The present work was submitted to the LuFG Theory of Hybrid Systems

BACHELOR OF SCIENCE THESIS

PROBABILISTIC HYPERPROPERTIES WITH REWARDS

Lukas Wilke

Examiners:
Prof. Dr. Erika Ábrahám, RWTH Aachen
Prof. Dr. Ir. Dr. h. c. Joost-Pieter Katoen, RWTH Aachen

Additional Advisors:
Ezio Bartocci, TU Wien
Borzoo Bonakdarpour, Michigan State University
Oyendrila Dobe, Michigan State University

Aachen, August 13, 2021

Abstract

HyperPCTL is a temporal logic designed to reason about hyperproperties of probabilistic systems. It has been introduced for both DTMCs as well as MDPs, which allow nondeterminism. Essentially, HyperPCTL allows arguing about multiple executions of a system at the same time. In this thesis, we discuss hyperproperties with rewards. We add state rewards to DTMCs and MDPs and extend the syntax and semantics of HyperPCTL to allow reasoning about the rewards of paths in a model. Furthermore, we modify an existing model checking algorithm for HyperPCTL to work with the modified logic. Finally, we also present two case studies as possible applications for HyperPCTL with rewards.

Contents

1 Introduction **7**

2 Preliminaries **9**
2.1 Discrete-time Markov Chains 9
2.2 Markov Decision Processes 10
2.3 HyperPCTL 12
2.4 HyperPCTL Model Checking 14

3 HyperPCTL with Rewards **21**
3.1 MDPs with Rewards 21
3.2 Syntax & Semantics 22
3.3 Model Checking Algorithm 24

4 Case Studies **35**
4.1 Timing Attack 35
4.2 Probabilistic Conformance 36
4.3 Implementation 37

5 Conclusion **39**
5.1 Summary 39
5.2 Discussion 39
5.3 Future work 40

Bibliography **41**

Chapter 1

Introduction

In computer science, we are often interested in certain properties of computer systems. Since many of these systems include elements of randomness, we are also particularly interested in properties of *probabilistic systems*. For the modeling and verification of such properties, logics such as PCTL [HJ95] have been introduced. This allows us to reason about the probability that an execution of a system will fulfill certain conditions. However, not all interesting characteristics of probabilistic systems can be examined this way. This leads us to the notion of *hyperproperties* [CS08]. Hyperproperties describe sets of properties. In contrast to standard temporal properties, hyperproperties can describe multiple executions of a system at the same time. They allow us, for instance, to describe whether any given pair of executions of a system has the same probabilities of satisfying some requirement, whereas regular properties only allow us to consider executions individually. This notion of hyperproperties allows us to examine some interesting characteristics that could not be expressed as properties, such as probabilistic noninterference [GM82].

In [ÁB18], the temporal logic HyperPCTL as an extension of PCTL was proposed for Discrete-Time Markov Chains to reason about hyperproperties of probabilistic systems. In [ÁBBD20], this logic was extended to Markov Decision Processes to allow for nondeterminism. In this thesis, we would like to further extend HyperPCTL to include rewards. Rewards allow us to easily model interesting aspects like execution time or energy consumption, so adding rewards to HyperPCTL would allow us to argue about these aspects in combination with hyperproperties in probabilistic systems.

In Chapter 2, we will introduce preliminary concepts necessary for understanding this thesis. This includes DTMCs and MDPs as models for probabilistic systems as well as the logic HyperPCTL, which can be used to reason about hyperproperties of these models. In Chapter 3, we will extend these models and HyperPCTL to reason about rewards. In Chapter 4, we will present two case studies that will serve as examples for possible applications of HyperPCTL with rewards. Finally, in Chapter 5, we will draw a conclusion and present possibilities for future work.

Chapter 2

Preliminaries

This chapter introduces preliminary concepts adapted from [ÁBBD20] that are necessary to understand this thesis. First, we will present DTMCs and MDPs as models for probabilistic systems. After that, we will introduce the temporal logic HyperPCTL, which extends the well-known logic PCTL (Probabilistic Computational Tree Logic) to allow the examination of hyperproperties. Finally, we will present a model checking algorithm for HyperPCTL formulas.

In this thesis, we will write $\mathbb{R}$ for the real numbers and $\mathbb{N}$ for the natural numbers. We consider 0 to be a natural number. If x is a vector, we will write x_i for the ith element of the vector.

2.1 Discrete-time Markov Chains

Definition 2.1.1. *A* discrete-time Markov chain (DTMC) *is a tuple* $\mathcal{D}=(S, \mathbf{P}, \mathsf{AP}, L)$ *with the following components:*

- S *is a nonempty finite set of* states*;*
- $\mathbf{P} : S \times S \to [0,1] \subseteq \mathbb{R}$ *is a* transition probability function *with* $\sum_{s' \in S} \mathbf{P}(s,s') = 1$ *for all* $s \in S$*;*
- AP *is a finite set of* atomic propositions *and*
- $L : S \to 2^{\mathsf{AP}}$ *is a* labeling function.

A *path* of $\mathcal{D}$ is a sequence $\pi = (s_0,s_1,s_2,...)$ of states, where $s_i \in S$ and $\mathbf{P}(s_i,s_{i+1}) > 0$ for all $i \in \mathbb{N}$. A *finite path* is a non-empty prefix of a path. We use $Paths_s^{\mathcal{D}}$ to denote the set of all infinite paths starting in the state $s \in S$ and $fPaths_s^{\mathcal{D}}$ to denote the set of all finite paths starting in s. For a path $\pi = (s_0,s_1,s_2,\ldots)$, we denote the state s_i reached after i steps as π_i for all $i \in \mathbb{N}$. We define the length of a finite path $\pi = (s_0,s_1,\ldots,s_n)$ as $|\pi| = n$. We define the probability of a finite path $\pi = (s_0,s_1,\ldots,s_n)$ as $\mathbf{P}(\pi) = \prod_{i=0}^{n-1} \mathbf{P}(s_i,s_{i+1})$. The probability of a set $R \subseteq fPaths_s^{\mathcal{D}}$ of finite paths starting in a state $s \in S$ is defined as $\Pr^{\mathcal{D}}(R) = \sum_{\pi \in R'} \mathbf{P}(\pi)$ with $R' = \{\pi \in R \mid \text{no } \pi' \in R \setminus \{\pi\} \text{ is a prefix of } \pi\}$. For a finite path $\pi \in fPaths_s^{\mathcal{D}}$, its *cylinder set* $Cyl^{\mathcal{D}}(\pi)$ is the set of all infinite paths of $\mathcal{D}$ with π as a prefix. The probability of a union of cylinder sets $R = \cup_{\pi \in R'} Cyl^{\mathcal{D}}(\pi) \subseteq Paths_s^{\mathcal{D}}$ with $R' \subseteq fPaths_s^{\mathcal{D}}$ is defined as $\Pr^{\mathcal{D}}(R) = \Pr^{\mathcal{D}}(R')$.

Definition 2.1.2. *The* parallel composition *of two DTMCs* $\mathcal{D}_1 = (S_1, \mathbf{P}_1, \mathsf{AP}_1, L_1)$ *and* $\mathcal{D}_2 = (S_2, \mathbf{P}_2, \mathsf{AP}_2, L_2)$ *with* $\mathsf{AP}_1 \cap \mathsf{AP}_2 = \emptyset$ *is defined as the DTMC* $\mathcal{D}_1 \| \mathcal{D}_2 = (S, \mathbf{P}, \mathsf{AP}, L)$*, where*

- $S = S_1 \times S_2$*;*
- $\mathbf{P}(s, s') = \mathbf{P}_1(s_1,s_1') \cdot \mathbf{P}_2(s_2,s_2')$ *for all* $s = (s_1,s_2) \in S$ *and* $s' = (s_1',s_2') \in S$*;*
- $\mathsf{AP} = \mathsf{AP}_1 \cup \mathsf{AP}_2$ *and*
- $L(s) = L_1(s_1) \cup L_2(s_2)$ *for all* $s = (s_1,s_2) \in S$*.*

We require AP_1 and AP_2 to be disjoint so that it will always be clear which of the two DTMCs the atomic propositions in any state of the parallel composition originate from. As an example, Figure 2.1 shows two DTMCs and their parallel composition. The figure also serves as an example of how DTMCs are commonly depicted graphically.

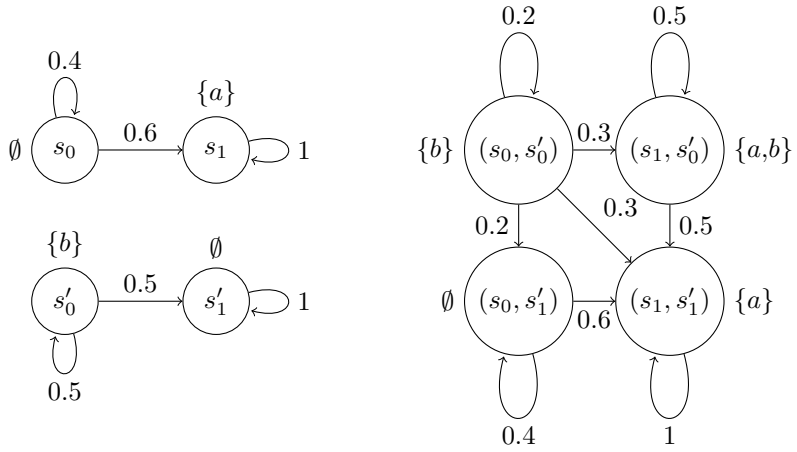

Figure 2.1: Two DTMCs and their parallel composition.

2.2 Markov Decision Processes

Definition 2.2.1. *A* Markov Decision Process (MDP) *is a tuple* $\mathcal{M} = (S, Act, \mathbf{P}, \mathsf{AP}, L)$ *with the following components:*

- S *is a nonempty finite set of* states*;*
- Act *is a nonempty finite set of* actions*;*
- $\mathbf{P} : S \times Act \times S \to [0,1] \subseteq \mathbb{R}$ *is a* transition probability function *such that for every state* $s \in S$ *there is a nonempty set of enabled actions* $Act(s) \subseteq Act$ *such that* $\sum_{s' \in S} \mathbf{P}(s, \alpha, s') = 1$ *for all* $\alpha \in Act(s)$ *and* $\sum_{s' \in S} \mathbf{P}(s, \alpha, s') = 0$ *for all* $\alpha \in Act \setminus Act(s)$*;*
- AP *is a finite set of* atomic propositions *and*

- $L : S \to 2^{\mathsf{AP}}$ *is a* labeling function.

MDPs differ from DTMCs through the introduction of nondeterminism enabled by the actions. In every state, there is a choice of one or more actions which determine the probabilities of transitioning to other states. This nondeterminism can be removed through the use of *schedulers*.

Definition 2.2.2. *A* scheduler *for an MDP* $\mathcal{M} = (S, Act, \mathbf{P}, \mathsf{AP}, L)$ *is a tuple* $\sigma = (Q, act, mode, init)$ *with the following components:*

- Q *is a countable set of modes;*
- $act : Q \times S \times Act \to [0,1] \subseteq \mathbb{R}$ *is a function such that* $\sum_{\alpha \in Act(s)} act(q,s,\alpha) = 1$ *for every* $s \in S$ *and* $q \in Q$ *and* $act(q,s,\alpha) = 0$ *for all* $\alpha \in Act \setminus Act(s)$ *for every* $q \in Q, s \in S$*;*
- $mode : Q \times S \to Q$ *is a mode transition function and*
- $init : S \to Q$ *assigns each state of* $\mathcal{M}$ *a starting mode.*

The set of all schedulers for the MDP $\mathcal{M}$ is $\Sigma^{\mathcal{M}}$. A scheduler is *finite-memory* if Q is finite, *memoryless* if $|Q| = 1$ and *non-probabilistic* if $act(q,s,\alpha) \in \{0,1\}$ for all $q \in Q, s \in S, \alpha \in Act$. A non-probabilistic, memoryless scheduler therefore essentially just assigns one action to every MDP state. Since schedulers remove nondeterminism, every scheduler of an MDP induces a DTMC.

Definition 2.2.3. *Assume an MDP* $\mathcal{M} = (S, Act, \mathbf{P}, \mathsf{AP}, L)$ *and a scheduler* $\sigma = (Q, act, mode, init)$ *for* $\mathcal{M}$*. The DTMC induced by* $\mathcal{M}$ *and* σ *is defined as* $\mathcal{M}^\sigma = (S^\sigma, \mathbf{P}^\sigma, \mathsf{AP}, L^\sigma)$ *with:*

- $S^\sigma = Q \times S$*;*
- $\mathbf{P}^\sigma((q,s),(q',s')) = \begin{cases} \sum_{\alpha \in Act(s)} act(q,s,\alpha) \cdot \mathbf{P}(s,\alpha,s') & \text{if } q' = mode(q,s) \\ 0 & \text{if } q' \neq mode(q,s) \end{cases}$ *and*
- $L^\sigma(q,s) = L(s)$ *for all* $q \in Q$ *and* $s \in S$*.*

If σ is memoryless, we will sometimes omit the mode in the states of $\mathcal{M}^\sigma$ and write s for $(q,s) \in S^\sigma$ with $q \in Q, s \in S$. As an example, Figure 2.2 depicts an MDP as well as its induced DTMC for a scheduler σ that always chooses the action α in s_0.

Definition 2.2.4. *The* n-ary self composition *of an MDP* $\mathcal{M} = (S, Act, \mathbf{P}, \mathsf{AP}, L)$ *for a sequence* $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_n) \in (\Sigma^{\mathcal{M}})^n$ *of schedulers for* $\mathcal{M}$ *is the DTMC parallel composition* $\mathcal{M}^{\boldsymbol{\sigma}} = (S^{\boldsymbol{\sigma}}, \mathbf{P}^{\boldsymbol{\sigma}}, \mathsf{AP}^{\boldsymbol{\sigma}}, L^{\boldsymbol{\sigma}}) = \mathcal{M}_1^{\sigma_1} \| \ldots \| \mathcal{M}_n^{\sigma_n}$*, where* $\mathcal{M}_i^{\sigma_i}$ *is the DTMC induced by* $\mathcal{M}_i$ *and* σ_i*, with* $\mathcal{M}_i = (S, Act, \mathbf{P}, \mathsf{AP}_i, L_i)$ *with* $\mathsf{AP}_i = \{a_i \mid a \in \mathsf{AP}\}$ *and* $L_i(s) = \{a_i \mid a \in L(s)\}$ *for all* $s \in S$*.*

It would be possible to allow the composition of multiple different MDPs, but this would make the semantics of HyperPCTL more complicated, which is why we restrict ourselves to the self-composition. We also note that this is the parallel composition of the DTMCs induced by the MDPs. The parallel composition of the MDPs themselves would also be possible to define, but is not of interest to us.

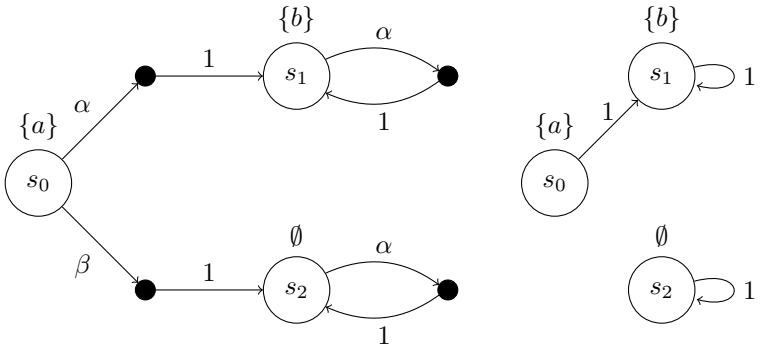

Figure 2.2: An MDP and one of its induced DTMCs.

2.3 HyperPCTL

HyperPCTL is an extension of PCTL. Compared to PCTL, HyperPCTL allows arithmetic operations with probabilities and comparing probabilities to each other. Through the use of state quantifiers, it is possible to consider multiple executions of a model at the same time and thus examine hyperproperties of models. It is also possible to compare experiments with different schedulers through the use of scheduler quantifiers. These extensions give HyperPCTL more expressivity than PCTL.

Definition 2.3.1. HyperPCTL (quantified) state formulas φ^q *for MDPs are inductively defined as follows:*

quantified formula	φ^q	::=	$\forall\hat{\sigma}.\varphi^q \mid \exists\hat{\sigma}.\varphi^q \mid \forall\hat{s}(\hat{\sigma}).\varphi^q \mid \exists\hat{s}(\hat{\sigma}).\varphi^q \mid \varphi^{nq}$
non-quantified formula	φ^{nq}	::=	$\texttt{true} \mid a_{\hat{s}} \mid \varphi^{nq} \wedge \varphi^{nq} \mid \neg\varphi^{nq} \mid \varphi^{ar} < \varphi^{ar}$
arithmetic expression	φ^{ar}	::=	$\mathbb{P}(\varphi^{path}) \mid f(\varphi_1^{ar},\ldots,\varphi_k^{ar})$
path formula	φ^{path}	::=	$\bigcirc\varphi^{nq} \mid \varphi^{nq}\,\mathcal{U}\,\varphi^{nq} \mid \varphi^{nq}\,\mathcal{U}^{[k_1,k_2]}\,\varphi^{nq}$

where $\hat{\sigma}$ *is a scheduler variable from an infinite set* $\hat{\Sigma}$, $\hat{s}$ *is a state variable from an infinite set* $\hat{S}$, $a \in \mathsf{AP}$ *is an atomic proposition,* $f : [0,1]^k \to \mathbb{R}$ *are* k*-ary arithmetic operators (addition, subtraction, multiplication), where constants are 0-ary functions, and* k_1,$k_2 \in \mathbb{N}$ *are nonnegative integers such that* $k_1 \leq k_2$.

We will sometimes just write φ for any formula when the type of formula is clear from context. We use $\hat{s}$ and $\hat{\sigma}$ for state and scheduler variables, while s and σ are used for states and schedulers. A HyperPCTL formula is *well-formed* if every occurrence of an $a_{\hat{s}}$ with $a \in \mathsf{AP}, \hat{s} \in \hat{S}$ is in the scope of a state quantifier for $\hat{s}(\hat{\sigma})$ for a $\hat{\sigma} \in \hat{\Sigma}$ and every state quantifier for $\hat{s}(\hat{\sigma})$ is in the scope of a scheduler quantifier for $\hat{\sigma}$.

We allow the following standard syntactic sugar:

$\texttt{false}$	=	$\neg\texttt{true}$
$\varphi_1^{nq} \vee \varphi_2^{nq}$	=	$\neg(\neg\varphi_1^{nq} \wedge \neg\varphi_2^{nq})$
$\varphi_1^{nq} \to \varphi_2^{nq}$	=	$\neg\varphi_1^{nq} \vee \varphi_2^{nq}$
$\varphi_1^{ar} > \varphi_2^{ar}$	=	$\varphi_2^{ar} < \varphi_1^{ar}$
$\varphi_1^{ar} \leq \varphi_2^{ar}$	=	$\neg(\varphi_1^{ar} > \varphi_2^{ar})$
$\varphi_1^{ar} \geq \varphi_2^{ar}$	=	$\neg(\varphi_1^{ar} < \varphi_2^{ar})$
$\varphi_1^{ar} = \varphi_2^{ar}$	=	$(\varphi_1^{ar} \leq \varphi_2^{ar}) \wedge (\varphi_1^{ar} \geq \varphi_2^{ar})$
$\varphi_1^{ar} \neq \varphi_2^{ar}$	=	$\neg(\varphi_1^{ar} = \varphi_2^{ar})$
$\Diamond\varphi^{nq}$	=	$\texttt{true}\,\mathcal{U}\,\varphi^{nq}$
$\mathbb{P}(\Box\varphi^{nq})$	=	$1 - \mathbb{P}(\Diamond\neg\varphi^{nq})$

HyperPCTL state formulas are evaluated in the context of

- an MDP $\mathcal{M}$,
- a sequence $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_n)$ of schedulers
- and a sequence $\boldsymbol{r} = ((q_1,s_1), \ldots, (q_n,s_n)) \in S^{\boldsymbol{\sigma}}$ of states.

We let () be the empty sequence and use $\circ$ for concatenation. The satisfaction of a HyperPCTL quantified formula by an MDP $\mathcal{M}$ is defined as follows:

$$\mathcal{M} \models \varphi \qquad \textit{iff} \qquad \mathcal{M},(),() \models \varphi \ .$$

Formulas are evaluated by structural recursion. Formally, the semantics judgment rules are as follows, where $\boldsymbol{\sigma}$ and $\boldsymbol{r}$ are defined as above and n is their length:

$$\begin{array}{lll}
\mathcal{M},\boldsymbol{\sigma},\boldsymbol{r} \models \texttt{true} & & \\
\mathcal{M},\boldsymbol{\sigma},\boldsymbol{r} \models a_i & \textit{iff} & a_i \in L^{\boldsymbol{\sigma}}(\boldsymbol{r}) \\
\mathcal{M},\boldsymbol{\sigma},\boldsymbol{r} \models \varphi_1^{nq} \wedge \varphi_2^{nq} & \textit{iff} & \mathcal{M},\boldsymbol{\sigma},\boldsymbol{r} \models \varphi_1^{nq} \textit{ and } \mathcal{M},\boldsymbol{\sigma},\boldsymbol{r} \models \varphi_2^{nq} \\
\mathcal{M},\boldsymbol{\sigma},\boldsymbol{r} \models \neg\varphi^{nq} & \textit{iff} & \mathcal{M},\boldsymbol{\sigma},\boldsymbol{r} \not\models \varphi^{nq} \\
\mathcal{M},\boldsymbol{\sigma},\boldsymbol{r} \models \forall\hat{\sigma}.\varphi^q & \textit{iff} & \forall\sigma \in \Sigma^{\mathcal{M}}.\ \mathcal{M},\boldsymbol{\sigma},\boldsymbol{r} \models \varphi^q[\hat{\sigma} \rightsquigarrow \sigma] \\
\mathcal{M},\boldsymbol{\sigma},\boldsymbol{r} \models \exists\hat{\sigma}.\varphi^q & \textit{iff} & \exists\sigma \in \Sigma^{\mathcal{M}}.\ \mathcal{M},\boldsymbol{\sigma},\boldsymbol{r} \models \varphi^q[\hat{\sigma} \rightsquigarrow \sigma] \\
\mathcal{M},\boldsymbol{\sigma},\boldsymbol{r} \models \forall\hat{s}(\sigma).\varphi^q & \textit{iff} & \forall s \in S.\ \mathcal{M},\boldsymbol{\sigma} \circ \sigma,\boldsymbol{r} \circ (\mathit{init}(s), s) \models \varphi^q[\hat{s} \rightsquigarrow n+1] \\
\mathcal{M},\boldsymbol{\sigma},\boldsymbol{r} \models \exists\hat{s}(\sigma).\varphi^q & \textit{iff} & \exists s \in S.\ \mathcal{M},\boldsymbol{\sigma} \circ \sigma,\boldsymbol{r} \circ (\mathit{init}(s), s) \models \varphi^q[\hat{s} \rightsquigarrow n+1] \\
\mathcal{M},\boldsymbol{\sigma},\boldsymbol{r} \models \varphi_1^{ar} < \varphi_2^{ar} & \textit{iff} & [\![\varphi_1^{ar}]\!]_{\mathcal{M},\boldsymbol{\sigma},\boldsymbol{r}} < [\![\varphi_2^{ar}]\!]_{\mathcal{M},\boldsymbol{\sigma},\boldsymbol{r}} \\
[\![\mathbb{P}(\varphi^{path})]\!]_{\mathcal{M},\boldsymbol{\sigma},\boldsymbol{r}} & = & \Pr^{\mathcal{M}^{\boldsymbol{\sigma}}}\big(\{\pi \in \mathit{Paths}_{\boldsymbol{r}}^{\mathcal{M}^{\boldsymbol{\sigma}}} \mid \mathcal{M},\boldsymbol{\sigma},\pi \models \varphi^{path}\}\big) \\
[\![f(\varphi_1^{ar}, \ldots, \varphi_k^{ar})]\!]_{\mathcal{M},\boldsymbol{\sigma},\boldsymbol{r}} & = & f\big([\![\varphi_1^{ar}]\!]_{\mathcal{M},\boldsymbol{\sigma},\boldsymbol{r}}, \ldots,\ [\![\varphi_k^{ar}]\!]_{\mathcal{M},\boldsymbol{\sigma},\boldsymbol{r}}\big)
\end{array}$$

where $\varphi^q[\hat{\sigma} \rightsquigarrow \sigma]$ is the formula obtained by instantiating $\exists\hat{\sigma}.\varphi^q$ or $\forall\hat{\sigma}.\varphi^q$ with a scheduler $\sigma \in \Sigma^{\mathcal{M}}$, which is done by replacing every occurrence of the scheduler variable $\hat{\sigma}$ in any subformula of φ^q with σ.

Likewise, $\varphi^q[\hat{s} \rightsquigarrow n+1]$ is the formula obtained by instantiating $\exists\hat{s}(\sigma).\varphi^q$ or $\forall\hat{s}(\sigma).\varphi^q$ with a state s, which is done by replacing every occurrence of the state variable $\hat{s}$ in any subformula of φ^q with $n+1$.

The satisfaction relation for path formulas is defined as follows, where $\pi = (\boldsymbol{r}_0,\boldsymbol{r}_1,\ldots)$ with $\boldsymbol{r}_i = ((q_{i,1},s_{i,1}), \ldots, (q_{i,n},s_{i,n}))$ is a path of $\mathcal{M}^{\boldsymbol{\sigma}}$:

$$\begin{array}{lll}
\mathcal{M},\boldsymbol{\sigma},\pi \models \bigcirc\varphi & \textit{iff} & \mathcal{M},\boldsymbol{\sigma},\pi_1 \models \varphi \\
\mathcal{M},\boldsymbol{\sigma},\pi \models \varphi_1\,\mathcal{U}\,\varphi_2 & \textit{iff} & \exists j \geq 0.\Big(\mathcal{M},\boldsymbol{\sigma},\pi_j \models \varphi_2 \wedge \forall i \in [0,j).\ \mathcal{M},\boldsymbol{\sigma},\pi_i \models \varphi_1\Big) \\
\mathcal{M},\boldsymbol{\sigma},\pi \models \varphi_1\,\mathcal{U}^{\,[k_1,k_2]}\,\varphi_2 & \textit{iff} & \exists j \in [k_1,k_2].\Big(\mathcal{M},\boldsymbol{\sigma},\pi_j \models \varphi_2 \wedge \forall i \in [0,j).\mathcal{M},\boldsymbol{\sigma},\pi_i \models \varphi_1\Big)
\end{array}$$

As an example, we will evaluate the formula $\varphi = \exists\hat{\sigma}.\exists\hat{\sigma}'.\exists\hat{s}(\hat{\sigma}).\exists\hat{s}'(\hat{\sigma}').\mathbb{P}((a_1 \wedge a_2)\,\mathcal{U}\,(b_1 \wedge \neg b_2)) = 1$ on the MDP presented in Figure 2.2 in the previous section, which we will now call $\mathcal{M}$. We first need to instantiate the scheduler and state variables. We instantiate $\hat{\sigma}$ with a scheduler σ_1 that always chooses action α in s_0, and we instantiate $\hat{\sigma}'$ with a scheduler σ_2 that always chooses β in s_0. We instantiate both $\hat{s}$ and $\hat{s}'$ with s_0. Thus, we now want to verify if $\mathcal{M},(\sigma_1,\sigma_2),(s_0,s_0) \models \mathbb{P}((a_1 \wedge a_2)\,\mathcal{U}\,(b_1 \wedge \neg b_2)) = 1$. We must evaluate $[\![\mathbb{P}((a_1 \wedge a_2)\,\mathcal{U}\,(b_1 \wedge \neg b_2))]\!]_{\mathcal{M},(\sigma_1,\sigma_2),(s_0,s_0)}$. There is only one infinite path in $\mathcal{M}^{(\sigma_1,\sigma_2)}$ that starts in (s_0,s_0) and it begins with $\pi = ((s_0,s_0),(s_1,s_2),\ldots)$. Since s_1 is labeled with b and s_2 is not, $\mathcal{M},(\sigma_1,\sigma_2),(s_1,s_2)$ satisfies $(b_1 \wedge \neg b_2)$. Since s_0 is labeled with a, every state before (s_1,s_2) on the path, which is only (s_0,s_0) in this case, satisfies $(a_1 \wedge a_2)$. Since the probability of this path is 1, $[\![\mathbb{P}((a_1 \wedge a_2)\,\mathcal{U}\,(b_1 \wedge \neg b_2))]\!]_{\mathcal{M},(\sigma_1,\sigma_2),(s_0,s_0)} = 1$. Therefore, $\mathcal{M},(\sigma_1,\sigma_2),(s_0,s_0) \models \mathbb{P}((a_1 \wedge a_2)\,\mathcal{U}\,(b_1 \wedge \neg b_2)) = 1$ and $\mathcal{M},(),() \models \varphi$.

2.4 HyperPCTL Model Checking

PCTL model checking is decidable, but the additional expressivity makes HyperPCTL for MDPs in general undecidable. For PCTL model checking, only non-probabilistic, memoryless schedulers are relevant. This is not the case for HyperPCTL model checking, as the restriction to only those schedulers changes the semantics of HyperPCTL. However, an MDP only has a finite amount of non-probabilistic memoryless schedulers, so if the semantics are restricted to only this kind of scheduler, the model checking problem becomes decidable. The following is an SMT-based algorithm from [ÁBBD20] for HyperPCTL model checking for non-probabilistic schedulers. The algorithm is restricted to formulas with only one scheduler quantifier. The case for formulas with multiple scheduler quantifiers would be similar, but technically more complicated.

Algorithm 1: Main SMT encoding algorithm

Input: $\mathcal{M} = (S, Act, \mathbf{P}, \mathsf{AP}, L)$: MDP; $Q\hat{\sigma}.Q_1\hat{s}_1(\hat{\sigma}).\ldots.Q_n\hat{s}_n(\hat{\sigma}).\varphi^{nq}$: HyperPCTL formula.

Output: Whether $\mathcal{M}$ satisfies the input formula.

1 **Function** `Main` $(\mathcal{M}, Q\hat{\sigma}.Q_1\hat{s}_1(\hat{\sigma}).\ldots.Q_n\hat{s}_n(\hat{\sigma}).\varphi^{nq})$ **:**
2 $E := \bigwedge_{s\in S}(\bigvee_{\alpha\in Act(s)} \sigma_s = \alpha)$; `// scheduler choice`
3 **if** *Q is existential* **then**
4 $E :=$ `Semantics`$(\mathcal{M}, \varphi^{nq}, n)$;
5 $E := E \wedge$ `Truth`$(\mathcal{M}, \exists\hat{\sigma}.Q_1\hat{s}_1(\hat{\sigma}).\ldots.Q_n\hat{s}_n(\hat{\sigma}).\varphi^{nq})$;
6 **if** $check(E) = SAT$ **then return** *TRUE*;
7 **else return** *FALSE*;
8 **else if** *Q is universal* **then**
 `//` $\overline{Q}_i$ `is` $\forall$ `if` $Q_i = \exists$ `and` $\exists$ `else`
9 $E :=$ `Semantics`$(\mathcal{M}, \neg\varphi^{nq}, n)$;
10 $E := E \wedge$ `Truth`$(\mathcal{M}, \exists\hat{\sigma}.\overline{Q}_1\hat{s}_1(\hat{\sigma}).\ldots.\overline{Q}_n\hat{s}_n(\hat{\sigma}).\neg\varphi^{nq})$;
11 **if** $check(E) = SAT$ **then return** *FALSE*;
12 **else return** *TRUE*;

The algorithm works by generating an SMT formula from the HyperPCTL input formula and MDP such that the generated formula is satisfiable exactly if the MDP satisfies the input formula. In line 2 of Algorithm 1, the main SMT encoding algorithm, the scheduler choice is encoded with a variable σ_s for each state s in the MDP, which must be assigned to one of the available actions in state s in any solution of the SMT formula. If the scheduler quantifier is existential, we first encode the semantics of the non-quantified part of the input formula (line 4), and then we encode which states must satisfy the non-quantified part of the formula for the entire formula to be satisfied (line 5). We then use an SMT solver to find a solution for this SMT formula. If there is one, the MDP satisfies the input formula and we return TRUE, otherwise we return FALSE. If the scheduler quantifier is a universal quantifier, we use the fact that $\forall\hat{\sigma}.\varphi$ is logically equivalent to $\neg\exists\hat{\sigma}.\neg\varphi$ by encoding the formula $\exists\hat{\sigma}.\neg\varphi$ and inverting the result of the SMT solver, which is what happens in lines 8-12 of Algorithm 1. In this case, a satisfying assignment for the SMT formula will give us a counterexample of a scheduler for which the formula is not satisfied.

Algorithm 2: SMT encoding for the meaning of the input formula

Input: $\mathcal{M} = (S, Act, \mathbf{P}, \mathsf{AP}, L)$: MDP; φ: quantifier-free HyperPCTL formula or expression; n: number of state variables in φ.

Output: SMT encoding of the meaning of φ in the n-ary self-composition of $\mathcal{M}$.

1 **Function** `Semantics` $(\mathcal{M}, \varphi, n)$ **:**

2 **if** φ *is true* **then** $E := \bigwedge_{s \in S^n} holds_{s,\varphi}$;

3 **else if** φ *is* $a_{\hat{s}_i}$ **then**

4 $E := (\bigwedge_{s \in S^n,\, a \in L(s_i)} (holds_{s,\varphi})) \wedge (\bigwedge_{s \in S^n,\, a \notin L(s_i)} (\neg holds_{s,\varphi}))$;

5 **else if** φ *is* $\varphi_1 \wedge \varphi_2$ **then**

6 $E := \texttt{Semantics}(\mathcal{M}, \varphi_1, n) \wedge \texttt{Semantics}(\mathcal{M}, \varphi_2, n) \wedge$

7 $\bigwedge_{s \in S^n} ((holds_{s,\varphi} \wedge holds_{s,\varphi_1} \wedge holds_{s,\varphi_2}) \vee$

8 $(\neg holds_{s,\varphi} \wedge (\neg holds_{s,\varphi_1} \vee \neg holds_{s,\varphi_2})))$;

9 **else if** φ *is* $\neg\varphi'$ **then**

10 $E := \texttt{Semantics}(\mathcal{M}, \varphi', n) \wedge \bigwedge_{s \in S^n} (holds_{s,\varphi} \oplus holds_{s,\varphi'})$;

11 **else if** φ *is* $\varphi_1^{ar} < \varphi_2^{ar}$ **then**

12 $E := \texttt{Semantics}(\mathcal{M}, \varphi_1^{ar}, n) \wedge \texttt{Semantics}(\mathcal{M}, \varphi_2^{ar}, n) \wedge$

13 $\bigwedge_{s \in S^n} ((holds_{s,\varphi} \wedge val_{s,\varphi_1^{ar}} < val_{s,\varphi_2^{ar}}) \vee$

14 $(\neg holds_{s,\varphi} \wedge val_{s,\varphi_1^{ar}} \geq val_{s,\varphi_2^{ar}}))$;

15 **else if** φ *is* $\mathbb{P}(\bigcirc\varphi')$ **then**

16 $E := \texttt{Semantics}(\mathcal{M}, \varphi', n) \wedge$

17 $\bigwedge_{s \in S^n} ((holdsToInt_{s,\varphi'} = 1 \wedge holds_{s,\varphi'}) \vee$

18 $(holdsToInt_{s,\varphi'} = 0 \wedge \neg holds_{s,\varphi'}))$;

19 **foreach** $s = (s_1, \ldots, s_n) \in S^n$ **do**

20 **foreach** $\alpha = (\alpha_1, \ldots, \alpha_n) \in Act(s_1) \times \ldots \times Act(s_n)$ **do**

21 $E := E \wedge ([\bigwedge_{i=1}^{n} \sigma_{s_i} = \alpha_i] \rightarrow [val_{s,\varphi} =$
$\sum_{s' \in supp(\alpha_1) \times \ldots \times supp(\alpha_n)} ((\prod_{i=1}^{n} P(s_i, \alpha_i, s_i')) \cdot holdsToInt_{s',\varphi'})])$;

22 **else if** φ *is* $\mathbb{P}(\varphi_1 \,\mathcal{U}^{[k_1,k_2]}\, \varphi_2)$ **then**
$E := \texttt{SemanticsBoundedUntil}(\mathcal{M}, \varphi, n)$;

23 **else if** φ *is* $\mathbb{P}(\varphi_1 \,\mathcal{U}\, \varphi_2)$ **then**
$E := \texttt{SemanticsUnboundedUntil}(\mathcal{M}, \varphi, n)$;

24 **else if** φ *is* c **then** $E := \bigwedge_{s \in S^n} (val_{s,\varphi} = c)$;

25 **else if** φ *is* φ_1^{ar} *op* φ_2^{ar} **then**

26 $E := \texttt{Semantics}(\mathcal{M}, \varphi_1^{ar}, n) \wedge \texttt{Semantics}(\mathcal{M}, \varphi_2^{ar}, n) \wedge$

27 $\bigwedge_{s \in S^n} (val_{s,\varphi} = (val_{s,\varphi_1^{ar}}$ *op* $val_{s,\varphi_2^{ar}}))$;

28 **return** E;

Algorithm 2 encodes the semantics of a quantifier-free formula or expression. Essentially, for every combination of the formula φ and a state $s \in S^n$, this function will create a variable $holds_{s,\varphi}$ with constraints such that any solution of the SMT formula will satisfy $holds_{s,\varphi}$ exactly if s satisfies φ for the scheduler chosen through the assignment of the σ_s variables. If φ is the boolean constant `true` for instance, the constraint $holds_{s,\varphi}$ will be added for every state $s \in S^n$ (line 2). If φ is an atomic proposition $a_{\hat{s}_i}$, we add the constraint $holds_{s,\varphi}$ for every state where s_i has

the label a and $\neg holds_{s,\varphi}$ for every other state. If φ contains subformulas, e.g. if φ is a conjunction $\varphi_1 \wedge \varphi_2$ of two formulas (line 5), we first encode the semantics of both subformulas before adding constraints for every state that force $holds_{s,\varphi}$ to be true in a solution if both $holds_{s,\varphi_1}$ and $holds_{s,\varphi_2}$ are true, and force it to be false if either of these variables is assigned to false (lines 7, 8). For every arithmetic expression that occurs in the formula, we add variables $val_{s,\varphi}$ for every state $s \in S^n$ that will hold the value of the arithmetic expression φ in the state s. The cases where φ is a constant value c (line 24) or an arithmetic operation performed on n other arithmetic expressions (line 25) are fairly simple: In the first case, we add a constraint that forces $val_{s,\varphi}$ to be equal to the constant value for every state; in the second case, we first encode the n operand expressions and then add a constraint that forces $val_{s,\varphi}$ to be the result of the mathematical operation performed on the corresponding variables. The cases where the arithmetic expression is of the form $\mathbb{P}(\varphi^{path})$ are more complicated, however. In these cases, $val_{s,\varphi}$ will hold the probability of satisfying φ^{path} in s. We first look at the case where φ^{path} is $\bigcirc\varphi'$, a next formula. We first encode the semantics for the formula φ' in line 16. After that, we introduce integer variables $holdsToInt_{s,\varphi'}$ that will be assigned 1 if s satisfies φ' and 0 if it does not. The value of $\mathbb{P}(\bigcirc\varphi')$ in state s is the probability that the next state transition will lead to a state that satisfies φ'. This means that the probability is the sum of the transition probabilities to all states that satisfy φ'. Obviously, we only need to sum over the set of states to which the transition probability is greater than 0. In the encoding, we therefore add that $val_{s,\varphi}$ should be equal to the sum of the transition probability multiplied by $holdsToInt_{s',\varphi'}$ over all states $s' \in supp(\alpha_1) \times \dots supp(\alpha_n)$, where $supp(\alpha_i)$ refers to the support of α_i, i.e. the states in S to which the transition probability from s_i is greater than 0 for the action α_i. Because the transition probabilities depend on the chosen scheduler, we add a separate implication for each possible combination of actions in the state s such that the left side of the implication is satisfied exactly if these actions are chosen through the assignment of the σ_s variables, and the right side of the implication contains the constraint that $val_{s,\varphi}$ is the desired sum with the transition probabilities for the chosen scheduler. For states that do not satisfy φ', $holdsToInt_{s',\varphi'}$ will be 0 and we therefore add 0 to $val_{s,\varphi}$ for these states. This can be seen in lines 20 and 21. As an example, we consider the expression $\varphi = \mathbb{P}(\bigcirc a)$, where we omit the state quantifier for simplicity, in the state s_0 in Figure 2.3. We can see that there is a probability of 0.4 to transition into s_1, which is labeled with a, a probability of 0.3 to transition into s_0, which is also labeled with a, and a probability of 0.3 to transition into s_2, which is not labeled with a. Accordingly, $holds_{s_0,a}$ and $holds_{s_1,a}$ will be true while $holds_{s_2,a}$ will be false. This results in $prob_{s_0,\mathbb{P}(\bigcirc a)} = 0.3 \cdot 1 + 0.4 \cdot 1 + 0.3 \cdot 0 = 0.7$. The cases where $\varphi = \mathbb{P}(\varphi^{path})$ with φ^{path} being either an unbounded or bounded until formula are more complicated and have been placed in separate functions because of this.

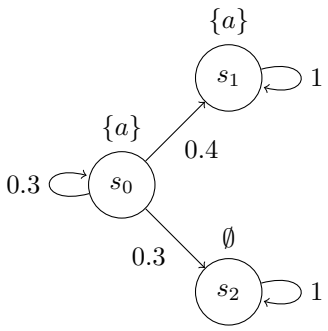

Figure 2.3: Example for the probability of a next formula.

We will first discuss the case where φ^{path} is a bounded until formula of the form $\mathbb{P}(\varphi_1 \mathcal{U}^{[k_1,k_2]} \varphi_2)$, which is handled in Algorithm 3. We would like $val_{s,\varphi}$ to have the

Algorithm 3: SMT encoding for the meaning of bounded until formulas

Input: $\mathcal{M} = (S, Act, \mathbf{P}, \mathsf{AP}, L)$: MDP; φ: HyperPCTL bounded until formula of the form $\mathbb{P}(\varphi_1 \mathcal{U}^{[k_1,k_2]} \varphi_2)$; n: number of state variables in φ.

Output: SMT encoding of φ's meaning in the n-ary self-composition of $\mathcal{M}$.

1 **Function** `SemanticsBoundedUntil` $(\mathcal{M}, \varphi = \mathbb{P}(\varphi_1 \mathcal{U}^{[k_1,k_2]} \varphi_2), n)$ **:**

2 $\quad$ **if** $k_2 = 0$ **then**

3 $\quad\quad E := \texttt{Semantics}(\mathcal{M}, \varphi_1, n) \wedge \texttt{Semantics}(\mathcal{M}, \varphi_2, n)$;

4 $\quad\quad$ **foreach** $s = (s_1, \ldots, s_n) \in S^n$ **do**

5 $\quad\quad\quad E := E \wedge (holds_{s,\varphi_2} \rightarrow val_{s,\varphi} = 1) \wedge (\neg holds_{s,\varphi_2} \rightarrow val_{s,\varphi} = 0)$;

6 $\quad$ **else if** $k_1 = 0$ **then**

7 $\quad\quad E := \texttt{SemanticsBoundedUntil}(\mathcal{M}, \varphi = \mathbb{P}(\varphi_1 \mathcal{U}^{[0,k_2-1]} \varphi_2), n)$;

8 $\quad\quad$ **foreach** $s = (s_1, \ldots, s_n) \in S^n$ **do**

9 $\quad\quad\quad E := E \wedge (holds_{s,\varphi_2} \rightarrow val_{s,\varphi} = 1) \wedge$

10 $\quad\quad\quad ((\neg holds_{s,\varphi_1} \wedge \neg holds_{s,\varphi_2}) \rightarrow val_{s,\varphi} = 0)$;

11 $\quad\quad\quad$ **foreach** $\alpha = (\alpha_1, \ldots, \alpha_n) \in Act(s_1) \times \ldots \times Act(s_n)$ **do**

12 $\quad\quad\quad\quad E := E \wedge ((holds_{s,\varphi_1} \wedge \neg holds_{s,\varphi_2} \wedge \bigwedge_{i=1}^{n} \sigma_{s_i} = \alpha_i) \rightarrow$

13 $\quad\quad\quad\quad (val_{s,\varphi} = \sum_{s' \in supp(\alpha_1) \times \ldots \times supp(\alpha_n)} ((\prod_{i=1}^{n} P(s_i, \alpha_i, s_i')) \cdot$

$\quad\quad\quad\quad\quad val_{s', \mathbb{P}(\varphi_1 \mathcal{U}^{[0,k_2-1]} \varphi_2)})))$;

14 $\quad$ **else if** $k_1 > 0$ **then**

15 $\quad\quad E := \texttt{SemanticsBoundedUntil}(\mathcal{M}, \varphi = \mathbb{P}(\varphi_1 \mathcal{U}^{[k_1-1,k_2-1]} \varphi_2), n)$;

16 $\quad\quad$ **foreach** $s = (s_1, \ldots, s_n) \in S^n$ **do**

17 $\quad\quad\quad E := E \wedge (\neg holds_{s,\varphi_1} \rightarrow val_{s,\varphi} = 0)$;

18 $\quad\quad\quad$ **foreach** $\alpha = (\alpha_1, \ldots, \alpha_n) \in Act(s_1) \times \ldots \times Act(s_n)$ **do**

19 $\quad\quad\quad\quad E := E \wedge ((holds_{s,\varphi_1} \wedge \bigwedge_{i=1}^{n} \sigma_{s_i} = \alpha_i) \rightarrow$

20 $\quad\quad\quad\quad (val_{s,\varphi} = \sum_{s' \in supp(\alpha_1) \times \ldots \times supp(\alpha_n)} ((\prod_{i=1}^{n} P(s_i, \alpha_i, s_i')) \cdot$

$\quad\quad\quad\quad\quad val_{s', \mathbb{P}(\varphi_1 \mathcal{U}^{[k_1-1,k_2-1]} \varphi_2)})))$;

21 $\quad$ **return** E;

probability of reaching a state that satisfies φ_2 in at least k_1, but at most k_2 state transitions when starting from s, with the constraint that every state on the path before that must satisfy φ_1. If the upper bound is 0, φ_2 has to be satisfied now or the probability will be 0, if the lower bound is 0 and the upper bound is not, we can satisfy φ_2 either now or later and if the lower bound is greater than 0, whether or not φ_2 is satisfied is irrelevant and it has to be satisfied later. We distinguish between three cases. If $k_2 = 0$, then a state satisfies φ_2 must be reached in 0 transitions, i.e. the current state must satisfy φ_2. Thus, if s satisfies φ_2, the desired probability is 1, otherwise it is 0. This case is covered in lines 2-5. The second case is if k_2 does not equal 0, but k_1 does. In this case we would like to reach a state that satisfies φ_2 in at most k_2 steps, with every state on the path before that satisfying φ_1. We first encode the semantics for reaching such a state in $k_2 - 1$ steps to obtain those probabilities. If s satisfies φ_2, then the probability is once again 1. On the other hand, if s does not satisfy φ_2, but also does not satisfy φ_1, the probability is 0 because every path from this state obviously begins in a state that does not satisfy φ_1. These two simple cases are shown in lines 9 and 10. If the state satisfies φ_1, but not φ_2, then

the desired probability is given as the expected value of the probability of reaching a state satisfying s_2 in $k_2 - 1$ steps from the successor states of s. Since the transition probabilities are dependent on the chosen scheduler, we add one implication for every possible combination of actions in s, like in the semantics for next formulas. Thus, if a scheduler is chosen, the state does not satisfy φ_2, but does satisfy φ_1, the probability will be the sum over all successor states of the probability $\mathbb{P}(\varphi_1 \mathcal{U}^{[0,k_2-1]} \varphi_2)$, multiplied by the probability to transition into that successor state with the chosen scheduler. This can be seen in lines 11-13. Finally, if k_1 is greater than 0, we want to reach a state satisfying φ_2 in at least k_1 steps, so it does not matter if the current state satisfies s_2 or not. Apart from that, this case is generally similar to the previous one. If the state does not satisfy φ_1, the probability is 0, if it does, we sum up over the possible successors the probabilities to reach a goal state in at least $k_1 - 1$ and at most $k_2 - 1$ steps. This final case is covered by lines 14-20.

Algorithm 4: SMT encoding for the meaning of unbounded until formulas

Input: $\mathcal{M} = (S, Act, \mathbf{P}, \mathsf{AP}, L)$: MDP; φ: HyperPCTL unbounded until formula of the form $\mathbb{P}(\varphi_1 \mathcal{U} \varphi_2)$; n: number of state variables in φ.
Output: SMT encoding of φ's meaning in the n-ary self-composition of $\mathcal{M}$.

1 **Function** `SemanticsUnboundedUntil`$(\mathcal{M}, \varphi = \mathbb{P}(\varphi_1 \mathcal{U} \varphi_2), n)$**:**
2 $\quad E := \texttt{Semantics}(\mathcal{M}, \varphi_1, n) \wedge \texttt{Semantics}(\mathcal{M}, \varphi_2, n)$;
3 $\quad$ **foreach** $s = (s_1, \ldots, s_n) \in S^n$ **do**
4 $\quad\quad E := E \wedge (holds_{s,\varphi_2} \rightarrow val_{s,\varphi} = 1) \wedge$
5 $\quad\quad ((\neg holds_{s,\varphi_1} \wedge \neg holds_{s,\varphi_2}) \rightarrow val_{s,\varphi} = 0)$;
6 $\quad\quad$ **foreach** $\alpha = (\alpha_1, \ldots, \alpha_n) \in Act(s_1) \times \ldots \times Act(s_n)$ **do**
7 $\quad\quad\quad E := E \wedge ((holds_{s,\varphi_1} \wedge \neg holds_{s,\varphi_2} \wedge \bigwedge_{i=1}^{n} \sigma_{s_i} = \alpha_i) \rightarrow$
8 $\quad\quad\quad (val_{s,\varphi} = \sum_{s' \in supp(\alpha_1) \times \ldots \times supp(\alpha_n)} ((\prod_{i=1}^{n} P(s_i, \alpha_i, s_i')) \cdot val_{s',\varphi'}) \wedge$
9 $\quad\quad\quad (val_{s,\varphi} > 0 \rightarrow (\bigvee_{s' \in supp(\alpha_1) \times \ldots \times supp(\alpha_n)} (holds_{s',\varphi_2} \vee d_{s,\varphi_2} >$
$\quad\quad\quad\quad d_{s',\varphi_2})))))$;
10 $\quad$ **return** E;

The case for an unbounded until formula, which is handled in Algorithm 4, is generally fairly similar to the bounded until formula with $k_2 > 0$ and $k_1 = 0$. The main difference can be found in line 9. In the bounded until formula, the time point at which a state can satisfy φ_2 is restricted. In the unbounded case however, there is no bound on the number of steps needed until reaching a state that satisfies φ_2. Figure 2.4 shows why this can cause issues. Let $\varphi = \mathbb{P}(\texttt{true}\,\mathcal{U}\,\texttt{false})$. Obviously, every state satisfies `true`, but no state satisfies `false`. Thus, the probability should always be 0 in all states. However, because of the cycle, it would be possible to assign every state the probability 1 in a solution for the SMT formula. Line 9 prevents this by assigning every state a d variable and forcing that at least one successor either satisfies φ_2 or has a lower d value than that of the current state if a probability larger

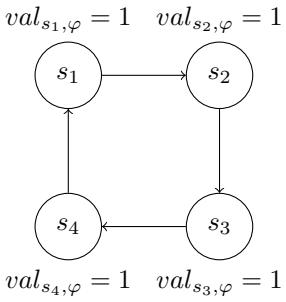

Figure 2.4: A possible variable assignment without line 9.

than 0 is assigned. Because of this requirement, we enforce smallest fixed points for the solutions and all positive probability values must lead to a state that satisfies φ_2 eventually. Because of this, the probabilities can be clearly determined for every scheduler.

Algorithm 5: SMT encoding of the truth of the input formula

Input: $\mathcal{M} = (S, Act, \mathbf{P}, \mathsf{AP}, L)$: MDP; $\exists\hat{\sigma}.Q_1\hat{s}_1(\hat{\sigma}).\ldots Q_n\hat{s}_n(\hat{\sigma}).\varphi^{nq}$: HyperPCTL formula.

Output: Encoding of the truth of the input formula in $\mathcal{M}$.

1 **Function** `Truth` $(\mathcal{M}, \exists\hat{\sigma}.Q_1\hat{s}_1(\hat{\sigma}).\ldots Q_n\hat{s}_n(\hat{\sigma}).\varphi^{nq})$ **:**

2 **foreach** $i = 1, \ldots, n$ **do**

3 **if** $Q_i = \forall$ **then** $B_i :=$ " $\bigwedge_{s_i \in S}$ ";

4 **else** $B_i :=$ " $\bigvee_{s_i \in S}$ ";

5 **return** $B_1 \ldots B_n$ $holds_{(s_1,\ldots,s_n),\varphi^{nq}}$;

The last algorithm, Algorithm 5, encodes the meaning of the state quantifiers. A forall quantifier $\forall\hat{s}_i(\sigma).\varphi$ means that every state $s_i \in S$ must satisfy the formula φ, so we add a conjunction over all possible states s_i. Likewise, an exists quantifier $\exists\hat{s}_i(\sigma).\varphi$ means that at least one state $s_i \in S$ must satisfy φ, so we create a disjunction over all s_i in this case. With this, we have encoded the entire input formula and MDP in an SMT formula that is satisfiable if and only if the MDP satisfies the input formula. Using an SMT solver, we can therefore solve the satisfiability problem for HyperPCTL restricted to non-probabilistic, memoryless schedulers.

Chapter 3

HyperPCTL with Rewards

In this chapter, we would like to modify HyperPCTL to allow reasoning about rewards. In the first section, we will modify the definitions of DTMCs and MDPs to include state rewards. After that, we extend the syntax and the semantics of HyperPCTL to include a new expression that can be used to calculate the expected rewards of paths in the model. Finally, we make the needed changes to the model checking algorithm introduced in the last chapter to include the new expression in the algorithm.

3.1 MDPs with Rewards

We would like to extend HyperPCTL with a rewards mechanism. To do this, we first need to extend the MDPs we are examining with a reward model. In this thesis, we are focusing on *state rewards*, i.e. we extend the definition of MDPs as follows:

Definition 3.1.1. *An* MDP with n-ary rewards (MDPR) *is defined as a tuple* $\mathcal{M} = (S, Act, \mathbf{P}, \mathsf{AP}, L, \mathbf{R})$ *where* $(S, Act, \mathbf{P}, \mathsf{AP}, L)$ *is an MDP and* $\mathbf{R} : S \to \mathbb{R}^n$ *is a reward function, assigning each state of the MDP a reward vector.*

We will assume all MDPRs to have unary rewards unless specified otherwise. Analogously to MDPRs, we can also add a reward function to the definition of DTMCs to obtain *DTMCs with n-ary rewards (DTMCRs)*. We will write $R_i(s)$ for $(R(s))_i$. We define the reward of a finite path $\pi = (s_0, s_1, \ldots, s_n)$ in the ith component as $\mathbf{R}_i(\pi) = \sum_{j=0}^{n} \mathbf{R}_i(s_j)$ for both MDPRs and DTMCRs. Furthermore, an MDPR and a scheduler will induce a DTMCR:

Definition 3.1.2. *Assume an MDPR* $\mathcal{M} = (S, Act, \mathbf{P}, \mathsf{AP}, L, \mathbf{R})$ *and a scheduler* $\sigma = (Q, act, mode, init)$ *for* $\mathcal{M}$*. The DTMCR induced by* $\mathcal{M}$ *and* σ *is defined as* $\mathcal{M}^\sigma = (S^\sigma, \mathbf{P}^\sigma, \mathsf{AP}, L^\sigma, \mathbf{R}^\sigma)$ *with* $(S^\sigma, \mathbf{P}^\sigma, \mathsf{AP}, L^\sigma) = (S, Act, \mathbf{P}, \mathsf{AP}, L)^\sigma$ *and* $\mathbf{R}^\sigma(q,s) = \mathbf{R}(s)$ *for all* $q \in Q, s \in S$*.*

For the semantics of HyperPCTL with rewards, we also need to define the n-ary self composition of an MDPR, which requires defining the parallel composition of two DTMCRs:

Definition 3.1.3. *The* parallel composition *of a DTMC with n-ary rewards* $\mathcal{D}_1 = (S_1, \mathbf{P}_1, \mathsf{AP}_1, L_1, \mathbf{R}_1)$ *and a DTMC with m-ary rewards* $\mathcal{D}_2 = (S_2, \mathbf{P}_2, \mathsf{AP}_2, L_2, \mathbf{R}_2)$ *is an* $(n+m)$*-ary DTMCR defined as* $\mathcal{D}_1 \| \mathcal{D}_2 = (S, \mathbf{P}, \mathsf{AP}, L, \mathbf{R})$*, where* $(S, \mathbf{P}, \mathsf{AP}, L)$

is the DTMC parallel composition $(S_1, \mathbf{P}_1, \mathsf{AP}_1, L_1) \| (S_2, \mathbf{P}_2, \mathsf{AP}_2, L_2)$ *and* $\mathbf{R} : S \to \mathbb{R}^{n+m}$ *with* $\mathbf{R}(s) = (\mathbf{R}_{1,1}(s_1), \ldots, \mathbf{R}_{1,n}(s_1), \mathbf{R}_{2,1}(s_1), \ldots, \mathbf{R}_{2,m}(s_1))$ *for all* $s = (s_1, s_2) \in S$.

Definition 3.1.4. *The* n-ary self composition *of an MDPR* $\mathcal{M} = (S, Act, \mathbf{P}, \mathsf{AP}, L, \mathbf{R})$ *for a sequence* $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_n) \in (\Sigma^{\mathcal{M}})^n$ *of schedulers for* $\mathcal{M}$ *is the DTMCR parallel composition* $\mathcal{M}^{\boldsymbol{\sigma}} = (S^{\boldsymbol{\sigma}}, \mathbf{P}^{\boldsymbol{\sigma}}, \mathsf{AP}^{\boldsymbol{\sigma}}, L^{\boldsymbol{\sigma}}, \mathbf{R}^{\boldsymbol{\sigma}}) = \mathcal{M}_1^{\sigma_1} \| \ldots \| \mathcal{M}_n^{\sigma_n}$, *where* $\mathcal{M}_i^{\sigma_i}$ *is the DTMCR induced by* $\mathcal{M}_i$ *and* σ_i, *with* $\mathcal{M}_i = (S, Act, \mathbf{P}, \mathsf{AP}_i, L_i, \mathbf{R})$ *with* $\mathsf{AP}_i = \{a_i \mid a \in \mathsf{AP}\}$ *and* $L_i(s) = \{a_i \mid a \in L(s)\}$ *for all* $s \in S$.

Figure 3.1 displays an MDPR as well as its binary self composition for the sequence of schedulers $\boldsymbol{\sigma} = (\sigma_1, \sigma_2)$ where σ_1 always selects action α in s_0 and σ_2 always selects β in s_0.

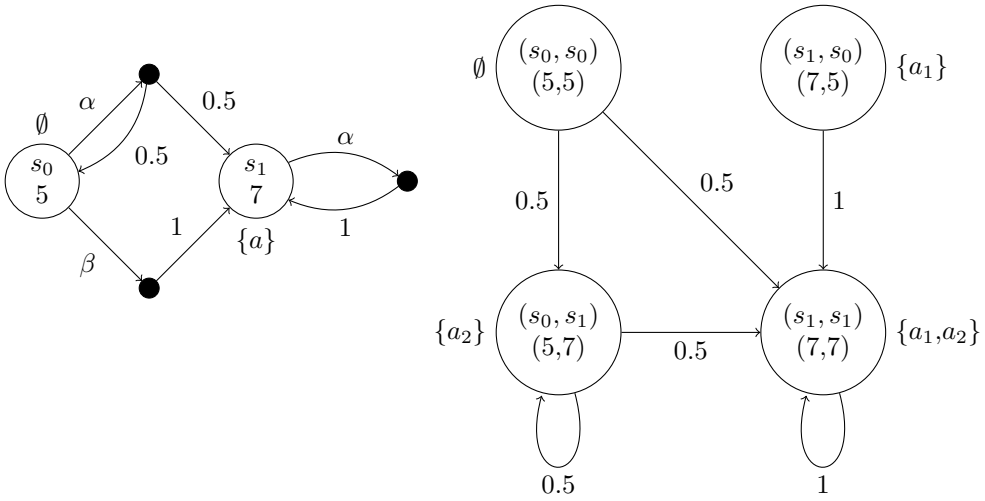

Figure 3.1: An MDPR and one of its binary self compositions.

3.2 Syntax & Semantics

To examine rewards in HyperPCTL, we must first extend the syntax of the logic with reward operators. We extend the syntax as follows:

Definition 3.2.1. HyperPCTL (quantified) state formulas φ^q *for MDPRs are inductively defined as follows:*

quantified formula	φ^q	::=	$\forall\hat{\sigma}.\varphi^q \mid \exists\hat{\sigma}.\varphi^q \mid \forall\hat{s}(\hat{\sigma}).\varphi^q \mid \exists\hat{s}(\hat{\sigma}).\varphi^q \mid \varphi^{nq}$
non-quantified formula	φ^{nq}	::=	$\texttt{true} \mid a_{\hat{s}} \mid \varphi^{nq} \wedge \varphi^{nq} \mid \neg\varphi^{nq} \mid \varphi^{ar} < \varphi^{ar}$
arithmetic expression	φ^{ar}	::=	$\mathbb{P}(\varphi^{path}) \mid \mathcal{R}_{\hat{s}}(\varphi^{path}) \mid f(\varphi_1^{ar}, \ldots, \varphi_k^{ar})$
path formula	φ^{path}	::=	$\bigcirc\varphi^{nq} \mid \varphi^{nq}\ \mathcal{U}\ \varphi^{nq} \mid \varphi^{nq}\ \mathcal{U}^{[k_1,k_2]}\ \varphi^{nq}$

Since the only difference from the original syntax consists of the addition of a reward operator that can be used as an arithmetic expression, all the expressivity of regular HyperPCTL is maintained. In an expression $\mathcal{R}_{\hat{s}}(\varphi^{path})$, we use $\hat{s}$ to specify for

which state quantifier we would like to accumulate rewards. We allow the following additional syntactic sugar:

$$\begin{array}{lcll} \mathcal{R}_{\hat{s}}(C_t) & := & \mathcal{R}_{\hat{s}}(\texttt{true}\,\mathcal{U}^{\,[t,t]}\texttt{true}) & \\ \mathcal{R}_{\hat{s}}(I_t) & := & \mathcal{R}_{\hat{s}}(C_t) - \mathcal{R}_{\hat{s}}(C_{t-1}) & , \textit{if } t > 0 \\ & & \mathcal{R}_{\hat{s}}(C_t) & , \textit{else} \end{array}$$

These are similar to properties that can be used in the probabilistic model checker PRISM [KNP11]. Essentially, the expression $\mathcal{R}_{\hat{s}}(C_t)$ will have the expected cumulative reward accumulated over the next t steps as its value, while $\mathcal{R}_{\hat{s}}(I_t)$ will have the expected reward in the state reached after t steps as its value.

Intuitively, the value of $\mathcal{R}_{\hat{s}}(\bigcirc\varphi^{nq})$ is the expected reward of the successor state plus the reward of the current state, if the probability that the successor state satisfies φ^{nq} is 1. The value of $\mathcal{R}_{\hat{s}}(\varphi_1^{nq}\,\mathcal{U}\,\varphi_2^{nq})$ or $\mathcal{R}_{\hat{s}}(\varphi_1^{nq}\,\mathcal{U}^{\,[k_1,k_2]}\,\varphi_2^{nq})$ is the expected cumulative reward accumulated until the first time a state is reached that satisfies φ_2^{nq}, as long as the probability of satisfying $\varphi_1^{nq}\,\mathcal{U}\,\varphi_2^{nq}$, or $\varphi_1^{nq}\,\mathcal{U}^{\,[k_1,k_2]}\,\varphi_2^{nq}$ respectively, is 1. The value of $\mathcal{R}_{\hat{s}}(\varphi^{path})$ is undefined if the value of $\mathbb{P}(\varphi^{path})$ is not equal to 1.

We define the following:

$$\begin{array}{lcl} \mathit{fPaths}(\varphi_1^{nq}\,\mathcal{U}\,\varphi_2^{nq})_{\boldsymbol{r}}^{\mathcal{M}^{\boldsymbol{\sigma}}} & = & \{\pi \in \mathit{fPaths}_{\boldsymbol{r}}^{\mathcal{M}^{\boldsymbol{\sigma}}} \mid \exists \pi' \in \mathit{Paths}_{\boldsymbol{r}}^{\mathcal{M}^{\boldsymbol{\sigma}}} : \pi \textit{ is a prefix of } \pi' \wedge \\ & & \mathcal{M},\boldsymbol{\sigma},\pi' \models \varphi_1^{nq}\,\mathcal{U}^{\,[|\pi|,|\pi|]}\,\varphi_2^{nq} \wedge \mathcal{M},\boldsymbol{\sigma},\pi'_j \not\models \varphi_2 \forall j < |\pi|\} \\ \mathit{fPaths}(\varphi_1^{nq}\,\mathcal{U}^{\,[k_1,k_2]}\,\varphi_2^{nq})_{r}^{\mathcal{M}^{\boldsymbol{\sigma}}} & = & \{\pi \in \mathit{fPaths}_{r}^{\mathcal{M}^{\boldsymbol{\sigma}}} \mid \exists \pi' \in \mathit{Paths}_{\boldsymbol{r}}^{\mathcal{M}^{\boldsymbol{\sigma}}} : \pi \textit{ is a prefix of } \pi' \wedge \\ & & k_1 \leq |\pi| \leq k_2 \wedge \mathcal{M},\boldsymbol{\sigma},\pi' \models \varphi_1^{nq}\,\mathcal{U}^{\,[|\pi|,|\pi|]}\,\varphi_2^{nq} \wedge \\ & & \mathcal{M},\boldsymbol{\sigma},\pi'_j \not\models \varphi_2 \forall k_1 \leq j < |\pi|\} \end{array}$$

Formally, the semantics for the reward operator $[\![\mathcal{R}_{s_i}(\varphi^{path})]\!]_{\mathcal{M},\boldsymbol{\sigma},\boldsymbol{r}}$ is as follows, given that $[\![\mathbb{P}(\varphi^{path})]\!]_{\mathcal{M},\boldsymbol{\sigma},\boldsymbol{r}} = 1$. If that is not the case, $[\![\mathcal{R}_{s_i}(\varphi^{path})]\!]_{\mathcal{M},\boldsymbol{\sigma},\boldsymbol{r}}$ is undefined.

$$\begin{array}{lcl} [\![\mathcal{R}_{s_i}(\bigcirc\varphi^{nq})]\!]_{\mathcal{M},\boldsymbol{\sigma},\boldsymbol{r}} & = & \mathbf{R}_i^{\boldsymbol{\sigma}}(\boldsymbol{r}) + \sum_{\boldsymbol{r}' \in S^{\boldsymbol{\sigma}}} \mathbf{P}^{\boldsymbol{\sigma}}(\boldsymbol{r},\boldsymbol{r}') \cdot \mathbf{R}_i^{\boldsymbol{\sigma}}(\boldsymbol{r}') \\ [\![\mathcal{R}_{s_i}(\varphi_1^{nq}\,\mathcal{U}\,\varphi_2^{nq})]\!]_{\mathcal{M},\boldsymbol{\sigma},\boldsymbol{r}} & = & \sum_{\pi \in \mathit{fPaths}(\varphi_1^{nq}\,\mathcal{U}\,\varphi_2^{nq})_{\boldsymbol{r}}^{\mathcal{M}^{\boldsymbol{\sigma}}}} (\mathbf{P}^{\boldsymbol{\sigma}}(\pi) \cdot \mathbf{R}_i^{\boldsymbol{\sigma}}(\pi)) \\ [\![\mathcal{R}_{s_i}(\varphi_1^{nq}\,\mathcal{U}^{\,[k_1,k_2]}\varphi_2^{nq})]\!]_{\mathcal{M},\boldsymbol{\sigma},\boldsymbol{r}} & = & \sum_{\pi \in \mathit{fPaths}(\varphi_1^{nq}\,\mathcal{U}^{\,[k_1,k_2]}\varphi_2^{nq})_{\boldsymbol{r}}^{\mathcal{M}^{\boldsymbol{\sigma}}}} (\mathbf{P}^{\boldsymbol{\sigma}}(\pi) \cdot \mathbf{R}_i^{\boldsymbol{\sigma}}(\pi)) \end{array}$$

Since arithmetic values can be undefined, it is necessary to also define the semantics for the handling of undefined values. Formally defining in which cases a value is undefined would be very complicated, as this needs to be done for every type of expression that can be used in HyperPCTL. We will therefore forgo a formal definition at this point and use the following more informal definition:

Definition 3.2.2. *The value $[\![\varphi]\!]_{\mathcal{M},\boldsymbol{\sigma},\boldsymbol{r}/\pi}$ of an expression φ is undefined for an MDPR $\mathcal{M}$, a sequence of schedulers $\boldsymbol{\sigma}$ and a state $\boldsymbol{r} \in S^{\boldsymbol{\sigma}}$ or a path $\pi \in \mathit{Paths}_{\boldsymbol{r}}^{\mathcal{M}^{\boldsymbol{\sigma}}}$ exactly if*

- *φ is of the form $\mathcal{R}_{s_i}(\varphi^{path})$ and $[\![\mathbb{P}(\varphi^{path})]\!]_{\mathcal{M},\boldsymbol{\sigma},\boldsymbol{r}} \neq 1$ or*
- *the value of φ for $\mathcal{M},\boldsymbol{\sigma}$ and $\boldsymbol{r}$ or π is dependent on one or more other expressions $\varphi_1,\ldots,\varphi_n$ with undefined values such that there are multiple different values that $[\![\varphi]\!]_{\mathcal{M},\boldsymbol{\sigma},\boldsymbol{r}/\pi}$ could take for different values of $\varphi_1,\ldots,\varphi_n$.*

We will go into further detail on when a value is undefined when presenting the algorithm in the next section.

3.3 Model Checking Algorithm

We will now present the modified algorithm for model checking of HyperPCTL with rewards. A significant difficulty is added due to the fact that rewards of until and next formulas are undefined in states where these formulas are not or not guaranteed to be satisfied. Clearly, a comparison between two reward values where one of them is undefined cannot be assigned a clear truth value. To solve this problem, we make some major changes to the algorithm. First, we allow the previously boolean variables $holds_{s,\varphi}$ to take three different values: *true*, *false* and *undefined*. We will use 1, 0 and $\bot$ as shorthand for these values in the algorithm. We extend the semantics of the $\wedge$ and $\neg$ operators for $\bot$ as follows:

$$\begin{array}{lcl} \bot \wedge 0 & = & 0 \\ \bot \wedge \bot & = & \bot \\ \bot \wedge 1 & = & \bot \\ \neg\bot & = & \bot \end{array}$$

Furthermore, for every variable $val_{s,\varphi}$, we introduce an additional boolean variable $def_{s,\varphi}$ that states whether or not the value of the arithmetic expression φ is defined in s. Since expressions of the form $\mathcal{R}_{\hat{s}}(\varphi^{path})$ are arithmetic expressions, these will also have $val_{s,\varphi}$ variables holding the expected reward for satisfying φ in s as well as $def_{s,\varphi}$ variables stating whether or not this reward is defined. In the following, we will detail the modified algorithm, explain in which cases a value is undefined, and detail the new additions to encode the semantics of reward-related expressions.

Algorithm 6: Main SMT encoding algorithm

Input: $\mathcal{M} = (S, Act, \mathbf{P}, \mathsf{AP}, L, \mathbf{R})$: MDPR; $Q_1\hat{s}_1 \ldots Q_n\hat{s}_n.\varphi^{nq}$: HyperPCTL formula.

Output: Whether $\mathcal{M}$ satisfies the input formula.

Function `Main` $(\mathcal{M}, Q\hat{\sigma}.Q_1\hat{s}_1(\hat{\sigma}) \ldots Q_n\hat{s}_n(\hat{\sigma}).\varphi^{nq})$ **:**
- $E := \bigwedge_{s\in S}(\bigvee_{\alpha\in Act(s)} \sigma_s = \alpha)$; `// scheduler choice`
- **if** *Q is existential* **then**
  - $E :=$ `Semantics`$(\mathcal{M}, \varphi^{nq}, n)$;
  - $T := E \wedge$ `Truth`$(\mathcal{M}, \exists\hat{\sigma}.Q_1\hat{s}_1(\hat{\sigma}) \ldots Q_n\hat{s}_n(\hat{\sigma}).\varphi^{nq})$;
  - $N := E \wedge$ `NotFalsehood`$(\mathcal{M}, \exists\hat{\sigma}.Q_1\hat{s}_1(\hat{\sigma}) \ldots Q_n\hat{s}_n(\hat{\sigma}).\varphi^{nq})$;
  - **if** $check(T) = SAT$ **then return** $TRUE$;
  - **else if** $check(N) = SAT$ **then return** $UNDEF$;
  - **else return** $FALSE$;
- **else if** *Q is universal* **then**
  - `//` $\overline{Q}_i$ `is` $\forall$ `if` $Q_i = \exists$ `and` $\exists$ `else`
  - $E :=$ `Semantics`$(\mathcal{M}, \neg\varphi^{nq}, n)$;
  - $T := E \wedge$ `Truth`$(\mathcal{M}, \exists\hat{\sigma}.\overline{Q}_1\hat{s}_1(\hat{\sigma}) \ldots \overline{Q}_n\hat{s}_n(\hat{\sigma}).\neg\varphi^{nq})$;
  - $N := E \wedge$ `NotFalsehood`$(\mathcal{M}, \exists\hat{\sigma}.\overline{Q}_1\hat{s}_1(\hat{\sigma}) \ldots \overline{Q}_n\hat{s}_n(\hat{\sigma}).\neg\varphi^{nq})$;
  - **if** $check(T) = SAT$ **then return** $FALSE$;
  - **else if** $check(N) = SAT$ **then return** $UNDEF$;
  - **else return** $TRUE$;

Due to the introduction of undefined values, a single satisfiability check is not sufficient to differentiate between all three cases. We therefore generate two different

Algorithm 7: SMT encoding for the meaning of the input formula

Input: $\mathcal{M} = (S, Act, \mathbf{P}, \mathsf{AP}, L, \mathbf{R})$: MDPR; φ: quantifier-free HyperPCTL formula or expression; n: number of state variables in φ.

Output: SMT encoding of the meaning of φ in the n-ary self-composition of $\mathcal{M}$.

Function `Semantics` $(\mathcal{M}, \varphi, n)$**:**

- **if** φ *is true* **then** $E := \bigwedge_{s \in S^n} holds_{s,\varphi} = 1$;
- **else if** φ *is* $a_{\hat{s}_i}$ **then**
  - $E := (\bigwedge_{s \in S^n,\, a \in L(s_i)} (holds_{s,\varphi} = 1)) \wedge (\bigwedge_{s \in S^n,\, a \notin L(s_i)} (holds_{s,\varphi} = 0))$;
- **else if** φ *is* $\varphi_1 \wedge \varphi_2$ **then**
  - $E :=$ `Semantics`$(\mathcal{M}, \varphi_1, n) \wedge$ `Semantics`$(\mathcal{M}, \varphi_2, n) \wedge$
  - $\bigwedge_{s \in S^n} ((holds_{s,\varphi_1} = 1 \wedge holds_{s,\varphi_2} = 1) \rightarrow holds_{s,\varphi} = 1) \wedge$
  - $\bigwedge_{s \in S^n} ((holds_{s,\varphi_1} = 0 \vee holds_{s,\varphi_2} = 0) \rightarrow holds_{s,\varphi} = 0) \wedge$
  - $\bigwedge_{s \in S^n} ((holds_{s,\varphi_1} = \bot \wedge holds_{s,\varphi_2} = 1) \rightarrow holds_{s,\varphi} = \bot) \wedge$
  - $\bigwedge_{s \in S^n} ((holds_{s,\varphi_1} = 1 \wedge holds_{s,\varphi_2} = \bot) \rightarrow holds_{s,\varphi} = \bot) \wedge$
  - $\bigwedge_{s \in S^n} ((holds_{s,\varphi_1} = \bot \wedge holds_{s,\varphi_2} = \bot) \rightarrow holds_{s,\varphi} = \bot)$;
- **else if** φ *is* $\neg\varphi'$ **then**
  - $E :=$ `Semantics`$(\mathcal{M}, \varphi', n) \wedge$
  - $\bigwedge_{s \in S^n} (holds_{s,\varphi'} = 0 \rightarrow holds_{s,\varphi} = 1) \wedge$
  - $\bigwedge_{s \in S^n} (holds_{s,\varphi'} = 1 \rightarrow holds_{s,\varphi} = 0) \wedge$
  - $\bigwedge_{s \in S^n} (holds_{s,\varphi'} = \bot \rightarrow holds_{s,\varphi} = \bot)$;
- **else if** φ *is* $\varphi_1^{ar} < \varphi_2^{ar}$ **then**
  - $E :=$ `Semantics`$(\mathcal{M}, \varphi_1^{ar}, n) \wedge$ `Semantics`$(\mathcal{M}, \varphi_2^{ar}, n) \wedge$
  - $\bigwedge_{s \in S^n} ((def_{s,\varphi_1^{ar}} \wedge def_{s,\varphi_2^{ar}} \wedge val_{s,\varphi_1^{ar}} < val_{s,\varphi_2^{ar}}) \rightarrow holds_{s,\varphi} = 1) \wedge$
  - $\bigwedge_{s \in S^n} ((def_{s,\varphi_1^{ar}} \wedge def_{s,\varphi_2^{ar}} \wedge val_{s,\varphi_1^{ar}} \geq val_{s,\varphi_2^{ar}}) \rightarrow holds_{s,\varphi} = 0) \wedge$
  - $\bigwedge_{s \in S^n} ((\neg def_{s,\varphi_1^{ar}} \vee \neg def_{s,\varphi_2^{ar}}) \rightarrow holds_{s,\varphi} = \bot)$;
- **else if** φ *is* $\mathbb{P}(\bigcirc\varphi')$ **then**
  - $E :=$ `Semantics`$(\mathcal{M}, \varphi', n) \wedge$
  - $\bigwedge_{s \in S^n} (holds_{s,\varphi'} = 0 \rightarrow holdsToInt_{s,\varphi'} = 0) \wedge$
  - $\bigwedge_{s \in S^n} (holds_{s,\varphi'} = 1 \rightarrow holdsToInt_{s,\varphi'} = 1)$;
  - **foreach** $s = (s_1, \ldots, s_n) \in S^n$ **do**
    - **foreach** $\alpha = (\alpha_1, \ldots, \alpha_n) \in Act(s_1) \times \ldots \times Act(s_n)$ **do**
      - $E := E \wedge ([\bigwedge_{i=1}^{n} \sigma_{s_i} = \alpha_i] \rightarrow [$
      - $((\bigvee_{s' \in supp(\alpha_1) \times \ldots \times supp(\alpha_n)} (holds_{s',\varphi'} = \bot)) \leftrightarrow \neg def_{s,\varphi}) \wedge$
      - $(def_{s,\varphi} \rightarrow val_{s,\varphi} = \sum_{s' \in supp(\alpha_1) \times \ldots \times supp(\alpha_n)} ((\prod_{i=1}^{n} P(s_i, \alpha_i, s_i')) \cdot holdsToInt_{s',\varphi'}))])$;
- **else if** φ *is* $\mathbb{P}(\varphi_1 \,\mathcal{U}\, \varphi_2)$ **then**
  - $E :=$ `SemanticsUnboundedUntil`$(\mathcal{M}, \varphi, n)$;
- **else if** φ *is* $\mathbb{P}(\varphi_1 \,\mathcal{U}^{[k_1,k_2]} \varphi_2)$ **then**
  - $E :=$ `SemanticsBoundedUntil`$(\mathcal{M}, \varphi, n)$;
- **else if** φ *is* c **then** $E := \bigwedge_{s \in S^n} (val_{s,\varphi} = c \wedge def_{s,\varphi})$;
- **else if** φ *is* φ_1^{ar} *op* φ_2^{ar} **then**
  - $E :=$ `Semantics`$(\mathcal{M}, \varphi_1^{ar}, n) \wedge$ `Semantics`$(\mathcal{M}, \varphi_2^{ar}, n) \wedge$
  - $\bigwedge_{s \in S^n} ((def_{s,\varphi_1^{ar}} \wedge def_{s,\varphi_2^{ar}}) \rightarrow (def_{s,\varphi} \wedge val_{s,\varphi} = (val_{s,\varphi_1^{ar}}\ op\ val_{s,\varphi_2^{ar}}))) \wedge$
  - $\bigwedge_{s \in S^n} ((\neg def_{s,\varphi_1^{ar}} \vee \neg def_{s,\varphi_2^{ar}}) \rightarrow \neg def_{s,\varphi})$;
- **else** $E :=$ `RewSemantics`$(\mathcal{M}, \varphi, n)$;
- **return** E;

SMT formulas. In the case of an existential scheduler quantifier, the first one, shown in line 5 of Algorithm 6, will be satisfiable if and only if $\mathcal{M}$ satisfies the input formula, the second (line 6) will be true if and only if $\mathcal{M}$ satisfies the formula or the satisfaction value is undefined. This will allow us to distinguish between all three cases. In the case of a universal scheduler quantifier, we negate the input formula as well as the result of the satisfiability check, just like in the original algorithm.

The encoding for the cases where φ is the boolean constant `true` or an atomic proposition are identical to the original, unmodified algorithm, as seen in lines 2-4 of Algorithm 7. For the case where φ is a conjunction $\varphi_1 \wedge \varphi_2$, we add several implications to force $holds_{s,\varphi}$ to take the correct value as described above for every possible combination of $holds_{s,\varphi_1}$ and $holds_{s,\varphi_2}$. This is encoded in lines 5-11. If φ is a negation $\neg\varphi'$, we add three implications for the three possible values of $holds_{s,\varphi'}$. If φ is a comparison $\varphi_1^{ar} < \varphi_2^{ar}$, then we consider its satisfiability to be undefined in a state s if one of the two values that are compared is undefined in the state s (line 21). If both values are defined, then the satisfiability is clearly defined as before, which is encoded in lines 19 and 20. The probability value of a next formula $\mathbb{P}(\bigcirc\varphi')$ in a state s is dependent on the satisfaction value of φ' in all possible successors of s. Thus, if the satisfaction value of φ' is undefined in at least one successor of s, we

Algorithm 8: SMT encoding for the meaning of reward-related input formula

Input: $\mathcal{M} = (S, Act, \mathbf{P}, \mathsf{AP}, L, \mathbf{R})$: MDPR; φ: quantifier-free HyperPCTL formula or expression; n: number of state variables in φ.

Output: SMT encoding of the meaning of φ in the n-ary self-composition of $\mathcal{M}$.

1 **Function** `RewSemantics` $(\mathcal{M}, \varphi, n)$ **:**
2 **if** φ *is* $\mathcal{R}_{\hat{s}_i}(\bigcirc\varphi')$ **then**
3 $E :=$ `Semantics`$(\mathcal{M}, \mathbb{P}(\bigcirc\varphi'), n)$;
4 **foreach** $s = (s_1, \ldots, s_n) \in S^n$ **do**
5 $E := E \wedge ((val_{s,\mathbb{P}(\bigcirc\varphi')} \neq 1 \vee \neg def_{s,\mathbb{P}(\bigcirc\varphi')}) \leftrightarrow \neg def_{s,\varphi})$;
6 **foreach** $\alpha = (\alpha_1, \ldots, \alpha_n) \in Act(s_1) \times \ldots \times Act(s_n)$ **do**
7 $E := E \wedge ([def_{s,\varphi} \wedge \bigwedge_{j=1}^{n} \sigma_{s_j} = \alpha_j] \rightarrow [val_{s,\varphi} =$
 $\mathbf{R}(s_i) + \sum_{s' \in supp(\alpha_1) \times \ldots \times supp(\alpha_n)} ((\prod_{j=1}^{n} P(s_j, \alpha_j, s'_j)) \cdot \mathbf{R}(s'_i))])$;
8 **else if** φ *is* $\mathcal{R}_{\hat{s}_i}(C_t)$ **then**
9 $E :=$ `CumulativeReward`$(\mathcal{M}, \varphi, n)$;
10 **else if** φ *is* $\mathcal{R}_{\hat{s}_i}(I_t)$ **then**
11 $E :=$ `InstantaneousReward`$(\mathcal{M}, \varphi, n)$;
12 **else if** φ *is* $\mathcal{R}_{\hat{s}_i}(\varphi_1 \mathcal{U} \varphi_2)$ **then**
13 $E :=$ `SemanticsUnboundedUntil`$(\mathcal{M}, \mathbb{P}(\varphi_1 \mathcal{U} \varphi_2), n)$;
14 $E := E \wedge$ `RewardUnboundedUntil`$(\mathcal{M}, \varphi, n)$;
15 **else if** φ *is* $\mathcal{R}_{\hat{s}_i}(\varphi_1 \mathcal{U}^{[k_1,k_2]} \varphi_2)$ **then**
16 $E :=$ `SemanticsBoundedUntil`$(\mathcal{M}, \mathbb{P}(\varphi_1 \mathcal{U}^{[k_1,k_2]} \varphi_2), n)$;
17 $E := E \wedge$ `RewardBoundedUntil`$(\mathcal{M}, \varphi, n)$;
18 **return** E;

Algorithm 9: SMT encoding for the meaning of bounded until formulas

Input: $\mathcal{M} = (S, Act, \mathbf{P}, \mathsf{AP}, L, \mathbf{R})$: MDPR; φ: HyperPCTL bounded until formula of the form $\mathbb{P}(\varphi_1 \mathcal{U}^{[k_1,k_2]} \varphi_2)$; n: number of state variables in φ.

Output: SMT encoding of φ's meaning in the n-ary self-composition of $\mathcal{M}$.

1 **Function** `SemanticsBoundedUntil` $(\mathcal{M}, \varphi = \mathbb{P}(\varphi_1 \mathcal{U}^{[k_1,k_2]} \varphi_2), n)$ **:**

2 **if** $k_2 = 0$ **then**

3 $E :=$ `Semantics`$(\mathcal{M}, \varphi_1, n) \wedge$ `Semantics`$(\mathcal{M}, \varphi_2, n)$;

4 **foreach** $s = (s_1, \ldots, s_n) \in S^n$ **do**

5 $E := E \wedge (holds_{s,\varphi_2} = 1 \rightarrow (prob_{s,\varphi} = 1 \wedge def_{s,\varphi}))$;

6 $E := E \wedge (holds_{s,\varphi_2} = 0 \rightarrow (prob_{s,\varphi} = 0 \wedge def_{s,\varphi}))$;

7 $E := E \wedge (holds_{s,\varphi_2} = \bot \rightarrow \neg def_{s,\varphi})$;

8 **else if** $k_1 = 0$ **then**

9 $E :=$ `SemanticsBoundedUntil`$(\mathcal{M}, \varphi = \mathbb{P}(\varphi_1 \mathcal{U}^{[0,k_2-1]} \varphi_2), n)$;

10 **foreach** $s = (s_1, \ldots, s_n) \in S^n$ **do**

11 $E := E \wedge (holds_{s,\varphi_2} = 1 \rightarrow (prob_{s,\varphi} = 1 \wedge def_{s,\varphi}))$;

12 $E := E \wedge ((holds_{s,\varphi_1} = 0 \wedge holds_{s,\varphi_2} = 0) \rightarrow (prob_{s,\varphi} = 0 \wedge def_{s,\varphi}))$;

13 $E := E \wedge ((holds_{s,\varphi_1} = 0 \wedge holds_{s,\varphi_2} = \bot) \rightarrow \neg def_{s,\varphi})$;

14 **foreach** $\alpha = (\alpha_1, \ldots, \alpha_n) \in Act(s_1) \times \ldots \times Act(s_n)$ **do**

15 $E := E \wedge ((holds_{s,\varphi_1} \neq 0 \wedge holds_{s,\varphi_2} \neq 1 \wedge \bigwedge_{i=1}^{n} \sigma_{s_i} = \alpha_i) \rightarrow$

16 $[prob_{s,\varphi} = \sum_{s' \in supp(\alpha_1) \times \ldots \times supp(\alpha_n)} ((\prod_{i=1}^{n} P(s_i, \alpha_i, s_i')) \cdot$
$prob_{s', \mathbb{P}(\varphi_1 \mathcal{U}^{[0,k_2-1]} \varphi_2)}) \wedge$

17 $(\neg def_{s,\varphi} \leftrightarrow [(\bigvee_{s' \in supp(\alpha_1) \times \ldots \times supp(\alpha_n)} \neg def_{s', \mathbb{P}(\varphi_1 \mathcal{U}^{[0,k_2-1]} \varphi_2)}) \vee$

18 $(holds_{s,\varphi_1} = \bot \wedge holds_{s,\varphi_2} = \bot) \vee$

19 $(holds_{s,\varphi_1} = \bot \wedge holds_{s,\varphi_2} = 0 \wedge prob_{s,\varphi} \neq 0) \vee$

20 $(holds_{s,\varphi_1} = 1 \wedge holds_{s,\varphi_2} = \bot \wedge prob_{s,\varphi} \neq 1)])])$;

21 **else if** $k_1 > 0$ **then**

22 $E :=$ `SemanticsBoundedUntil`$(\mathcal{M}, \varphi = \mathbb{P}(\varphi_1 \mathcal{U}^{[k_1-1,k_2-1]} \varphi_2), n)$;

23 **foreach** $s = (s_1, \ldots, s_n) \in S^n$ **do**

24 $E := E \wedge (holds_{s,\varphi_1} = 0 \rightarrow (prob_{s,\varphi} = 0 \wedge def_{s,\varphi}))$;

25 $E := E \wedge ((holds_{s,\varphi_1} = \bot \wedge prob_{s,\varphi} \neq 0) \rightarrow \neg def_{s,\varphi})$;

26 **foreach** $\alpha = (\alpha_1, \ldots, \alpha_n) \in Act(s_1) \times \ldots \times Act(s_n)$ **do**

27 $E := E \wedge ((holds_{s,\varphi_1} \neq 0 \wedge \bigwedge_{i=1}^{n} \sigma_{s_i} = \alpha_i) \rightarrow$

28 $[prob_{s,\varphi} = \sum_{s' \in supp(\alpha_1) \times \ldots \times supp(\alpha_n)} ((\prod_{i=1}^{n} P(s_i, \alpha_i, s_i')) \cdot$
$prob_{s', \mathbb{P}(\varphi_1 \mathcal{U}^{[k_1-1,k_2-1]} \varphi_2)}) \wedge$

29 $(\neg def_{s,\varphi} \leftrightarrow$
$[(\bigvee_{s' \in supp(\alpha_1) \times \ldots \times supp(\alpha_n)} \neg def_{s', \mathbb{P}(\varphi_1 \mathcal{U}^{[k_1-1,k_2-1]} \varphi_2)}) \vee$

30 $(holds_{s,\varphi_1} = \bot \wedge prob_{s,\varphi} \neq 0)])])$;

31 **return** E;

do not assign a clear probability value to φ and it is therefore undefined (line 29). If the probability is defined, it is calculated as before (line 30). The semantics for until formulas is more complicated once again and have been placed in separate functions.

If an arithmetic expression is a constant, then its value is always defined, as covered in line 33. The value of a function applied to n arithmetic expressions is defined if the values of all of the expressions are defined (line 36), and it is undefined if one of the values is undefined (line 37).

Algorithm 8 encodes the semantics for those expressions that were added to the syntax of HyperPCTL for arguing about rewards. In the case that φ is $\mathcal{R}_{\hat{s}_i}(\bigcirc\varphi')$, we first encode the corresponding probability expression so we can use the probability of satisfying $\bigcirc\varphi'$ in the encoding for the reward expression. This reward is undefined if the probability is not 1 or if it is undefined (line 5). If the probability is defined, then it is simply the sum of the reward of the ith component of the current state and the expected reward in the next state (line 7). The other cases are more involved, so the encoding is done in individual functions for each. For until formulas, we also encode the equivalent probability expression first to be able to use these values in the encoding.

Algorithm 9 is the modified algorithm for encoding the semantics of a formula φ of the form $\mathbb{P}(\varphi_1 \mathcal{U}^{[k_1,k_2]} \varphi_2)$. We distinguish between the same three cases as in the original algorithm. The case where $k_2 = 0$ is simple: If a state s satisfies φ_2, the probability is 1, if it does not satisfy φ_2, the probability is 0 and if the satisfaction of φ_2 in s is undefined, the probability is also undefined. We have to be careful how to handle the case where $k_2 \neq 0$, but $k_1 = 0$. Table 3.1 details in which cases the probability will be defined, with * standing for an arbitrary value. This table is split into two parts. The first part depicts the cases where definedness and the probability can be derived from the two *holds* variables alone. If the state s satisfies φ_2, the probability will always be defined and 1. This case is covered in line 11. If the state satisfies neither φ_1 nor φ_2, the probability will be defined to be 0 (line 12). However, if the state does not satisfy φ_1 and the satisfaction of φ_2 is undefined, the probability will also be undefined. Intuitively, this is because we can interpret an undefined value as an unknown value. If the state satisfied φ_2, the probability would be 1, if it did not satisfy φ_2, it would be 0 instead. Since the satisfaction is undefined, we do not know which is the case and cannot assign a clear probability. This case is covered in line 13.

$holds_{s,\varphi_1}$	$holds_{s,\varphi_2}$	$prob_{s,\varphi}$	$def_{s,\varphi}$
*	1	1	`true`
0	0	0	`true`
0	$\bot$	-	`false`
$\bot$	0	0	`true`
$\bot$	0	> 0	`false`
$\bot$	$\bot$	*	`false`
1	0	*	`true`
1	$\bot$	1	`true`
1	$\bot$	< 1	`false`

Table 3.1: Overview over definedness of until formulas

The second part of the table depicts the cases where the left side of the implication in line 15 will be satisfied for one combination of actions, which means that the probability will be calculated from the probabilities of the successor states, in the same way as we have explained in the original algorithm. A special case not mentioned in the table is that if one of the successor states has an undefined probability, the probability will always be undefined. The second part of the table therefore only refers to the case where all successors have a defined probability. If s does not satisfy φ_2 and the satisfaction of φ_1 in s is undefined, the probability will be defined only if it is calculated to be 0 from the successors. If the state did not satisfy φ_1, the probability would be 0, but if it did, the probability would be derived from the successors. Since it

Algorithm 10: SMT encoding for the reward of bounded until formulas

Input: $\mathcal{M} = (S, Act, \mathbf{P}, \mathsf{AP}, L, \mathbf{R})$: MDPR; φ: HyperPCTL bounded until formula of the form $\mathcal{R}_{\hat{s}_i}(\varphi_1 \mathcal{U}^{[k_1,k_2]} \varphi_2)$; n: number of state variables in φ.

Output: SMT encoding of φ's reward in the n-ary self-composition of $\mathcal{M}$.

1 **Function** `RewardBoundedUntil`$(\mathcal{M}, \varphi = \mathcal{R}_{\hat{s}_i}(\varphi_1 \mathcal{U}^{[k_1,k_2]} \varphi_2), n)$**:**

2 $\quad \varphi' := \mathbb{P}(\varphi_1 \mathcal{U}^{[k_1,k_2]} \varphi_2)$;

3 $\quad$ **if** $k_2 = 0$ **then**

4 $\quad\quad E :=$ `true`;

5 $\quad\quad$ **foreach** $s = (s_1, \ldots, s_n) \in S^n$ **do**

6 $\quad\quad\quad E := E \wedge (holds_{s,\varphi_2} = 1 \rightarrow (rew_{s,\varphi} = \mathbf{R}(s_i) \wedge def_{s,\varphi}))$;

7 $\quad\quad\quad E := E \wedge (holds_{s,\varphi_2} \neq 1 \rightarrow \neg def_{s,\varphi})$;

8 $\quad$ **else if** $k_1 = 0$ **then**

9 $\quad\quad E :=$ `RewardBoundedUntil`$(\mathcal{M}, \varphi = \mathcal{R}_{\hat{s}_i}(\varphi_1 \mathcal{U}^{[0,k_2-1]} \varphi_2), n)$;

10 $\quad\quad$ **foreach** $s = (s_1, \ldots, s_n) \in S^n$ **do**

11 $\quad\quad\quad E := E \wedge (holds_{s,\varphi_2} = 1 \rightarrow (rew_{s,\varphi} = \mathbf{R}(s_i) \wedge def_{s,\varphi}))$;

12 $\quad\quad\quad E := E \wedge ((prob_{s,\varphi'} \neq 1 \vee \neg def_{s,\varphi'}) \rightarrow \neg def_{s,\varphi})$;

13 $\quad\quad\quad$ **foreach** $\alpha = (\alpha_1, \ldots, \alpha_n) \in Act(s_1) \times \ldots \times Act(s_n)$ **do**

14 $\quad\quad\quad\quad E := E \wedge ((prob_{s,\varphi'} = 1 \wedge def_{s,\varphi'} \wedge holds_{s,\varphi_2} \neq 1 \wedge \bigwedge_{j=1}^{n} \sigma_{s_j} = \alpha_j) \rightarrow$

15 $\quad\quad\quad\quad [rew_{s,\varphi} = \mathbf{R}(s_i) + \sum_{s' \in supp(\alpha_1) \times \ldots \times supp(\alpha_n)} ((\prod_{j=1}^{n} P(s_j, \alpha_j, s'_j)) \cdot rew_{s', \mathcal{R}_{\hat{s}_i}(\varphi_1 \mathcal{U}^{[0,k_2-1]} \varphi_2)}) \wedge$

16 $\quad\quad\quad\quad (\neg def_{s,\varphi} \leftrightarrow [(\bigvee_{s' \in supp(\alpha_1) \times \ldots \times supp(\alpha_n)} \neg def_{s', \mathbb{P}(\varphi_1 \mathcal{U}^{[0,k_2-1]} \varphi_2)}) \vee$

17 $\quad\quad\quad\quad (holds_{s,\varphi_2} = \bot \wedge rew_{s,\varphi} \neq \mathbf{R}(s_i))])])$;

18 $\quad$ **else if** $k_1 > 0$ **then**

19 $\quad\quad E :=$ `RewardBoundedUntil`$(\mathcal{M}, \varphi = \mathcal{R}_{\hat{s}_i}(\varphi_1 \mathcal{U}^{[k_1-1,k_2-1]} \varphi_2), n)$;

20 $\quad\quad$ **foreach** $s = (s_1, \ldots, s_n) \in S^n$ **do**

21 $\quad\quad\quad E := E \wedge ((prob_{s,\varphi'} \neq 1 \vee \neg def_{s,\varphi'}) \rightarrow \neg def_{s,\varphi})$;

22 $\quad\quad\quad$ **foreach** $\alpha = (\alpha_1, \ldots, \alpha_n) \in Act(s_1) \times \ldots \times Act(s_n)$ **do**

23 $\quad\quad\quad\quad E := E \wedge ((prob_{s,\varphi'} = 1 \wedge def_{s,\varphi'} \wedge \bigwedge_{j=1}^{n} \sigma_{s_j} = \alpha_j) \rightarrow$

24 $\quad\quad\quad\quad [rew_{s,\varphi} = \mathbf{R}(s_i) + \sum_{s' \in supp(\alpha_1) \times \ldots \times supp(\alpha_n)} ((\prod_{j=1}^{n} P(s_j, \alpha_j, s'_j)) \cdot rew_{s', \mathcal{R}_{\hat{s}_i}(\varphi_1 \mathcal{U}^{[k_1-1,k_2-1]} \varphi_2)}) \wedge$

25 $\quad\quad\quad\quad ((\bigvee_{s' \in supp(\alpha_1) \times \ldots \times supp(\alpha_n)} (\neg def_{s',\varphi})) \leftrightarrow \neg def_{s,\varphi})])$;

26 $\quad$ **return** E;

is undefined and we do not know which is the case, we can only define the probability clearly if both of these values are the same, i.e. 0. In the algorithm, this condition is encoded in line 19. If $holds_{s,\varphi_1}$ and $holds_{s,\varphi_2}$ are both undefined, the probability is always undefined (line 18). If the state satisfies φ_1, but not φ_2, the probability is defined. Finally, if s satisfies φ_1, but the satisfaction of φ_2 is undefined, the probability will only be defined if the calculation equals 1. The reasoning is similar as before: If the state satisfied φ_2, the probability would be 1, otherwise it would be calculated from the successors. Since we do not know which is the case, we can only define the

probability clearly if both values are the same, i.e. 1. This case is handled in line 20.

The last case is the one where k_1 and k_2 are both greater than 0. This case is essentially a simplified version of the previous case: Since the satisfaction of φ_2 will not satisfy the formula unless k_1 is 0, we can act as though φ_2 is always unsatisfied and simplify the formula appropriately, which results in lines 21-30.

Algorithm 10 encodes the semantics for the reward of bounded until formulas. Structurally, this function is similar to Algorithm 9. For the case where $k_2 = 0$, the reward is defined and is the reward of the ith component of s if s satisfies φ_2, otherwise it is undefined.

The case where $k_1 = 0$, but $k_2 > 0$ is once again more complicated. If s satisfies φ_2, the reward is defined and is the reward of the ith component of s, just like in the previous case. Naturally, if the probability of satisfying the until formula is lower than 1 or undefined, the reward will also be undefined. Furthermore, the reward will also be undefined if the satisfaction of φ_2 in s is undefined, even if the probability of the until formula is still 1. Figure 3.2 exemplifies why this is the case. Assume both states satisfy φ_1 and the value in the nodes is the relevant state reward. Evidently, the probability of satisfying φ_2 eventually is 1. However, if s satisfied φ_2, $val_{s,\varphi}$ would be equal to its state reward, 1. If s did not satisfy φ_2, the reward would instead be equal to the sum of the state rewards of s and s', which is 2. Therefore, it is necessary for $holds_{s,\varphi_2}$ to be defined to have a clear

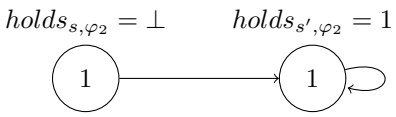

Figure 3.2: An example for undefined rewards.

Algorithm 11: SMT encoding for the meaning of unbounded until formulas

Input: $\mathcal{M} = (S, Act, \mathbf{P}, \mathsf{AP}, L, \mathbf{R})$: MDPR; φ: HyperPCTL unbounded until formula of the form $\mathbb{P}(\varphi_1 \mathcal{U} \varphi_2)$; n: number of state variables in φ.

Output: SMT encoding of φ's meaning in the n-ary self-composition of $\mathcal{M}$.

1 **Function** `SemanticsUnboundedUntil`$(\mathcal{M}, \varphi = \mathbb{P}(\varphi_1 \mathcal{U} \varphi_2), n)$**:**

2 $E := \texttt{Semantics}(\mathcal{M}, \varphi_1, n) \wedge \texttt{Semantics}(\mathcal{M}, \varphi_2, n)$;

3 **foreach** $s = (s_1, \ldots, s_n) \in S^n$ **do**

4 $E := E \wedge (holds_{s,\varphi_2} = 1 \rightarrow (prob_{s,\varphi} = 1 \wedge def_{s,\varphi}))$;

5 $E := E \wedge ((holds_{s,\varphi_1} = 0 \wedge holds_{s,\varphi_2} = 0) \rightarrow (prob_{s,\varphi} = 0 \wedge def_{s,\varphi}))$;

6 $E := E \wedge ((holds_{s,\varphi_1} = 0 \wedge holds_{s,\varphi_2} = \bot) \rightarrow \neg def_{s,\varphi})$;

7 **foreach** $\alpha = (\alpha_1, \ldots, \alpha_n) \in Act(s_1) \times \ldots \times Act(s_n)$ **do**

8 $E := E \wedge ((holds_{s,\varphi_1} \neq 0 \wedge holds_{s,\varphi_2} \neq 1 \wedge \bigwedge_{i=1}^{n} \sigma_{s_i} = \alpha_i) \rightarrow$

9 $[prob_{s,\varphi} = \sum_{s' \in supp(\alpha_1) \times \ldots \times supp(\alpha_n)} ((\prod_{i=1}^{n} P(s_i, \alpha_i, s_i')) \cdot prob_{s',\varphi}) \wedge$

10 $(prob_{s,\varphi} > 0 \rightarrow (\bigvee_{s' \in supp(\alpha_1) \times \ldots \times supp(\alpha_n)} (holds_{s',\varphi_2} = 1 \vee d_{s,\varphi_2} > d_{s',\varphi_2}))) \wedge$

11 $(\neg def_{s,\varphi} \leftrightarrow [(\bigvee_{s' \in supp(\alpha_1) \times \ldots \times supp(\alpha_n)} \neg def_{s',\varphi}) \vee$

12 $(holds_{s,\varphi_1} = \bot \wedge holds_{s,\varphi_2} = \bot) \vee$

13 $(holds_{s,\varphi_1} = \bot \wedge holds_{s,\varphi_2} = 0 \wedge prob_{s,\varphi} \neq 0) \vee$

14 $(holds_{s,\varphi_1} = 1 \wedge holds_{s,\varphi_2} = \bot \wedge prob_{s,\varphi} \neq 1)])])$;

15 **return** E;

Algorithm 12: SMT encoding for the reward of unbounded until formulas

Input: $\mathcal{M} = (S, Act, \mathbf{P}, \mathsf{AP}, L, \mathbf{R})$: MDPR; φ: HyperPCTL unbounded until formula of the form $\mathcal{R}_{\hat{s}_i}(\varphi_1 \mathcal{U} \varphi_2)$; n: number of state variables in φ.
Output: SMT encoding of φ's meaning in the n-ary self-composition of $\mathcal{M}$.

Function `RewardUnboundedUntil` $(\mathcal{M}, \varphi = \mathcal{R}_{\hat{s}_i}(\varphi_1 \mathcal{U} \varphi_2), n)$**:**
  $\varphi' := \mathbb{P}(\varphi_1 \,\mathcal{U}\, \varphi_2)$;
  $E :=$ `true`;
  foreach $s = (s_1, \ldots, s_n) \in S^n$ **do**
    $E := E \wedge (holds_{s,\varphi_2} = 1 \rightarrow (rew_{s,\varphi} = \mathbf{R}(s_i) \wedge def_{s,\varphi}))$;
    $E := E \wedge ((prob_{s,\varphi'} \neq 1 \vee \neg def_{s,\varphi'}) \rightarrow \neg def_{s,\varphi})$;
    foreach $\alpha = (\alpha_1, \ldots, \alpha_n) \in Act(s_1) \times \ldots \times Act(s_n)$ **do**
      $E := E \wedge ((prob_{s,\varphi'} = 1 \wedge def_{s,\varphi'} \wedge holds_{s,\varphi_2} \neq 1 \wedge \bigwedge_{j=1}^{n} \sigma_{s_j} = \alpha_j) \rightarrow$
      $[rew_{s,\varphi} =$
       $\mathbf{R}(s_i) + \sum_{s' \in supp(\alpha_1) \times \ldots \times supp(\alpha_n)} ((\prod_{i=1}^{n} P(s_j, \alpha_j, s'_j)) \cdot rew_{s',\varphi}) \wedge$
      $(\neg def_{s,\varphi} \leftrightarrow [(\bigvee_{s' \in supp(\alpha_1) \times \ldots \times supp(\alpha_n)} \neg def_{s',\varphi}) \vee$
      $(holds_{s,\varphi_2} = \bot \wedge rew_{s,\varphi} \neq \mathbf{R}(s_i))])])$;
  return E;

reward value. In any case other than those just mentioned, the expected reward of φ in s will be the sum of the reward of the ith component of s and the expected value of the reward for reaching a state satisfying φ_2 within $k_2 - 1$ steps from the successors of s.

Like in the semantics for the probability formula, the case where $k_1 > 0$ is essentially a simplified version of the previous case where we can assume φ_2 to be unsatisfied in s.

Algorithm 11 encodes the semantics of an unbounded until formula. Like in the

Algorithm 13: SMT encoding for the cumulative reward after t steps

Input: $\mathcal{M} = (S, Act, \mathbf{P}, \mathsf{AP}, L, \mathbf{R})$: MDPR; φ: HyperPCTL cumulative reward formula of the form $\mathcal{R}_{\hat{s}_i}(C_t)$; n: number of state variables in φ.
Output: SMT encoding of φ's meaning in the n-ary self-composition of $\mathcal{M}$.

Function `CumulativeReward` $(\mathcal{M}, \varphi = \mathcal{R}_{\hat{s}_i}(C_t), n)$**:**
  $E :=$ `true`
  foreach $s = (s_1, \ldots, s_n) \in S^n$ **do**
    $E := E \wedge (def_{s,\mathcal{R}_{\hat{s}_i}(C_0)} \wedge rew_{s,\mathcal{R}_{\hat{s}_i}(C_0)} = \mathbf{R}(s_i))$;
  foreach $k \in \{1, \ldots, t\}$ **do**
    foreach $s = (s_1, \ldots, s_n) \in S^n$ **do**
      $E := E \wedge def_{s,\mathcal{R}_{\hat{s}_i}(C_k)}$;
      foreach $\alpha = (\alpha_1, \ldots, \alpha_n) \in Act(s_1) \times \ldots \times Act(s_n)$ **do**
        $E := E \wedge ([\bigwedge_{j=1}^{n} \sigma_{s_j} = \alpha_j] \rightarrow rew_{s,\mathcal{R}_{\hat{s}_i}(C_k)} = \mathbf{R}(s_i) +$
         $\sum_{s' \in succ(s_1) \times \ldots \times succ(s_n)} ((\prod_{j=1}^{n} P(s_j, s'_j)) \cdot rew_{s,\mathcal{R}_{\hat{s}_i}(C_{k-1})}))$;
  return E;

original algorithm, the semantics of an unbounded until formula are very similar to those of a bounded until formula for the case where $k_1 = 0$ and $k_2 > 0$. The main difference is once again the addition of line 10 which enforces that positive probabilities can eventually be traced back to states that satisfy φ_2.

Algorithm 14: SMT encoding for the instantaneous reward after t steps

Input: $\mathcal{M} = (S, Act, \mathbf{P}, \mathsf{AP}, L, \mathbf{R})$: MDPR; φ: HyperPCTL instantaneous reward formula of the form $\mathcal{R}_{\hat{s}_i}(I_t)$; n: number of state variables in φ.
Output: SMT encoding of φ's meaning in the n-ary self-composition of $\mathcal{M}$.

```
1  Function InstantaneousReward(M, φ = R_ŝi(I_t), n):
2      E := true
3      foreach s = (s_1,...,s_n) ∈ S^n do
4          E := E ∧ (def_{s,R_ŝi(I_0)} ∧ rew_{s,R_ŝi(I_0)} = R(s_i));
5      foreach k ∈ {1,...,t} do
6          foreach s = (s_1,...,s_n) ∈ S^n do
7              E := E ∧ def_{s,R_ŝi(I_k)};
8              foreach α = (α_1,...,α_n) ∈ Act(s_1) × ... × Act(s_n) do
9                  E := E ∧ ([⋀_{j=1}^n σ_{s_j} = α_j] → rew_{s,R_ŝi(I_k)} =
                       Σ_{s'∈succ(s_1)×...×succ(s_n)}((∏_{j=1}^n P(s_j,s'_j)) · rew_{s,R_ŝi(I_{k-1})}));
10     return E;
```

As was the case for the equivalent probability expression, the encoding for the reward of unbounded until formulas seen in Algorithm 12 is very similar to the encoding for bounded until formulas for the case $k_1 = 0$ and $k_2 > 0$.

Algorithm 13 shows the encoding for the cumulative reward within t steps. Since this expression has been defined as syntactic sugar, it is not actually necessary to encode it separately. However, it is a much simpler case than the equivalent until formula, so adding a separate encoding for this expression can reduce the number of variables and subformulas generated and improve the running time of the algorithm. We simply sequentially define the cumulative reward for each step count starting from 0 until we have reached t. For 0 steps, the reward is simply the current state reward (line 4). For k steps with $k > 0$, the reward is simply the current state reward added to the expected cumulative reward within $k - 1$ steps of the successors (line 9). This

Algorithm 15: SMT encoding of the truth of the input formula

Input: $\mathcal{M} = (S, Act, \mathbf{P}, \mathsf{AP}, L, \mathbf{R})$: MDPR; $Q_1\hat{s}_1 \ldots Q_n\hat{s}_n.\varphi^{nq}$: HyperPCTL formula.
Output: Encoding of the truth of the input formula in $\mathcal{M}$.

```
1  Function Truth(M, ∃σ̂.Q_1ŝ_1 ... Q_nŝ_n.φ^nq):
2      foreach i = 1,...,n do
3          if Q_i = ∀ then B_i := " ⋀_{s_i∈S} ";
4          else B_i := " ⋁_{s_i∈S} ";
5      return B_1 ... B_n (holds_{(s_1,...,s_n),φ^nq} = 1);
```

reward is always defined, which makes this case comparatively simple.

Algorithm 14 shows the encoding for the instantaneous reward after t steps. This algorithm is almost identical to Algorithm 13. The only difference is that the instantaneous reward after $k > 0$ steps is simply identical to the expected instantaneous reward after $k - 1$ steps of the successors, without the addition of the current state reward (line 9).

Algorithm 15 encodes the meaning of the state quantifiers for the SMT formula that will be satisfiable exactly if $\mathcal{M}$ satisfies the input formula. There are no major changes to the original version presented in Chapter 2.

Algorithm 16: SMT encoding of the input formula not being definitely false

Input: $\mathcal{M} = (S, Act, \mathbf{P}, \mathsf{AP}, L, \mathbf{R})$: MDPR; $Q_1\hat{s}_1 \dots Q_n\hat{s}_n.\varphi^{nq}$: HyperPCTL formula.

Output: Encoding of the input formula not being definitely false in $\mathcal{M}$.

1 **Function** `NotFalsehood`$(\mathcal{M}, \exists\hat{\sigma}.Q_1\hat{s}_1 \dots Q_n\hat{s}_n.\varphi^{nq})$**:**
2 **foreach** $i = 1, \dots, n$ **do**
3 **if** $Q_i = \forall$ **then** $B_i :=$ " $\bigwedge_{s_i \in S}$ ";
4 **else** $B_i :=$ " $\bigvee_{s_i \in S}$ ";
5 **return** $B_1 \dots B_n$ $(holds_{(s_1,\dots,s_n),\varphi^{nq}} \neq 0)$;

Algorithm 16 encodes the meaning of the state quantifiers for the SMT formula that will be satisfiable exactly if $\mathcal{M}$ satisfies the input formula or the satisfaction is undefined. The only difference to the previous function is that $holds_{(s_1,\dots,s_n),\varphi^{nq}} = 1$ has been replaced with $holds_{(s_1,\dots,s_n),\varphi^{nq}} \neq 0$, which means that it is not necessary for the states to satisfy the formula, but that it is also fine if the satisfaction is undefined instead. If the first generated SMT formula is satisfiable, the MDP $\mathcal{M}$ satisfies the input formula, if the first formula is unsatisfiable, but the second one is satisfiable, the satisfaction of the input formula for $\mathcal{M}$ is undefined, and if both formulas are unsatisfiable, $\mathcal{M}$ does not satisfy the input formula.

Chapter 4

Case Studies

In this chapter, we will present two case studies to exemplify possible applications of HyperPCTL with rewards. These case studies are based on those presented in [ÁBBD20], modified to make use of the new reward property added to HyperPCTL.

4.1 Timing Attack

The security of cryptographic systems is generally based on the secrecy of certain information, like encryption keys. If an attacker was able to access this secret information, the security of the system would be compromised. Therefore, special care must be taken to ensure that the cryptographic algorithm does not leak any information about the secret. One way this could happen is if the execution time of the algorithm is different depending on the value of the secret. As an example, RSA uses the modular exponentiation algorithm to compute $a^b \mod n$, where a is the message and b is the encryption key. An implementation of this algorithm is shown in Figure 4.1. This implementation is flawed, however: The *if* in line 6, which checks whether the current bit of b is 0 or 1 does not have an *else* branch, so the execution of the algorithm will take longer if b contains more 1-bits. An attacker could therefore measure the execution time of the algorithm to derive the number of 1-bits in the encryption key. This is known as a *timing attack*. To prevent such vulnerabilities, we would like the execution time to be independent of the encryption key. We can represent this program in an MDPR by letting states represent the current position in the code as well as the current loop iteration. We represent the encryption key b by letting the two branches of the *if*-statement be entered with two nondeterministic actions. Thus, the value of the encryption key b will correspond to the chosen scheduler. The addition of rewards allows us to model execution time fairly easily: If we assign every state a

```
1  void mexp(){
2    c = 0; d = 1; i = k;
3    while (i >= 0){
4      i = i-1; c = c*2;
5      d = (d*d) % n;
6      if (b(i) = 1)
7        c = c+1;
8        d = (d*a) % n;
9      }
10 }
```

Figure 4.1: Modular exponentiation algorithm

constant reward of 1, the reward on two paths from the initial state to the end state will be the same if and only if the paths contain the same number of states and their execution time will be the same. Ideally, every encryption key should have the same execution time, so the desired property is for the algorithm to satisfy the following HyperPCTL formula:

$$\forall\hat{\sigma}_1.\forall\hat{\sigma}_2.\forall\hat{s}(\hat{\sigma}_1).\forall\hat{s}'(\hat{\sigma}_2).\,(init_{\hat{s}} \wedge init_{\hat{s}'}) \rightarrow (\mathcal{R}_{\hat{s}}(\Diamond end_{\hat{s}}) = \mathcal{R}_{\hat{s}'}(\Diamond end_{\hat{s}'}))$$

If the model satisfies the formula, any two schedulers, i.e. encryption keys will have the same reward, i.e. execution time for the algorithm. If it does not satisfy the formula, we can derive encryption keys with differing execution times from a non-satisfying pair of schedulers.

4.2 Probabilistic Conformance

We will consider the example of a 6-sided die. We are interested in simulating the behavior of this die by repeatedly throwing a coin. It is possible to model this problem with MDPs and HyperPCTL such that a satisfying scheduler for a certain formula describes an algorithm for simulating the 6-sided die with repeated coin tosses. The first part of the model will have an initial state and six absorbing states, each of which represents one result of the die and is entered with probability 1/6. This part models the die. The second part, which models the coin-tossing, includes six absorbing states for the outcomes as well as a given maximal number of states. For each choice of two successor states, a state will have an action that transitions into each of these states with probability 1/2 to model a coin toss. The choice of scheduler therefore decides which two states each state will transition into and this way defines an implementation. The following formula describes if the two parts of the model conform to each other probabilistically:

$$\exists\hat{\sigma}.\forall\hat{s}(\hat{\sigma}).\exists\hat{s}'(\hat{\sigma}).dieinit_{\hat{s}} \rightarrow \left(coininit_{\hat{s}'} \wedge \bigwedge_{l=1}^{6} (\mathbb{P}(\Diamond(die = l)_{\hat{s}}) = \mathbb{P}(\Diamond(die = l)_{\hat{s}'})) \right)$$

A scheduler for which this formula is satisfied describes a coin-implementation such that the probability of each result is the same as in the die model. Extending this model with rewards also allows us to synthesize efficient implementations: If we give every state except the absorbing states a reward of 1, the expected reward until reaching one of the absorbing states will be equal to the expected number of coin tosses in that implementation. If we add the condition that the reward must be below a certain value x, the formula will only be satisfied by a scheduler corresponding to an algorithm where the expected number of coin tosses is below x. The following formula for instance specifies that the expected number of coin tosses in the implementation must be less than 4:

$$\exists\hat{\sigma}.\forall\hat{s}(\hat{\sigma}).\exists\hat{s}'(\hat{\sigma}).dieinit_{\hat{s}} \rightarrow \left(\varphi \wedge \mathcal{R}_{\hat{s}'}(\Diamond(\bigvee_{l=1}^{6}(die = l)_{\hat{s}'})) < 4 \right)$$

with $\varphi = coininit_{\hat{s}'} \wedge \bigwedge_{l=1}^{6} (\mathbb{P}(\Diamond(die = l)_{\hat{s}}) = \mathbb{P}(\Diamond(die = l)_{\hat{s}'}))$.

Experiment	Encoding time(s)	Solving time(s)	Total time(s)	SMT vars	Sub-formulas	States	Tran-sitions
1-bit TA	0.11	0.01	0.12	344	1008	8	10
2-bit TA	0.22	0.01	0.23	708	2016	12	16
3-bit TA	0.38	0.03	0.41	1200	3344	16	22
4-bit TA	0.61	0.06	0.67	1820	4992	20	28
1-state PC	5.03	2.03	7.06	7281	34681	20	186
3-state PC	6.66	8.91	15.57	7281	61631	20	494
5-state PC	8.82	35.0	43.82	7281	88581	20	802
7-state PC	11.64	53.05	64.69	7281	115531	20	1110

Table 4.1: Results of the experiments. TA is Timing Attack, PC is Probabilistic Conformance.

4.3 Implementation

We have modified the implementation of the algorithm presented in Chapter 2 that was mentioned in [ÁBBD20] to support reward properties. The implementation was written in Python and makes use of the libraries Lark [Lar] for parsing the input formula and Stormpy [Sto] for storing the input MDP. The generated formula is solved with the SMT solver Z3 [MB08]. The extension of the algorithm is not complete, as it cannot handle undefined values correctly yet, but the case studies have been designed such that this does not pose a problem.

For the timing attack case study, we have modeled the problem with 1, 2, 3 and 4 bit encryption keys. Since the HyperPCTL formula for the property has two scheduler quantifiers while the algorithm was only defined for one algorithm, we have added a second copy of the model to the input MDPR such that the single scheduler can assign different actions to the states in the two copies of the model.

For the case study on probabilistic conformance, we have limited number of states to 7. The first experiment fixed the actions for 6 of those states and only allowed the scheduler free choice for the first state. Each successive experiment gave the scheduler a choice of actions in two additional states until the final experiment, which allows any implementation with 7 states. We have limited the expected number of coin tosses to 4.

Table 4.1 lists the results of the experiments. The experiments have been performed in a Docker container running on a laptop with a 2.80 GHz i7 CPU and 16 GB of RAM. Because of the incomplete implementation of undefined values, which would add a significant number of additional subformulas and variables, these values are lower than they would normally be. The simple time attack example can be solved in a very short time, but we can already see that more complicated examples with many actions will have very long running times.

Chapter 5

Conclusion

5.1 Summary

In this thesis, we have discussed hyperproperties with rewards. First, we have introduced important preliminary concepts such as DTMCs as a model for probabilistic systems as well as MDPs, which allow nondeterminism. This nondeterminism can be eliminated with the use of schedulers to induce DTMCs in an MDP. We have also introduced the temporal logic HyperPCTL which was designed to reason about hyperproperties in MDPs. We have also introduced a model checking algorithm for HyperPCTL restricted to non-probabilistic, memoryless schedulers.

After that, we have added reward properties: We have extended the DTMC and MDP models with state rewards so that we could add a reward operator to HyperPCTL and reason about the reward of paths in the MDPR. We have also modified the original model checking algorithm to support the new reward property as well as undefined values.

Finally, we have presented two case studies to exemplify possible uses of HyperPCTL with rewards. The timing attack case study showed that rewards make it very easy to model certain hyperproperties, while the case study on probabilistic conformance gave an example of using rewards to add performance constraints to models.

5.2 Discussion

As the case studies show, rewards can be very useful to easily model certain properties like the execution time of a program. Other use cases for rewards could include energy consumption of systems or path costs in robotics. We can conclude that the extension of HyperPCTL with rewards has a number of practical applications. However, we note that the undefined values that can occur due to the addition of rewards increase the complexity of the algorithm by adding a number of variables and constraints. This means that the running time of the model checking algorithm could increase significantly, even in formulas where no rewards occur. It is therefore likely preferable to use the original algorithm if rewards are not needed.

5.3 Future work

There are several possibilities for future work. We have only discussed state rewards in this thesis. It would be possible to also assign rewards to the transitions of an MDP and extend the Semantics of HyperPCTL to include these transition rewards. Naturally, this would also require further modifications to the model checking algorithm.

Another possibility concerns the handling of undefined values: Consider Figure 5.1. With the current implementation, the probability of finally reaching a state that satisfies φ from s is undefined. The formula $\mathbb{P}(\Diamond\varphi) \geq 0.5$ would have an undefined value in s_0. However, we can see that one of the two successors of s_0 satisfies φ. Therefore, regardless of the satisfaction of φ in s_2, the probability in s_0 would always be at least 0.5. A possibility for future work would be to extend the algorithm such that we give upper and lower boundaries for undefined arithmetic values. That way, we could assign clear satisfaction values to formulas that have undefined values in the current implementation.

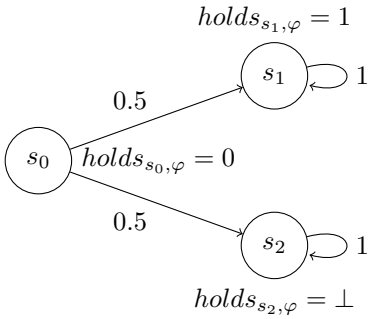

Figure 5.1: Example for undefined values.

Bibliography

[ÁB18] Erika Ábrahám and Borzoo Bonakdarpour. HyperPCTL: A temporal logic for probabilistic hyperproperties. In Annabelle McIver and Andras Horvath, editors, *Quantitative Evaluation of Systems*, pages 20–35, Cham, 2018. Springer International Publishing.

[ÁBBD20] Erika Ábrahám, Ezio Bartocci, Borzoo Bonakdarpour, and Oyendrila Dobe. Probabilistic hyperproperties with nondeterminism. In Dang Van Hung and Oleg Sokolsky, editors, *Automated Technology for Verification and Analysis*, pages 518–534, Cham, 2020. Springer International Publishing.

[CS08] Michael R. Clarkson and Fred B. Schneider. Hyperproperties. In *2008 21st IEEE Computer Security Foundations Symposium*, pages 51–65, 2008.

[GM82] J. A. Goguen and J. Meseguer. Security policies and security models. In *1982 IEEE Symposium on Security and Privacy*, pages 11–11, 1982.

[HJ95] Hans Hansson and Bengt Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6, 02 1995.

[KNP11] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In G. Gopalakrishnan and S. Qadeer, editors, *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

[Lar] Lark. `https://lark-parser.readthedocs.io`. [Accessed 30-July-2021].

[MB08] Leonardo De Moura and Nikolaj Bjørner. Z3: an efficient SMT solver. In *TACAS'08/ETAPS'08 Proceedings of the Theory and practice of software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340, 2008.

[Sto] Stormpy. `https://moves-rwth.github.io/stormpy/`. [Accessed 26-July-2021].