problem

Ideally, we would like to solve the model checking problem for the negative PHL fragment. Intuitively, given an MDP M and a PHL formula Φ of the form (3.2), we want to determine whether M satisfies Φ, i.e. whether there exist assignments $\mathfrak{S}_1, \dots, \mathfrak{S}_n$ for scheduler variables $\sigma_1, \dots, \sigma_n$ that make both χ and $c_1 \cdot \mathbb{P}(\varphi_1) + \dots + c_k \cdot \mathbb{P}(\varphi_k) \bowtie c$ evaluate to true on M. Note that this is equivalent to determining whether M satisfies $\neg\Phi$: if we can find adequate assignments $\mathfrak{S}_1, \dots, \mathfrak{S}_n$ for $\sigma_1, \dots, \sigma_n$ to prove that M satisfies Φ, then we can also view $\mathfrak{S}_1, \dots, \mathfrak{S}_n$ as a counterexample for disproving that M satisfies $\neg\Phi$.

Dimitrova, Finkbeiner, and Torfah [DFT20] have proved that the model checking problem for the negative PHL fragment is generally undecidable.

However, we can restrict the problem to so-called deterministic finite-memory schedulers and additionally impose an upper bound on their size. This way, there will only be a finite number of possible schedulers, and the problem will become decidable. We will call this restricted problem the *bounded model checking problem* for the negative PHL fragment.

Intuitively, a finite-memory scheduler is a scheduler that can only remember a bounded amount of information about the execution so far. Formally, we represent the memory of a finite-memory scheduler as a finite-state machine and update its state on every step of the MDP. For general finite-memory schedulers, the memory is updated based on its own current state, the state of the MDP before it makes its step, and the action that is chosen for the step of the MDP. However, we are only interested in deterministic finite-memory schedulers. A scheduler is deterministic if it only assigns probabilities of either 1 or 0 to action choices, which makes these choices essentially deterministic. In our setting, we can therefore simplify the memory update function by leaving out the chosen action. Furthermore, it suffices for the scheduler to output one action instead of a probability mass function over all possible actions.

Since we are only dealing with deterministic schedulers, we will also take the liberty of modifying the way schedulers are applied to MDPs. In addition to the labels of the MDP's current state, we will also label each state of the DTMC induced by the scheduler with the action that will be chosen next. This enables us to use actions like atomic propositions in specifications, which might prove beneficial for many applications: after all, most of the examples from Section 3.1 in [DFT20] require some way of talking about actions in specifications. We will also make use of this in Chapter 4.

Definition 16 (Deterministic Finite-Memory Scheduler).
Let $M = (S, \mathit{Act}, \mathrm{P}, \iota, \mathsf{AP}, L)$ be an MDP with $\mathit{Act} \cap \mathsf{AP} = \emptyset$. A *deterministic finite-memory scheduler* for M is a tuple $\mathfrak{S} = (Q, \delta, q_0, \mathit{act})$ where

- Q is a finite, nonempty set of *states*,
- $\delta : Q \times S \to Q$ is the *memory update function*,
- $q_0 \in Q$ is the *initial state*, and
- $\mathit{act} : Q \times S \to \mathit{Act}$ is the *action choice function*

such that

i) for all $q \in Q$ and $s \in S$ we have $\sum_{s' \in S} \mathrm{P}\big(s, \mathit{act}(q, s), s'\big) > 0$.

The *size* of $\mathfrak{S}$ is denoted by $|\mathfrak{S}|$ and defined as $|\mathfrak{S}| = |Q|$. Scheduler $\mathfrak{S}$ *induces* the DTMC $M_{\mathfrak{S}} = (S \times Q, \mathrm{P}_{\mathfrak{S}}, \iota_{\mathfrak{S}}, \mathsf{AP} \cup \mathit{Act}, L_{\mathfrak{S}})$ where for all $s, s' \in S$ and $q, q' \in Q$ we have

ii) $\mathrm{P}_{\mathfrak{S}}\big((s, q), (s', q')\big) = \begin{cases} \mathrm{P}\big(s, \mathit{act}(q, s), s'\big) & \text{if } q' = \delta(q, s) \\ 0 & \text{otherwise} \end{cases}$,

iii) $\iota_{\mathfrak{S}}\big((s, q)\big) = \begin{cases} \iota(s) & \text{if } q = q_0 \\ 0 & \text{otherwise} \end{cases}$, and

iv) $L_{\mathfrak{S}}\big((s, q)\big) = L(s) \cup \{\mathit{act}(q, s)\}$.

Remark. We must require i) in order to ensure that $\mathfrak{S}$ only chooses actions that are enabled in the current state of M.

Definition 17 (n-Self-Composition of an MDP)**.** Let $M = (S, Act, \mathrm{P}, \iota, \mathsf{AP}, L)$ be an MDP and $n \in \mathbb{N}^+$ a constant. The *n-self-composition of M* is the MDP $M^n = (S^n, Act^n, \widehat{\mathrm{P}}, \widehat{\iota}, \widehat{\mathsf{AP}}, \widehat{L})$ where

- $\widehat{\mathsf{AP}} = \{a_1, \dots, a_n : a \in \mathsf{AP}\}$ contains indexed versions of all atomic propositions

and for all $s = (s_1, \dots, s_n), s' = (s'_1, \dots, s'_n) \in S^n$ and $a = (a_1, \dots, a_n) \in Act^n$ we have

- $\widehat{\mathrm{P}}(s, a, s') = \prod_{i=1}^{n} \mathrm{P}(s_i, a_i, s'_i)$,
- $\widehat{\iota}(s) = \begin{cases} \iota(s_1) & \text{if } s_1 = \dots = s_n \\ 0 & \text{otherwise} \end{cases}$, and
- $\widehat{L}(s) = \displaystyle\bigcup_{i=1}^{n} \{a_i : a \in L(s_i)\}$.

Let $\mathfrak{S}_1 = (Q_1, \delta_1, q_{0_1}, act_1), \dots, \mathfrak{S}_n = (Q_n, \delta_n, q_{0_n}, act_n)$ be finite-memory schedulers for M. The *composition of* $\mathfrak{S}_1, \dots, \mathfrak{S}_n$ is a finite-memory scheduler for M^n that is denoted by $\mathfrak{S}_1 \| \dots \| \mathfrak{S}_n$ and defined as

$$\mathfrak{S}_1 \| \dots \| \mathfrak{S}_n = \big(Q_1 \times \dots \times Q_n, \delta, (q_{0_1}, \dots, q_{0_n}), act\big)$$

where for all $q = (q_1, \dots, q_n) \in Q_1 \times \dots \times Q_n$ and $s = (s_1, \dots, s_n) \in S^n$ we have

- $\delta(q, s) = \big(\delta_1(q_1, s_1), \dots, \delta_n(q_n, s_n)\big)$
- $act(q, s) = \big(act_1(q_1, s_1), \dots, act_n(q_n, s_n)\big)$

Remark. Note that $\widehat{\iota}$ is defined in the same way as in [DFT20]. However, since $\widehat{\iota}$ will only be used to define constants in our encoding, alternative definitions can be used without changing the structure of the constraint system.

We can now formally state the bounded model checking problem for the negative PHL fragment as follows. Given an MDP $M = (S, Act, \mathrm{P}, \iota, \mathsf{AP}, L)$, a PHL formula Φ of the form (3.2), and bounds $u_1, \dots, u_n > 0$, the *bounded model checking problem* for M, Φ, and $u_1, \dots, u_n$ is to decide whether there exist deterministic finite-memory schedulers $\mathfrak{S}_1, \dots, \mathfrak{S}_n$ for M such that

i) for all $i \in \{1, \dots, n\}$ we have $|\mathfrak{S}_i| = u_i$, and

ii) $M^n_{\mathfrak{S}_1 \| \dots \| \mathfrak{S}_n}$ satisfies $\chi \wedge c_1 \cdot \mathbb{P}(\varphi_1) + \cdots c_k \cdot \mathbb{P}(\varphi_k) \bowtie c$.

Hereinafter, we will refer to schedulers that fulfill ii) as *adequate* schedulers. It may be worth pointing out that i) also allows for schedulers $\mathfrak{S}_i$ of smaller size than u_i since smaller schedulers can always be extended by adding new, unreachable memory states until $|\mathfrak{S}_i| = u_i$.

3.3 SMT Encoding

Let $M = (S, Act, \mathrm{P}, \iota, \mathsf{AP}, L)$ be an MDP. Let furthermore Φ be a PHL formula of the form (3.2), and $u_1, \ldots, u_n > 0$ be bounds. We will now construct an SMT constraint system $\mathcal{F}$ that is satisfiable if, and only if, there exist adequate schedulers $\mathfrak{S}_1, \ldots, \mathfrak{S}_n$ with $|\mathfrak{S}_i| = u_i$ for $1 \leq i \leq n$. Our constraint system is of the form

$$\mathcal{F} \;=\; \mathcal{F}_{dtmc} \wedge \mathcal{F}_{hyper} \wedge \mathcal{F}_{prob}$$

where

- $\mathcal{F}_{dtmc}$ encodes the DTMC $M^n_{\mathfrak{S}}$ with $\mathfrak{S} = \mathfrak{S}_1 \| \cdots \| \mathfrak{S}_n$ using uninterpreted function symbols to represent the memory update functions and action choice functions of $\mathfrak{S}_1, \ldots, \mathfrak{S}_n$,
- $\mathcal{F}_{hyper}$ encodes that $M^n_{\mathfrak{S}}$ satisfies the hyperproperty χ, and
- $\mathcal{F}_{prob}$ encodes that $M^n_{\mathfrak{S}}$ satisfies $c_1 \cdot \mathbb{P}(\varphi_1) + \cdots + c_k \cdot \mathbb{P}(\varphi_k) \bowtie c$.

In this encoding, we use constant symbols to represent objects from the underlying mathematical model, such as states and actions of an MDP. We will use a bold, upright font for symbols in the SMT encoding, while variables from the mathematical model will be typeset in the usual font (in most cases italic). For example, if $x \in S$ represents a state of M, then we will use $\mathbf{x}$ to refer to the constant symbol for this state.

When we introduce constant symbols to represent the elements of some set X from the underlying mathematical model, we need to ensure that each of these constant symbols will be interpreted as a unique value. This can be achieved through the constraint

$$\mathcal{F}_{unique}(X) \;=\; \bigwedge_{\substack{\{x,y\} \subseteq X \\ x \neq y}} \mathbf{x} \neq \mathbf{y}.$$

In the following sections, we will explicitly introduce several new sorts to represent sets from the underlying mathematical model. In addition, we will use the sorts **Real** for rational numbers, **Int** for integers, and **Bool** for Booleans, as well as the standard operators on these sorts, which we assume to be built into the SMT solver.

With these preliminaries in mind, we will now see in detail how the constraints $\mathcal{F}_{dtmc}$, $\mathcal{F}_{hyper}$, and $\mathcal{F}_{prob}$ are constructed.

3.3.1 $\mathcal{F}_{dtmc}$

In order to encode $M^n_{\mathfrak{S}}$, we need to represent some of the components of

- $M = (S, Act, \mathrm{P}, \iota, \mathsf{AP}, L)$,
- $M^n = (S^n, Act^n, \widehat{\mathrm{P}}, \widehat{\iota}, \widehat{\mathsf{AP}}, \widehat{L})$,
- $\mathfrak{S}_i = (Q_i, \delta_i, q_{0_i}, act_i)$ for $1 \leq i \leq n$,
- $\mathfrak{S} = \mathfrak{S}_1 \| \cdots \| \mathfrak{S}_n = (Q, \delta, q_0, act)$, and

- $M^n_{\mathfrak{S}} = (S^n \times Q, \widehat{\mathrm{P}}_{\mathfrak{S}}, \widehat{\iota}_{\mathfrak{S}}, \widehat{\mathsf{AP}} \cup Act^n, \widehat{L}_{\mathfrak{S}})$.

Therefore, we first declare the new sorts

- $\mathbf{S}$ to represent S,
- $\mathbf{Act}$ to represent Act,
- $\mathbf{S^n}$ to represent S^n,
- $\mathbf{Act^n}$ to represent Act^n,
- $\mathbf{Q_i}$ for $1 \leq i \leq n$ to represent Q_i, and
- $\mathbf{Q}$ to represent Q

and we introduce constant symbols for the elements of the corresponding sets. Then we introduce the function symbols

- $\mathbf{p} : \mathbf{S^n} \times \mathbf{Act^n} \times \mathbf{S^n} \to \mathbf{Real}$ to represent $\widehat{\mathrm{P}}$,
- $\mathbf{d_i} : \mathbf{Q_i} \times \mathbf{S} \to \mathbf{Q_i}$ for $1 \leq i \leq n$ to represent δ_i,
- $\mathbf{act_i} : \mathbf{Q_i} \times \mathbf{S} \to \mathbf{Act}$ for $1 \leq i \leq n$ to represent act_i,
- $\mathbf{d} : \mathbf{Q} \times \mathbf{S^n} \to \mathbf{Q}$ to represent δ,
- $\mathbf{act} : \mathbf{Q} \times \mathbf{S^n} \to \mathbf{Act^n}$ to represent act, and
- $\mathbf{p_s} : \mathbf{S^n} \times \mathbf{Q} \times \mathbf{S^n} \times \mathbf{Q} \to \mathbf{Real}$ to represent $\widehat{\mathrm{P}}_{\mathfrak{S}}$.

Using these sorts and symbols, we can now construct the constraint $\mathcal{F}_{dtmc}$. It is of the form

$$\mathcal{F}_{dtmc} \;=\; \mathcal{F}_{uniq} \wedge \mathcal{F}_{sched} \wedge \mathcal{F}_{d} \wedge \mathcal{F}_{act} \wedge \mathcal{F}_{p} \wedge \mathcal{F}_{p_s}$$

where

- $\mathcal{F}_{uniq}$ requires all constant symbols to be interpreted as unique values,
- $\mathcal{F}_{sched}$ encodes that schedulers $\mathfrak{S}_1, \ldots, \mathfrak{S}_n$ fulfill the requirements imposed by the definition of deterministic finite-memory schedulers,
- $\mathcal{F}_{d}$ encodes that the memory update function δ of $\mathfrak{S}$ matches the memory update functions $\delta_1, \ldots, \delta_n$ of $\mathfrak{S}_1, \ldots, \mathfrak{S}_n$,
- $\mathcal{F}_{act}$ encodes that the action choice function act of $\mathfrak{S}$ matches the action choice functions $act_1, \ldots, act_n$ of $\mathfrak{S}_1, \ldots, \mathfrak{S}_n$,
- $\mathcal{F}_{p}$ encodes the transition probability function of M^n, and
- $\mathcal{F}_{p_s}$ encodes that the transition probability function of $M^n_{\mathfrak{S}}$ follows the action choices of $\mathfrak{S}$.

The constraint $\mathcal{F}_{uniq}$ is given by

$$\mathcal{F}_{uniq} \;=\; \bigwedge_{X \in \{S, Act, S^n, Act^n, Q\}} \mathcal{F}_{unique}(X) \;\wedge\; \bigwedge_{i=1}^{n} \mathcal{F}_{unique}(Q_i).$$

We will leave the interpretations of $\mathbf{d_i}$ and $\mathbf{act_i}$ for $1 \leq i \leq n$ as open as possible. However, in order to obtain valid schedulers, we must require that the interpretations respect the ranges of the corresponding functions δ_i and act_i respectively, and that $\mathbf{act_i}$ always chooses actions that are enabled in the corresponding state of M. This can be encoded as

$$\begin{aligned} \mathcal{F}_{sched} \;=\; \bigwedge_{i=1}^{n} \bigwedge_{q \in Q_i} \bigwedge_{s \in S} \Big(\bigvee_{q' \in Q_i} & \mathbf{d_i(q,s) = q'} \Big) \\ \wedge \Big(& \bigvee_{\substack{a \in Act \\ \sum_{s' \in S} \mathrm{P}(s,a,s') > 0}} \mathbf{act_i(q,s) = a} \Big). \end{aligned}$$

The constraints $\mathcal{F}_d$ and $\mathcal{F}_{act}$ are given by

$$\begin{aligned} \mathcal{F}_d \;=\; \bigwedge_{q_1 \in Q_1} \cdots \bigwedge_{q_n \in Q_n} \bigwedge_{s_1 \in S} \cdots \bigwedge_{s_n \in S} \bigwedge_{q'_1 \in Q_1} \cdots \bigwedge_{q'_n \in Q_n} & \Big(\Big(\bigwedge_{i=1}^{n} \mathbf{d_i(q_i, s_i) = q'_i} \Big) \\ & \to \mathbf{d}\big((\mathbf{q_1}, \ldots, \mathbf{q_n}), (\mathbf{s_1}, \ldots, \mathbf{s_n})\big) = (\mathbf{q'_1}, \ldots, \mathbf{q'_n}) \Big) \end{aligned}$$

and

$$\begin{aligned} \mathcal{F}_{act} \;=\; \bigwedge_{q_1 \in Q_1} \cdots \bigwedge_{q_n \in Q_n} \bigwedge_{s_1 \in S} \cdots \bigwedge_{s_n \in S} \bigwedge_{a_1 \in Act} \cdots \bigwedge_{a_n \in Act} & \Big(\Big(\bigwedge_{i=1}^{n} \mathbf{act_i(q_i, s_i) = a_i} \Big) \\ & \to \mathbf{act}\big((\mathbf{q_1}, \ldots, \mathbf{q_n}), (\mathbf{s_1}, \ldots, \mathbf{s_n})\big) = (\mathbf{a_1}, \ldots, \mathbf{a_n}) \Big) \end{aligned}$$

where tuples of constant symbols for states or actions are shorthand for the constant symbol for the corresponding tuple of states or actions. Finally, we fix the interpretations of $\mathbf{p}$ and $\mathbf{p_s}$ through the constraints

$$\mathcal{F}_p \;=\; \bigwedge_{s \in S^n} \bigwedge_{a \in Act^n} \bigwedge_{s' \in S^n} \mathbf{p(s, a, s')} = \widehat{\mathrm{P}}(s, a, s')$$

and

$$\begin{aligned} \mathcal{F}_{p_s} \;=\; \bigwedge_{s \in S^n} \bigwedge_{q \in Q} \bigwedge_{s' \in S^n} \bigwedge_{q' \in Q} & \Big(\mathbf{d(q,s) \neq q'} \;\to\; \mathbf{p_s(s, q, s', q')} = 0 \Big) \\ & \wedge \Big(\mathbf{d(q,s) = q'} \;\to\; \mathbf{p_s(s, q, s', q')} = \mathbf{p}\big(\mathbf{s}, \mathbf{act(q,s)}, \mathbf{s'}\big) \Big). \end{aligned}$$

3.3.2 $\mathcal{F}_{hyper}$

In order to encode that $M^n_{\mathfrak{S}}$ satisfies χ, we will use the main idea behind the HyperLTL synthesis algorithm from [Fin+20]. This algorithm first constructs a universal co-Büchi automaton $\mathcal{B}$ for the LTL suffix of the HyperLTL formula and then encodes in SMT that all runs of the self-composition of the synthesized system on $\mathcal{B}$ are accepting. The SMT encoding tries to establish an order on the reachable rejecting states of the product DTMC of the synthesized system and $\mathcal{B}$. This is achieved by determining for every reachable state (an overapproximation of) the maximal number of rejecting states that have been visited when the state is reached.

Let $\mathcal{B} = (B, 2^{\widehat{\mathsf{AP}} \cup Act^n}, \delta_{\mathcal{B}}, b_0, F)$ denote the universal co-Büchi automaton for the LTL suffix ψ of χ. We declare the new sort $\mathbf{B}$ to represent B and introduce constant symbols for the elements of B. Furthermore, we introduce the function symbols

– $\mathbf{reach} : \mathbf{S^n} \times \mathbf{Q} \times \mathbf{B} \to \mathbf{Bool}$ to encode the reachability of states, and

– $\mathbf{count} : \mathbf{S^n} \times \mathbf{Q} \times \mathbf{B} \to \mathbf{Int}$ to encode the rejecting state count of reachable states.

The constraint $\mathcal{F}_{hyper}$ is of the form

$$\mathcal{F}_{hyper} \;=\; \mathcal{F}_{unique}(B) \wedge \mathcal{F}_{init} \wedge \mathcal{F}_{step}$$

where

– $\mathcal{F}_{unique}(B)$ requires the constant symbols for the states of $\mathcal{B}$ to be interpreted as unique values,

– $\mathcal{F}_{init}$ encodes that the initial states of $M^n_{\mathfrak{S}} \otimes \mathcal{B}$ are reachable, and

– $\mathcal{F}_{step}$ encodes that if a state of $M^n_{\mathfrak{S}} \otimes \mathcal{B}$ is reachable, then its successors are reachable as well, and their order is correct.

The constraint $\mathcal{F}_{init}$ is given by

$$\mathcal{F}_{init} \;=\; \bigwedge_{\substack{s \in S^n \\ \widehat{\iota}(s) > 0}} \mathbf{reach}(\mathbf{s}, \mathbf{q_0}, \mathbf{b_0}).$$

Following closely the constraint presented in [Fin+20], we encode the intended semantics of **reach** and **count** through the constraint

$$\begin{aligned}\mathcal{F}_{step} \;=\; \bigwedge_{s \in S^n} \bigwedge_{q \in Q} \bigwedge_{b \in B} \bigwedge_{s' \in S^n} \bigwedge_{q' \in Q} \bigwedge_{b' \in B} \bigwedge_{a \in Act^n} \Bigg(& \mathbf{act}(\mathbf{q}, \mathbf{s}) = \mathbf{a} \\ \to \Bigg(& \mathbf{reach}(\mathbf{s}, \mathbf{q}, \mathbf{b}) \;\wedge\; \mathbf{p}(\mathbf{s}, \mathbf{q}, \mathbf{s'}, \mathbf{q'}) > 0 \;\wedge\; b' \in \delta_{\mathcal{B}}(b, \widehat{L}(s) \cup a) \\ & \to\; \mathbf{reach}(\mathbf{s'}, \mathbf{q'}, \mathbf{b'}) \;\wedge\; \mathbf{count}(\mathbf{s'}, \mathbf{q'}, \mathbf{b'}) \geq \mathbf{count}(\mathbf{s}, \mathbf{q}, \mathbf{b}) \\ & \wedge \Big(b' \in F \;\to\; \mathbf{count}(\mathbf{s'}, \mathbf{q'}, \mathbf{b'}) > \mathbf{count}(\mathbf{s}, \mathbf{q}, \mathbf{b}) \Big) \Bigg) \Bigg).\end{aligned}$$

3.3.3 $\mathcal{F}_{prob}$

With this constraint, we want to encode that $M^n_{\mathfrak{G}}$ satisfies $c_1 \cdot \mathbb{P}(\varphi_1) + \cdots + c_k \cdot \mathbb{P}(\varphi_k) \bowtie c$. For each LTL formula $\varphi_1, \ldots, \varphi_k$ from the probabilistic constraint, $\mathcal{F}_{prob}$ contains an encoding of the execution of part of the probabilistic model checking algorithm for LTL formulas on DTMCs as presented in Section 10.3 of [BK08].

The first step of this algorithm is to compute the DRA for the LTL formula under consideration. Since this computation does not depend on the DTMC $M^n_{\mathfrak{G}}$, we can perform it before we construct our SMT encoding. For the remainder of this section, we will therefore assume that the DRAs for $\varphi_1, \ldots, \varphi_k$ have already been constructed. Hereinafter, the DRA for φ_i will be denoted by $\mathcal{R}_i = (R_i, 2^{\widehat{\mathsf{AP} \cup Act^n}}, \delta_{\mathcal{R}_i}, r_{0_i}, Acc_i)$ with $Acc_i = (B_{i_j}, G_{i_j})_{j=1}^{m_i}$ for $1 \leq i \leq k$. We declare the new sorts $\mathbf{R_i}$ to represent R_i for $1 \leq i \leq k$ and introduce constant symbols for the elements of these sets.

Next, the algorithm constructs the product DTMC of the input DTMC, in our case $M^n_{\mathfrak{G}}$, and the DRA. We introduce the function symbol

$$\mathbf{p_i} : \mathbf{S^n} \times \mathbf{Q} \times \mathbf{R_i} \times \mathbf{S^n} \times \mathbf{Q} \times \mathbf{R_i} \to \mathbf{Real}$$

for $1 \leq i \leq k$ to represent the transition probability function of the product DTMC $M^n_{\mathfrak{G}} \otimes \mathcal{R}_i$.

Then the algorithm computes the union of the accepting BSCCs of the product DTMC. We will not exactly compute the union of accepting BSCCs, but overapproximate it through what we will call the set of *winning* states: a superset of the accepting BSCCs from which the probability of reaching an accepting BSCC equals 1. Finally, we need to solve a system of linear equations to determine the probability of reaching a winning state. In order to make sure that this system of linear equations has a unique solution, we will also need to determine the set of all states from which it is impossible to reach a winning state. We will refer to this set as the set of *losing* states. In order to encode this part of the algorithm, we introduce the function symbols

- $\mathbf{win_i} : \mathbf{S^n} \times \mathbf{Q} \times \mathbf{R_i} \to \mathbf{Bool}$ for $1 \leq i \leq k$ to represent the indicator function of the set of winning states,
- $\mathbf{lose_i} : \mathbf{S^n} \times \mathbf{Q} \times \mathbf{R_i} \to \mathbf{Bool}$ for $1 \leq i \leq k$ to represent the indicator function of the set of losing states, and
- $\mathbf{x_i} : \mathbf{S^n} \times \mathbf{Q} \times \mathbf{R_i} \to \mathbf{Real}$ for $1 \leq i \leq k$ to represent the winning probability when starting in the given state.

Now we can construct the constraint $\mathcal{F}_{prob}$. It is of the form

$$\mathcal{F}_{prob} \;=\; \bigwedge_{i=1}^{k} \left(\mathcal{F}_{dtmc_i} \wedge \mathcal{F}_{win_i} \wedge \mathcal{F}_{lose_i} \wedge \mathcal{F}_{x_i}\right) \;\wedge\; \mathcal{F}_{constraint}$$

where

- $\mathcal{F}_{dtmc_i}$ for $1 \leq i \leq k$ encodes the DTMC $M^n_{\mathfrak{G}} \otimes \mathcal{R}_i$,
- $\mathcal{F}_{win_i}$ for $1 \leq i \leq k$ encodes the set of winning states of $M^n_{\mathfrak{G}} \otimes \mathcal{R}_i$,

- $\mathcal{F}_{lose_i}$ for $1 \leq i \leq k$ encodes the set of losing states in $M^n_{\mathfrak{G}} \otimes \mathcal{R}_i$,
- $\mathcal{F}_{x_i}$ for $1 \leq i \leq k$ encodes the winning probabilities for the states of $M^n_{\mathfrak{G}} \otimes \mathcal{R}_i$, and
- $\mathcal{F}_{constraint}$ encodes the constraint $c_1 \cdot \mathbb{P}(\varphi_1) + \cdots + c_k \cdot \mathbb{P}(\varphi_k) \bowtie c$.

The constraint $\mathcal{F}_{dtmc_i}$ for $1 \leq i \leq k$ must ensure that the constant symbols for the elements of R_i are interpreted as unique values and that the interpretation of $\mathbf{p_i}$ follows the transition probability function of $M^n_{\mathfrak{G}}$ as well as the transitions of $\mathcal{R}_i$. It is given by

$$\begin{aligned}\mathcal{F}_{dtmc_i} = {} & \mathcal{F}_{unique}(R_i) \\ & \wedge \bigwedge_{s\in S^n} \bigwedge_{q\in Q} \bigwedge_{r\in R_i} \bigwedge_{s'\in S^n} \bigwedge_{q'\in Q} \bigwedge_{r'\in R_i} \bigwedge_{a\in Act^n} \Big(\mathbf{act}(\mathbf{q},\mathbf{s}) = \mathbf{a} \\ & \quad \rightarrow \Big(r' \neq \delta_{\mathcal{R}_i}\big(r, \widehat{L}(s) \cup \{a\}\big) \rightarrow \mathbf{p_i}(\mathbf{s},\mathbf{q},\mathbf{r},\mathbf{s}',\mathbf{q}',\mathbf{r}') = 0 \Big) \\ & \quad \wedge \Big(r' = \delta_{\mathcal{R}_i}\big(r, \widehat{L}(s) \cup \{a\}\big) \\ & \qquad \rightarrow \mathbf{p_i}(\mathbf{s},\mathbf{q},\mathbf{r},\mathbf{s}',\mathbf{q}',\mathbf{r}') = \mathbf{p_s}(\mathbf{s},\mathbf{q},\mathbf{s}',\mathbf{q}') \Big) \Big).\end{aligned}$$

Before we can construct the constraints $\mathcal{F}_{win_i}$ and $\mathcal{F}_{lose_i}$, we must introduce the BSCC approximation that we want to compute. Let us for the moment fix an index $1 \leq i \leq k$. Our overapproximation of the union of the accepting BSCCs of $M^n_{\mathfrak{G}} \otimes \mathcal{R}_i$ is defined as

$$\bigcup_{j=1}^{m_i} \Bigg((S^n \times Q \times R_i) \setminus Pre^* \Big((S^n \times Q \times R_i) \setminus \Big(Pre^*(S^n \times Q \times G_{i_j}) \setminus (S^n \times Q \times B_{i_j}) \Big) \Big) \Bigg)$$

where Pre^* denotes the reflexive and transitive closure of the direct predecessor function on sets of states. We call the states that are contained in this overapproximation the *winning* states. We refer to the states from which no winning state can be reached as the *losing* states. The *probability of winning* from a state s, or the *winning probability* of s for short, is the probability of reaching a winning state from s. By definition, the probability of winning from a winning state is 1, and the probability of winning from a losing state is 0. Intuitively, the winning states are exactly those states $s \in S^n \times Q \times R_i$ for which it holds that

- a good state can be reached from s,
- s is not bad, and
- no state outside the overapproximation can be reached from s.

In Appendix A, we give a proof that the abovedescribed construction is indeed an overapproximation of the union of the accepting BSCCs that will be reached with the same probability.

Since Pre^* can be computed as the fixpoint of the direct predecessor function, the set of winning states can be determined by performing two fixpoint computations for each $1 \leq j \leq m_i$. Since the fixpoint must be reached after at most $max = |S^n \times R_i| - 1$ iterations, we can encode these fixpoint computations in SMT by unrolling the first max iterations. In order to represent the fixpoints, we introduce for each $1 \leq j \leq m_i$

- $\mathbf{win'_{ij}} : \mathbf{S^n} \times \mathbf{Q} \times \mathbf{R_i} \to \mathbf{Bool}$ to represent the first fixpoint, and
- $\mathbf{win_{ij}} : \mathbf{S^n} \times \mathbf{Q} \times \mathbf{R_i} \to \mathbf{Bool}$ to represent the second fixpoint

and additionally for each $1 \leq h \leq max - 1$

- $\mathbf{win'_{ij_{\mathit{h}}}} : \mathbf{S^n} \times \mathbf{Q} \times \mathbf{R_i} \to \mathbf{Bool}$ to represent the iterations of the first computation, and
- $\mathbf{win_{ij_{\mathit{h}}}} : \mathbf{S^n} \times \mathbf{Q} \times \mathbf{R_i} \to \mathbf{Bool}$ to represent the iterations of the second computation.

In the presentation of the following constraints, we will use the notation $\mathbf{win'_{ij_{\mathit{max}}}}$ for $\mathbf{win'_{ij}}$ and $\mathbf{win_{ij_{\mathit{max}}}}$ for $\mathbf{win_{ij}}$. With these auxiliary function symbols, we can encode the fixpoints $\mathbf{win'_{ij}}$ and $\mathbf{win_{ij}}$ through the constraints

$$\begin{aligned}\mathcal{F}_{win'_{ij}} = \bigwedge_{s\in S^n} \bigwedge_{q\in Q} \bigwedge_{r\in R_i} \Bigg(\Big(& \mathbf{win'_{ij_1}}(\mathbf{s},\mathbf{q},\mathbf{r}) \leftrightarrow r \in G_{i_j} \Big) \\ \wedge \bigwedge_{h=2}^{max} \Big(& \mathbf{win'_{ij_{\mathit{h}}}}(\mathbf{s},\mathbf{q},\mathbf{r}) \leftrightarrow \mathbf{win'_{ij_{\mathit{h}-1}}}(\mathbf{s},\mathbf{q},\mathbf{r}) \\ & \vee \bigvee_{s'\in S^n} \bigvee_{q'\in Q} \bigvee_{r'\in R_i} (\mathbf{p_i}(\mathbf{s},\mathbf{q},\mathbf{r},\mathbf{s'},\mathbf{q'},\mathbf{r'}) > 0 \\ & \qquad \wedge \mathbf{win'_{ij_{\mathit{h}-1}}}(\mathbf{s'},\mathbf{q'},\mathbf{r'})) \Big) \Bigg) \end{aligned}$$

and

$$\begin{aligned}\mathcal{F}_{win_{ij}} = \bigwedge_{s\in S^n} \bigwedge_{q\in Q} \bigwedge_{r\in R_i} \Bigg(\Big(& \mathbf{win_{ij_1}}(\mathbf{s},\mathbf{q},\mathbf{r}) \leftrightarrow \mathbf{win'_{ij}}(\mathbf{s},\mathbf{q},\mathbf{r}) \wedge r \notin B_{i_j} \Big) \\ \wedge \bigwedge_{h=2}^{max} \Big(& \mathbf{win_{ij_{\mathit{h}}}}(\mathbf{s},\mathbf{q},\mathbf{r}) \leftrightarrow \mathbf{win_{ij_{\mathit{h}-1}}}(\mathbf{s},\mathbf{q},\mathbf{r}) \\ & \wedge \bigwedge_{s'\in S^n} \bigwedge_{q'\in Q} \bigwedge_{r'\in R_i} (\mathbf{p_i}(\mathbf{s},\mathbf{q},\mathbf{r},\mathbf{s'},\mathbf{q'},\mathbf{r'}) > 0 \\ & \qquad \to \mathbf{win_{ij_{\mathit{h}-1}}}(\mathbf{s'},\mathbf{q'},\mathbf{r'})) \Big) \Bigg). \end{aligned}$$

The set of winning states can now be encoded through the constraint

$$\begin{aligned}\mathcal{F}_{win_i} = & \bigwedge_{j=1}^{m_i} \left(\mathcal{F}_{win'_{ij}} \wedge \mathcal{F}_{win_{ij}} \right) \\ & \wedge \bigwedge_{s\in S^n} \bigwedge_{q\in Q} \bigwedge_{r\in R_i} \Big(\mathbf{win_i}(\mathbf{s},\mathbf{q},\mathbf{r}) \leftrightarrow \bigvee_{j=1}^{m_i} \mathbf{win_{ij}}(\mathbf{s},\mathbf{q},\mathbf{r}) \Big). \end{aligned}$$

In the last step of the algorithm, we compute the winning probabilities of all states of $M^n_{\mathfrak{G}} \otimes \mathcal{R}_i$ by solving a linear equation system as presented in

Section 10.1.1 of [BK08]. However, as indicated earlier, we can only be sure that this equation system has a unique solution if we explicitly require all losing states to have a winning probability of 0. Using a similar fixpoint construction as for $\mathcal{F}_{win_i}$ and auxiliary function symbols

$$\mathbf{lose_{i_h}} : \mathbf{S^n} \times \mathbf{Q} \times \mathbf{R_i} \to \mathbf{Bool}$$

for $1 \leq h \leq max - 1$ and the notation $\mathbf{lose_{i_{max}}}$ for $\mathbf{lose_i}$ we can encode the losing states through the constraint

$$\begin{aligned}\mathcal{F}_{lose_i} \;=\; \bigwedge_{s\in S^n} \bigwedge_{q\in Q} \bigwedge_{r\in R_i} \Bigg(\Big(& \mathbf{lose_{i_1}}(\mathbf{s},\mathbf{q},\mathbf{r}) \;\leftrightarrow\; \neg\mathbf{win_i}(\mathbf{s},\mathbf{q},\mathbf{r}) \Big) \\ & \wedge \bigwedge_{h=2}^{max} \Big(\mathbf{lose_{i_h}}(\mathbf{s},\mathbf{q},\mathbf{r}) \;\leftrightarrow\; \mathbf{lose_{i_{h-1}}}(\mathbf{s},\mathbf{q},\mathbf{r}) \\ & \qquad \wedge \bigwedge_{s'\in S^n} \bigwedge_{q'\in Q} \bigwedge_{r'\in R_i} (\mathbf{p_i}(\mathbf{s},\mathbf{q},\mathbf{r},\mathbf{s'},\mathbf{q'},\mathbf{r'}) > 0 \\ & \qquad\qquad \to \mathbf{lose_{i_{h-1}}}(\mathbf{s'},\mathbf{q'},\mathbf{r'}))\Big)\Bigg).\end{aligned}$$

Now we can encode the linear equation system that will define the winning probabilities, i.e. the interpretation of $\mathbf{x_i}$, through the constraint

$$\begin{aligned}\mathcal{F}_{x_i} \;=\; \bigwedge_{s\in S^n} \bigwedge_{q\in Q} \bigwedge_{r\in R_i} \Bigg(\Big(& \mathbf{lose_i}(\mathbf{s},\mathbf{q},\mathbf{r}) \;\to\; \mathbf{x_i}(\mathbf{s},\mathbf{q},\mathbf{r}) = 0 \Big) \\ \wedge\; & \Big(\mathbf{win_i}(\mathbf{s},\mathbf{q},\mathbf{r}) \;\to\; \mathbf{x_i}(\mathbf{s},\mathbf{q},\mathbf{r}) = 1 \Big) \\ \wedge\; & \Big(\neg\mathbf{lose_i}(\mathbf{s},\mathbf{q},\mathbf{r}) \wedge \neg\mathbf{win_i}(\mathbf{s},\mathbf{q},\mathbf{r}) \;\to\; \mathbf{x_i}(\mathbf{s},\mathbf{q},\mathbf{r}) = \\ & \qquad \sum_{s'\in S^n} \sum_{q'\in Q} \sum_{r'\in R_i} \mathbf{p_i}(\mathbf{s},\mathbf{q},\mathbf{r},\mathbf{s'},\mathbf{q'},\mathbf{r'}) \cdot \mathbf{x_i}(\mathbf{s'},\mathbf{q'},\mathbf{r'})\Big)\Bigg).\end{aligned}$$

Finally, we encode that $M^n_{\mathfrak{G}}$ satisfies $c_1 \cdot \mathbb{P}(\varphi_1) + \cdots + c_k \cdot \mathbb{P}(\varphi_k) \bowtie c$ through the constraint

$$\mathcal{F}_{constraint} \;=\; \left(\sum_{i=1}^{k} c_i \cdot \Big(\sum_{s\in S^n} \widehat{\iota}(s) \cdot \mathbf{x_i}(\mathbf{s},\mathbf{q_0},\mathbf{r_{0_i}}) \Big) \right) \bowtie c$$

where q_0 is the initial state of $\mathfrak{G}$ and r_{0_i} is the initial state of $\mathcal{R}_i$.

3.4 Finding Optimal Schedulers

Ideally, we would also like to synthesize optimal schedulers. At first glance, this should be possible by replacing the constraint $\mathcal{F}_{constraint}$ with an optimization objective, since $c_1 \cdot \mathbb{P}(\varphi_1) + \cdots + c_k \cdot \mathbb{P}(\varphi_k)$ is a linear expression. However, the formulation of $\mathcal{F}_{x_i}$ in Section 3.3 includes multiplication of multiple SMT variables with each other, namely of values of $\mathbf{p_i}$ with values of $\mathbf{x_i}$. This makes it impossible to use optimization of linear objective functions on the resulting probabilities, because those are subject to nonlinear constraints themselves. We can address this problem by introducing a new function symbol

$$\mathbf{px_i} : \mathbf{S^n} \times \mathbf{Q} \times \mathbf{R_i} \times \mathbf{S^n} \times \mathbf{Q} \times \mathbf{R_i} \to \mathbf{Real}$$

with the intended semantics

$$\mathbf{px_i}(\mathbf{s}, \mathbf{q}, \mathbf{r}, \mathbf{s'}, \mathbf{q'}, \mathbf{r'}) = \mathbf{p_i}(\mathbf{s}, \mathbf{q}, \mathbf{r}, \mathbf{s'}, \mathbf{q'}, \mathbf{r'}) \cdot \mathbf{x_i}(\mathbf{s'}, \mathbf{q'}, \mathbf{r'}).$$

However, we encode this without using multiplication of multiple SMT variables. Since $\mathbf{p_i}$ can only take two possible values for each combination of arguments, we can replace it with a case distinction, multiplying the value of $\mathbf{x_i}$ with a constant in each of the two possible cases. We encode this through the constraint

$$\begin{aligned}
\mathcal{F}_{px_i} = \bigwedge_{s \in S^n} \bigwedge_{q \in Q} \bigwedge_{r \in R_i} \bigwedge_{s' \in S^n} \bigwedge_{q' \in Q} \bigwedge_{r' \in R_i} \bigwedge_{a \in Act^n} \Big(& \mathbf{act}(\mathbf{q}, \mathbf{s}) = \mathbf{a} \\
& \to \Big(\mathbf{q'} \neq \mathbf{d}(\mathbf{q}, \mathbf{s}) \ \vee \ r' \neq \delta_i\big(r, \widehat{L}(s) \cup \{a\}\big) \\
& \qquad \to \ \mathbf{px_i}(\mathbf{s}, \mathbf{q}, \mathbf{r}, \mathbf{s'}, \mathbf{q'}, \mathbf{r'}) = 0 \Big) \\
& \wedge \ \Big(\mathbf{q'} = \mathbf{d}(\mathbf{q}, \mathbf{s}) \ \wedge \ r' = \delta_i\big(r, \widehat{L}(s) \cup \{a\}\big) \\
& \qquad \to \ \mathbf{px_i}(\mathbf{s}, \mathbf{q}, \mathbf{r}, \mathbf{s'}, \mathbf{q'}, \mathbf{r'}) = \widehat{\mathrm{P}}(s, a, s') \cdot \mathbf{x_i}(\mathbf{s'}, \mathbf{q'}, \mathbf{r'}) \Big) \Big).
\end{aligned}$$

Now we can enable optimization by conjoining $\mathcal{F}_{px_i}$ to the constraint system from Section 3.3 and replacing the multiplication in $\mathcal{F}_{x_i}$ with the linear expression $\mathbf{px_i}(\mathbf{s}, \mathbf{q}, \mathbf{r}, \mathbf{s'}, \mathbf{q'}, \mathbf{r'})$.

3.5 Potential Optimizations

Intuitively, reducing the size of an SMT constraint system should have the potential of speeding up the process of solving it. In this section, we will briefly discuss two approaches how a reduction in size can be achieved for the SMT encoding from Section 3.3.

The first approach is based on the following observation. Our encoding contains many constraints that begin with a conjunction over all possible pairs of states of M^n. However, we are only interested in pairs of states s and s' of M^n for which there exists an action a such that $\widehat{\mathrm{P}}(s, a, s') > 0$. By consistently modifying all constraints such that only pairs of states with the possibility of a transition occur, the size of the SMT encoding can be reduced for most MDPs. Since this does not affect the general structure of the encoding but only leaves out unnecessary constraints, we can expect this approach to have a positive effect on the running time of our algorithm.

The second approach is to eliminate the unrolling of the fixpoint computations in $\mathcal{F}_{prob}$. It turns out that this is indeed possible using a counter-based encoding similar to the one we already use in $\mathcal{F}_{hyper}$. This also has the potential of greatly reducing the size of the SMT encoding. However, in contrast to the first approach, the introduction of a new counter function symbol increases the number of possible interpretations by orders of magnitude. Therefore, it is hard to predict whether or not this approach will bring about any benefit in practice.

4 Proof of Concept

This chapters presents first experimental results with a proof-of-concept implementation of the scheduler synthesis procedure from Chapter 3 for a specific case study, which consists of a scalable MDP and a simple probabilistic hyperproperty. The property is fixed except for one variable parameter, while the MDP only has a fixed general structure, but its size is determined by two parameters. Both the property and the structure of the MDP are hard-coded into our implementation, while the parameters for both the MDP and the property can be freely chosen.

4.1 Case Study

We will start with a description of the case study. It is based on the example problem which Dimitrova, Finkbeiner, and Torfah [DFT20] used to evaluate the performance of their bounded model checking algorithm for a fragment of PHL. The original statement of this problem can be found under *Example 2 (Plan non-interference)* in Section 3.1 of the paper.

4.1.1 Two Robots

We consider the following scenario. Two robots, R_1 and R_2, are moving towards the same goal. The initial distance from the goal is d_1 many steps for R_1 and d_2 many steps for R_2. In every time step, each robot can either do nothing or attempt to make a move. If a robot attempts to make a move, it will succeed with a probability of 50% if it has not yet reached the goal; if the robot has already reached the goal, any attempt to make a move will always fail. If the robot attempts to make a move and succeeds, it will move one step towards the goal; otherwise, it will stay in its current position. If both robots attempt to make a move, the event that R_1 succeeds is independent of the event that R_2 succeeds.

We can model this scenario as an MDP, where each possible combination of the robots' positions is represented as a distinct state. Note that this is only possible because the number of possible positions of each robot is finite. The nondeterministic choice of actions is used to model the robots' decisions whether or not they attempt to make a move, and their success or failure is captured by the probabilistic choice of a corresponding successor state.

Figure 4.1 shows an MDP that models the scenario for parameters $d_1 = 1$ and $d_2 = 2$. Each state in the figure is labeled with both its name and, below the name, the set of atomic propositions that hold in this state. If multiple actions that have the same effect, they are only drawn once, but labeled with multiple action names separated by slashes. The indices in the state names indicate the robots' distance to the goal: s_{ij} is the state where R_1 is i many steps away from the goal and R_2 is j many steps away from the goal. Similarly, the indices in the action names indicate whether or not the robots attempt to make a move: if in action a_{kl} we have $k = 1$, this means that R_1 attempts to

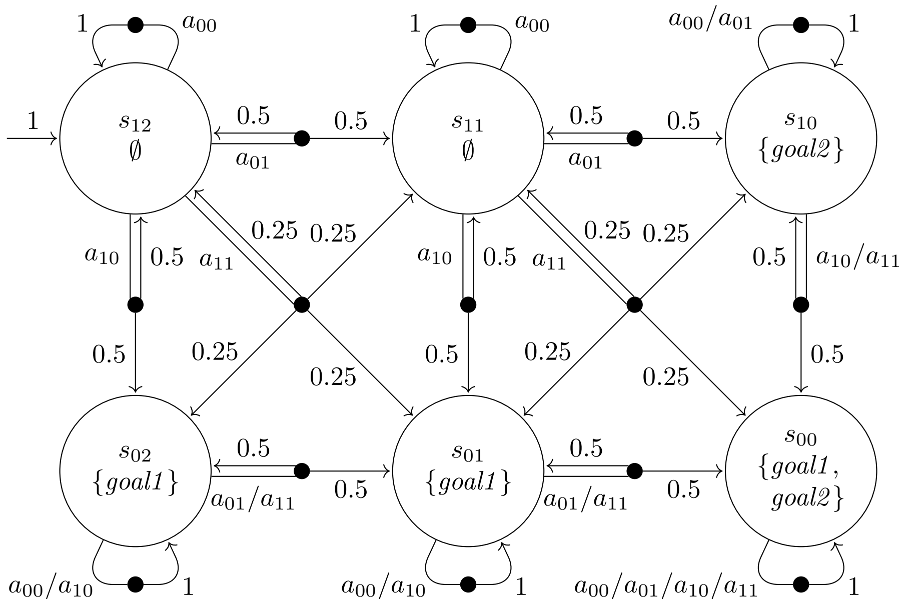

Figure 4.1: MDP modeling the scenario for $d_1 = 1$ and $d_2 = 2$.

make a move, while $k = 0$ means that R_1 does nothing; analogously, l indicates whether R_2 attempts to make a move.

4.1.2 Plan Non-Interference

A *plan* for a robot is a strategy of deciding when to attempt to make a move. In general, a plan may base this decision on the positions of both robots and/or some internal state. We call a plan *deterministic* if it always makes the same decision at the same time step, irrespective of the probabilistic choices in the movement of the robots. Note that the concept of deterministic plans is not related to deterministic schedulers: not every deterministic scheduler for the MDP from Fig. 4.1 corresponds to deterministic plans for the robots, and any kind of scheduler can represent deterministic plans, as long as it always produces observable behavior of the robots that is deterministic in the above-mentioned sense. We say that a robot *wins* if it reaches the goal before the other robot. A deterministic plan for R_1 is *robust against interference* from arbitrary plans for R_2 if the probability that R_1 wins under some plan for R_2 does not deviate by more than some fixed constant ε from the probability that R_1 wins under any other plan for R_2. The property *plan non-interference* requires all deterministic plans for R_1 to be robust against interference from arbitrary plans for R_2.

Plan non-interference can be expressed as a positive PHL formula of the form

$$\forall \sigma_1.\, \forall \sigma_2.\, \chi \to \phi \tag{4.1}$$

where

- σ_1 and σ_2 each represent a combination of a plan for R_1 and a plan for R_2,
- χ expresses that R_1 follows the same deterministic plan under both σ_1 and σ_2, and
- ϕ expresses that the probability of R_1 winning under σ_1 differs by no more than ε from the probability of R_1 winning under σ_2.

Subformula χ is given by

$$\chi = \forall \pi_1 : \sigma_1.\, \forall \pi_2 : \sigma_2.\, \square(\mathit{move1}_{\pi_1} \leftrightarrow \mathit{move1}_{\pi_2})$$

where *move1* is shorthand for $(a_{10} \vee a_{11})$, meaning that an action is chosen which involves R_1 attempting to make a move. Note that χ does not explicitly require the plan for R_1 to be deterministic. However, if the plan depends on the position of a robot that will at some point attempt to make a move, then it cannot satisfy χ. Even if the same scheduler is assigned to both σ_1 and σ_2, as soon as an action is chosen that attempts to move a robot, there will always be traces that differ in the position of this robot.

The property that R_1 wins can be expressed as $\diamond(\mathit{goal1} \wedge \neg \mathit{goal2})$. This is due to the fact that once a robot reaches the goal, it will stay there forever. Consequently, if R_1 does not win, *goal2* will hold whenever *goal1* holds. Thus, the probabilistic constraint ϕ can be expressed as

$$\phi = \Big|\mathbb{P}\big(\diamond(\mathit{goal1}_{\sigma_1} \wedge \neg \mathit{goal2}_{\sigma_1})\big) - \mathbb{P}\big(\diamond(\mathit{goal1}_{\sigma_2} \wedge \neg \mathit{goal2}_{\sigma_2})\big)\Big| \leq \varepsilon.$$

Note that for all reals x and ε it holds that

$$x \leq \varepsilon \;\wedge\; -x \leq \varepsilon \quad \Rightarrow \quad |x| \leq \varepsilon \tag{4.2}$$

Also note that in ϕ, swapping σ_1 and σ_2 results in a change of sign of the operand of the modulus operator. The universal quantification over σ_1 and σ_2 in (4.1) requires the inequality in ϕ to hold for both the variant with a positive sign and the variant with a negative sign. Thus, we can use (4.2) and omit the modulus operator in ϕ, yielding

$$\phi = \mathbb{P}\big(\diamond(\mathit{goal1}_{\sigma_1} \wedge \neg \mathit{goal2}_{\sigma_1})\big) - \mathbb{P}\big(\diamond(\mathit{goal1}_{\sigma_2} \wedge \neg \mathit{goal2}_{\sigma_2})\big) \leq \varepsilon.$$

Putting together the subformulas, we obtain the positive PHL formula

$$\begin{aligned}&\forall \sigma_1.\, \forall \sigma_2.\, \big(\forall \pi_1 : \sigma_1.\, \forall \pi_2 : \sigma_2.\, \square(\mathit{move1}_{\pi_1} \leftrightarrow \mathit{move1}_{\pi_2})\big) \rightarrow \\ &\qquad \mathbb{P}\big(\diamond(\mathit{goal1}_{\sigma_1} \wedge \neg \mathit{goal2}_{\sigma_1})\big) - \mathbb{P}\big(\diamond(\mathit{goal1}_{\sigma_2} \wedge \neg \mathit{goal2}_{\sigma_2})\big) \leq \varepsilon\end{aligned}$$

for plan non-interference.

4.1.3 Problem

In this case study, our goal is to disprove the property of plan non-interference for several instances of the abovedescribed scenario. In other words, we want to prove that the negative PHL formula

$$\begin{aligned}&\exists \sigma_1.\, \exists \sigma_2.\, \big(\forall \pi_1 : \sigma_1.\, \forall \pi_2 : \sigma_2.\, \square(\mathit{move1}_{\pi_1} \leftrightarrow \mathit{move1}_{\pi_2})\big) \wedge \\ &\qquad \mathbb{P}\big(\diamond(\mathit{goal1}_{\sigma_1} \wedge \neg \mathit{goal2}_{\sigma_1})\big) - \mathbb{P}\big(\diamond(\mathit{goal1}_{\sigma_2} \wedge \neg \mathit{goal2}_{\sigma_2})\big) > \varepsilon\end{aligned}$$

holds on several MDPs of different sizes that share the general structure shown in Fig. 4.1.

Instance	Mode of Operation			
	chk/count	chk/unroll	opt/count	opt/unroll
$1{\times}1/1/1$	2.53 s	3.27 s	2.89 s	3.37 s
$1{\times}2/1/1$	6.88 s	8.63 s	7.79 s	9.59 s
$2{\times}2/1/1$	22.30 s	28.00 s	30.93 s	32.63 s
$1{\times}2/2/2$	112.73 s	151.31 s	4232.75 s	2715.44 s
$3{\times}3/1/1$	176.45 s	198.42 s	1262.47 s	1346.34 s
$4{\times}4/1/1$	2253.67 s	2365.21 s	*timeout*	*timeout*

Table 4.1: Running time by instance and mode of operation.

4.2 Prototype

We have built a proof-of-concept implementation of our bounded model checking algorithm specifically for the case study described above. It is written in Python using Z3's Python API. The universal co-Büchi automaton and the DRA for the property of plan non-interference, as well as the general structure of the MDP are hard-coded into this prototype, but the parameters d_1, d_2, u_1, u_2, and ε can be freely chosen. In addition to the encoding from Section 3.3, we have also implemented the modifications described in Sections 3.4 and 3.5. While unnecessary state pairs are always omitted, either counter-based or unrolling-based predecessor computation can be chosen using a command-line switch.

4.3 Evaluation

We have evaluated the running time of our prototype on small instances of the abovedescribed scenario. Like Dimitrova, Finkbeiner, and Torfah [DFT20], we always used the parameter $\varepsilon = 0.25$. The experiments were conducted on a machine with an AMD Ryzen 3 3200G processor and 16 GB of memory running Ubuntu 20.04 and Z3 version 4.8.7.

Table 4.1 shows the measured running times in seconds. Instances of the scenario are labeled with their parameters using the format $d_1{\times}d_2/u_1/u_2$. Each instance has been evaluated using four different modes of operation, namely all combinations of: checking for the existence of adequate schedulers (chk) versus synthesizing optimal schedulers (opt); and counter-based (count) versus unrolling-based (unroll) predecessor computation. The timeout was at least 2 hours for each individual run.

It is obvious that the running time increases dramatically with increasing size of both the MDP and the schedulers. Since even for an MDP with 25 states, a property with only two scheduler quantifiers, and memoryless schedulers (instance $4{\times}4/1/1$), the running time already amounts to more than 30 minutes in any mode of operation, we assume that our algorithm is far too slow for any real-world use case. The results also suggest that synthesizing optimal schedulers is harder than synthesizing adequate schedulers. Concerning counter-based versus unrolling-based predecessor computation, no clear trend is visible.

5 Related Work

The concept of temporal logic was first introduced into formal verification of computing systems in 1977 by Pnueli [Pnu77]. He proposed a linear-time temporal logic that augments propositional logic with two modal operators, which correspond to the $\Diamond$ and $\Box$ operators in LTL. The standard temporal logic LTL soon arose when Manna and Pnueli [MP81] extended the logic from [Pnu77] with the $\bigcirc$ and $\mathcal{U}$ operators. Clarke and Emerson [CE81] proposed CTL, a branching-time temporal logic that prefixes each temporal operator with a quantifier over paths through a computation tree. In 1983, Emerson and Halpern [EH86] generalized CTL to CTL* by relaxing the one-to-one relationship between temporal operators and path quantifiers. In CTL*, temporal operators are only required to be in the scope of some path quantifier. Thus, CTL* subsumes both CTL and LTL.

The automata-theoretic approach to LTL model checking makes use of the fact that for every LTL formula φ, there exists an ω-automaton that accepts exactly those traces that satisfy φ [VW94]. Simple automata-based LTL model checking algorithms for Kripke structures involve constructing (nondeterministic) Büchi automata from LTL formulas [BK08; Var95]. However, it is also possible to translate these Büchi automata into deterministic Rabin automata [Rog01]. This way, the automata-theoretic approach can also be used for quantitative analysis of DTMCs against properties specified in LTL [BK08]. The probabilistic model checker PRISM [KNP11] is based on this approach. In this thesis, we use the basic idea behind this approach, but modify it slightly such that it can be more efficiently encoded in SMT.

The study of hyperproperties was initiated in 2010 by Clarkson and Schneider [CS10]. In 2014, Clarkson et al. [Cla+14] introduced two new temporal logics specifically for specifying hyperproperties: HyperLTL, which extends LTL with quantification over traces; and HyperCTL*, an extension of CTL* that allows simultaneous quantification over multiple paths.

The HyperLTL synthesis problem was first studied by Finkbeiner et al. [Fin+20]. While it is generally undecidable whether there exists a system that satisfies a given HyperLTL formula, they were able to develop a semi-decision procedure for the universal fragment of HyperLTL. This semi-decision procedure consists of two SMT-based algorithms: a bounded synthesis algorithm; and an algorithm for finding bounded counterexamples. Both algorithms have been implemented in the tool BoSyHyper. The bounded synthesis algorithm from [Fin+20] forms the basis for the HyperLTL synthesis part of the SMT encoding presented in this thesis.

In 1989, Hansson and Jonsson [HJ94] introduced PCTL (Probabilistic Computation Tree Logic), a probabilistic branching-time logic for DTMCs. PCTL can be viewed as a modification of CTL, where path quantification has been replaced with a probabilistic operator that takes a path formula as argument. In PCTL, one can thus specify that a path formula must hold with a given probability, while CTL can only express that a path formula must hold either on all paths that start in the current state, or on at least one of them.

In 2018, Ábrahám and Bonakdarpour [ÁB18] proposed HyperPCTL, which extends PCTL with explicit and simultaneous quantification over multiple states. HyperPCTL can thus express probabilistic hyperproperties of DTMCs.

In 2020, Ábrahám et al. [Ábr+20] modified HyperPCTL in order to make it suitable for specifying probabilistic hyperproperties of MDPs. Their modifications consist in adding quantification over schedulers and making every state quantifier explicitly bind its state variable to a scheduler variable. Ábrahám et al. proved that the HyperPCTL model checking problem for MDPs is generally undecidable, but that it becomes decidable when schedulers are required to be deterministic and memoryless. They developed and implemented an SMT-based algorithm for the resulting restricted model checking problem. In contrast to our approach, this algorithm uses structural recursion over HyperPCTL formulas to construct an SMT constraint system. This is possible since the domain of the probabilistic operator in HyperPCTL is restricted to path forumlas. The PHL probabilistic operator, on the other hand, takes full LTL formulas as argument, and we cannot conceive a way of adapting the recursive SMT encoding procedure from [Ábr+20] to this setting. Therefore, we base the probabilistic part of our approach on a classical probabilistic LTL model checking algorithm instead.

Also in 2020, Dimitrova, Finkbeiner, and Torfah [DFT20] independently and concurrently proposed PHL for specifying probabilistic hyperproperties of MDPs. They proved that model checking of PHL formulas, like HyperPCTL model checking, is generally undecidable. However, in contrast to Ábrahám et al. [Ábr+20], they did not restrict the problem to deterministic, memoryless schedulers in order to achieve decidability. Instead, they focused on a fragment of PHL and developed two model checking algorithms for this fragment. The algorithm presented in this thesis constitutes an attempt to improve one of these model checking algorithms.

6 Conclusion

This thesis presented a new, fully SMT-based approach to bounded model checking for a fragment of PHL. Formulas from this fragment combine a HyperLTL subformula with a linear constraint on the probabilities with which LTL subformulas are satisfied. So far, the only existing bounded model checking algorithm for this fragment of PHL used an SMT-based HyperLTL synthesis tool and a BDD-based probabilistic model checker in a guess-and-check loop. Our approach eliminates this loop by integrating both HyperLTL synthesis and probabilistic LTL model checking into a single SMT constraint system.

The main contribution of this thesis and key to our new bounded model checking algorithm is a general construction for encoding probabilistic LTL model checking in SMT. The key challenge here was to symbolically compute (an approximation of) the accepting BSCCs of the product of a DTMC and a Rabin automaton, where the transition probabilities of the DTMC are not known upfront. We identified a suitable approximation of the accepting BSCCs that can be determined by performing only two fixpoint computations, which are straightforward to encode in SMT.

First experiments with a proof-of-concept implementation for a simple case study showed that, for small MDPs and simple PHL formulas, our approach can work in practice. However, further optimization will be necessary in order to make it scale up to real-world use cases.

To this end, it would be interesting to conduct experiments with other case studies in order to find out how exactly the size of the MDP under consideration and/or the complexity of the PHL formula affect the running time of our algorithm. Unfortunately, this is not possible with our proof-of-concept implementation, since it has one specific case study hard-coded into it. A natural next step would therefore be to build a general implementation to facilitate such experiments.

A BSCC Approximation

Theorem 1. *Let $M = (S, \mathrm{P}, \iota, \mathsf{AP}, L)$ be a DTMC and $\mathcal{R} = (R, 2^{\mathsf{AP}}, \delta, r_0, Acc)$ be a DRA with $Acc = \big(B_j, G_j\big)_{j=1}^{m}$. Then for all $1 \leq j \leq m$ the set*

$$A_j = \big(S \times R\big) \setminus Pre^*\Big(\big(S \times R\big) \setminus \Big(Pre^*\big(S \times G_j\big) \setminus \big(S \times B_j\big)\Big)\Big)$$

is a superset of the union of the accepting BSCCs of $M \otimes \mathcal{R}$ for $\big(B_j, G_j\big)$, and from all $t \in A_j$ an accepting BSCC will be reached almost surely.

Proof. Let $1 \leq j \leq m$. We first introduce the notation

$$A'_j = Pre^*\big(S \times G_j\big) \setminus \big(S \times B_j\big)$$

so that

$$A_j = \big(S \times R\big) \setminus Pre^*\Big(\big(S \times R\big) \setminus A'_j\Big).$$

Note that it holds that $A_j \subseteq A'_j$.

Let now $t \in A_j$. We begin by showing that

i) $Post^*(t) \cap \big(S \times G_j\big) \neq \emptyset$,

ii) $t \notin S \times B_j$, and

iii) $Post^*(t) \subseteq A_j$.

We can see i) as follows. We know that $t \in A_j \subseteq A'_j \subseteq Pre^*(S \times G_j)$. It follows that $Post^*(t) \cap (S \times G_j) \neq \emptyset$.

We can see ii) as follows. We know that $t \in A_j \subseteq A'_j$. By the definition of A'_j it follows that $t \notin S \times B_j$.

We can see iii) as follows. Assume that $Post^*(t) \not\subseteq A_j$. Then there exists a state $t' \in (S \times R) \setminus A_j$ with $t' \in Post^*(t)$. We know that $t' \in (S \times R) \setminus A_j = Pre^*\big((S \times R) \setminus A'_j\big)$. It follows that $t \in Pre^*(t') \subseteq Pre^*\big((S \times R) \setminus A'_j\big) = (S \times R) \setminus A_j$ in contradiction to the assumption $t \in A_j$.

Now we show that

iv) every BSCC B of $M \otimes \mathcal{R}$ with $B \subseteq A_j$ is accepting for (B_j, G_j), and

v) for every accepting BSCC B of $M \otimes \mathcal{R}$ for (B_j, G_j) we have $B \subseteq A_j$.

We begin by proving iv). Let $B \subseteq A_j$ be a BSCC of $M \otimes \mathcal{R}$. By ii) we know that $B \cap (S \times B_j) = \emptyset$. Since B is nonempty, there exists a state $t \in B$. By i) it follows that $B \cap (S \times G_j) = Post^*(t) \cap (S \times G_j) \neq \emptyset$. Thus, B is accepting for (B_j, G_j).

Now we prove v). Let B be an accepting BSCC of $M \otimes \mathcal{R}$ for (B_j, G_j). We first show that $B \subseteq A'_j$. Let $t \in B$. We have to show that $t \in A'_j$.

Since B is an accepting for (B_j, G_j), we know that $t \notin S \times B_j$ and that there exists a state $t' \in B$ with $t' \in S \times G_j$. Since B is a BSCC, it also holds that $t \in \mathit{Pre}^*(t') \subseteq \mathit{Pre}^*(S \times G_j)$. It follows that $t \in A'_j$.

Now we show that $B \subseteq A_j$. Let again $t \in B$. Assume that $t \notin A_j$. Then $t \in (S \times R) \setminus A_j = \mathit{Pre}^*\big((S \times R) \setminus A'_j\big)$. Thus, there exists a state $t'' \in (S \times R) \setminus A'_j$ such that $t \in \mathit{Pre}^*(t'')$. Since B is a BSCC, it holds that $\mathit{Post}^*(t) = B$. It follows that $t'' \in \mathit{Post}^*(t) = B \subseteq A'_j$ in contradiction to $t'' \in (S \times R) \setminus A'_j$. Thus we have $t \in A_j$.

By v) we know that A_j is a superset of the union of the accepting BSCCs of $M \otimes \mathcal{R}$ for (B_j, G_j). It remains to be shown that from each state $t \in A_j$ an accepting BSCC for (B_j, G_j) will be reached almost surely. Let $M \otimes \mathcal{R} = (S \times R, \widehat{\mathrm{P}}, \widehat{\iota}, \mathsf{AP}, \widehat{L})$ denote the product DTMC of M and $\mathcal{R}$. By iii) we know that $A = (A_j, \mathrm{P}', \iota', \mathsf{AP}, L')$ with

- $\mathrm{P}' = \widehat{\mathrm{P}}|_{A_j \times A_j}$,
- $\iota'(t) = \frac{1}{|A_j|}$ for all $t \in A_j$, and
- $L' = \widehat{L}|_{A_j}$

forms a sub-DTMC of $M \otimes \mathcal{R}$. We know that from every state of a DTMC, a BSCC will be reached almost surely. Thus, from every state $t \in A_j$ of A, a BSCC of A will be reached almost surely. By iv) every BSCC of A is an accepting BSCC of $M \otimes \mathcal{R}$ for (B_j, G_j). $\square$

Bibliography

[ÁB18] Erika Ábrahám and Borzoo Bonakdarpour. "HyperPCTL: A Temporal Logic for Probabilistic Hyperproperties". In: *QEST*. Vol. 11024. Lecture Notes in Computer Science. Springer, 2018, pp. 20–35.

[ABL13] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. "SAT-based MaxSAT algorithms". In: *Artif. Intell.* 196 (2013), pp. 77–105.

[Ábr+20] Erika Ábrahám, Ezio Bartocci, Borzoo Bonakdarpour, and Oyendrila Dobe. "Probabilistic Hyperproperties with Nondeterminism". In: *ATVA*. Vol. 12302. Lecture Notes in Computer Science. Springer, 2020, pp. 518–534.

[Bar+09] Clark W. Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. "Satisfiability Modulo Theories". In: *Handbook of Satisfiability.* IOS Press, 2009, pp. 825–885.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of model checking.* MIT Press, 2008.

[BP14] Nikolaj Bjørner and Anh-Dung Phan. "νZ - Maximal Satisfaction with Z3". In: *SCSS*. Vol. 30. EPiC Series in Computing. EasyChair, 2014, pp. 1–9.

[BT18] Clark W. Barrett and Cesare Tinelli. "Satisfiability Modulo Theories". In: *Handbook of Model Checking.* Springer, 2018, pp. 305–343.

[CE81] Edmund M. Clarke and E. Allen Emerson. "Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic". In: *Logic of Programs.* Vol. 131. Lecture Notes in Computer Science. Springer, 1981, pp. 52–71.

[Cim+13] Alessandro Cimatti, Alberto Griggio, Bastiaan Joost Schaafsma, and Roberto Sebastiani. "The MathSAT5 SMT Solver". In: *TACAS*. Vol. 7795. Lecture Notes in Computer Science. Springer, 2013, pp. 93–107.

[Cla+14] Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, and César Sánchez. "Temporal Logics for Hyperproperties". In: *POST*. Vol. 8414. Lecture Notes in Computer Science. Springer, 2014, pp. 265–284.

[CS10] Michael R. Clarkson and Fred B. Schneider. "Hyperproperties". In: *J. Comput. Secur.* 18.6 (2010), pp. 1157–1210.

[DFT20] Rayna Dimitrova, Bernd Finkbeiner, and Hazem Torfah. "Probabilistic Hyperproperties of Markov Decision Processes". In: *ATVA*. Vol. 12302. Lecture Notes in Computer Science. Springer, 2020, pp. 484–500.

[Dwo11] Cynthia Dwork. "Differential Privacy". In: *Encyclopedia of Cryptography and Security (2nd Ed.)* Springer, 2011, pp. 338–340.

[EH86] E. Allen Emerson and Joseph Y. Halpern. “”Sometimes” and ”Not Never” revisited: on branching versus linear time temporal logic”. In: *J. ACM* 33.1 (1986), pp. 151–178.

[Fin+20] Bernd Finkbeiner, Christopher Hahn, Philip Lukert, Marvin Stenger, and Leander Tentrup. “Synthesis from hyperproperties”. In: *Acta Informatica* 57.1-2 (2020), pp. 137–163.

[HJ94] Hans Hansson and Bengt Jonsson. “A Logic for Reasoning about Time and Reliability”. In: *Formal Aspects Comput.* 6.5 (1994), pp. 512–535.

[KNP11] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. “PRISM 4.0: Verification of Probabilistic Real-Time Systems”. In: *CAV*. Vol. 6806. Lecture Notes in Computer Science. Springer, 2011, pp. 585–591.

[MB08] Leonardo Mendonça de Moura and Nikolaj Bjørner. “Z3: An Efficient SMT Solver”. In: *TACAS*. Vol. 4963. Lecture Notes in Computer Science. Springer, 2008, pp. 337–340.

[MM18] João Marques-Silva and Sharad Malik. “Propositional SAT Solving”. In: *Handbook of Model Checking*. Springer, 2018, pp. 247–275.

[MP81] Zohar Manna and Amir Pnueli. *Verification of Concurrent Programs, Part I: The Temporal Framework*. 1981.

[Pnu77] Amir Pnueli. “The Temporal Logic of Programs”. In: *FOCS*. IEEE Computer Society, 1977, pp. 46–57.

[Rog01] Markus Roggenbach. “Determinization of Büchi-Automata”. In: *Automata, Logics, and Infinite Games*. Vol. 2500. Lecture Notes in Computer Science. Springer, 2001, pp. 43–60.

[ST20] Roberto Sebastiani and Patrick Trentin. “OptiMathSAT: A Tool for Optimization Modulo Theories”. In: *J. Autom. Reason.* 64.3 (2020), pp. 423–460.

[Tom14] Silvia Tomasi. “Optimization Modulo Theories with Linear Rational Costs”. PhD thesis. University of Trento, Italy, 2014.

[Var95] Moshe Y. Vardi. “An Automata-Theoretic Approach to Linear Temporal Logic”. In: *Banff Higher Order Workshop*. Vol. 1043. Lecture Notes in Computer Science. Springer, 1995, pp. 238–266.

[Ven11] Roc Oliver Vendrell. “Optimization Modulo Theories”. MA thesis. Universitat Politècnica de Catalunya, 2011.

[VW94] Moshe Y. Vardi and Pierre Wolper. “Reasoning About Infinite Computations”. In: *Inf. Comput.* 115.1 (1994), pp. 1–37.

[Web+19] Tjark Weber, Sylvain Conchon, David Déharbe, Matthias Heizmann, Aina Niemetz, and Giles Reger. “The SMT Competition 2015-2018”. In: *J. Satisf. Boolean Model. Comput.* 11.1 (2019), pp. 221–259.