Model Checking Hyperproperties for Markov Decision Processes

Oyendrila Dobe[a], Erika Ábrahám[b], Ezio Bartocci[c], Borzoo Bonakdarpour[a]

[a]*Michigan State University, USA,*
[b]*RWTH Aachen University, Germany,*
[c]*Technische Universität Wien, Austria.*

Abstract

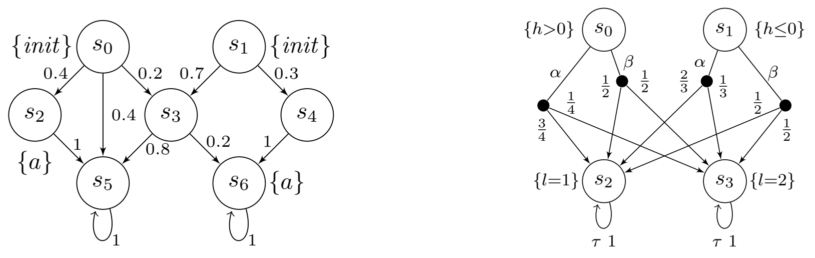
We study the problem of formalizing and checking *probabilistic hyperproperties* for Markov decision processes (MDPs). We introduce the temporal logic HyperPCTL that allows explicit and simultaneous quantification over schedulers as well as probabilistic computation trees. We show that the logic can express important quantitative requirements in security and privacy such as probabilistic noninterference, differential privacy, timing side-channel countermeasures, and probabilistic conformance testing. We show that HyperPCTL model checking over MDPs is in general undecidable, but restricting the domain of scheduler quantification to memoryless non-probabilistic schedulers turns the model checking problem decidable. Subsequently, we propose an SMT-based encoding for model checking this language. Finally, we demonstrate the applicability of our method by providing experimental results for verification, and we show how it can be used to solve even certain synthesis problems.

Keywords: Markov Models, Hyperproperties, Model Checking, Policy Synthesis, Information Leakage.

1. Introduction

Hyperproperties [1] extend the conventional notion of *trace properties* [2] from a set of traces to a set of sets of traces. In other words, a hyperproperty stipulates a *system* property and not the property of just individual traces. It has been shown that many interesting requirements in computing systems are hyperproperties and cannot be expressed by trace properties. Examples include (1) a wide range of information-flow security policies such as *noninterference* [3] and *observational determinism* [4], (2) sensitivity and robustness requirements in cyber-physical systems [5], and (3) consistency conditions such as *linearizability* in concurrent data structures [6].

Hyperproperties can describe the requirements of probabilistic systems as well. They generally express probabilistic relations between multiple executions of a system. For example, in information-flow security, the addition of probabilities is motivated by establishing a connection between information theory and



(a) Example DTMC.

(b) MDP with non-probabilistic actions.

Figure 1: Example Markov models.

information flow across multiple traces. A prominent example is probabilistic schedulers that open up the opportunity for an attacker to set up a probabilistic covert channel. Or, *probabilistic causation* that compares the probability of occurrence of an effect between scenarios where the cause is or is not present. Also, the goal of *quantitative information flow* is to measure the amount of information leaked about a secret by observing different runs of a program.

In [7], we made the first step and proposed the temporal logic HyperPCTL for discrete-time Markov chains (DTMCs), which extends PCTL by allowing explicit and simultaneous quantification over probabilistic computation trees. For example, the DTMC in Fig. 1a satisfies the following HyperPCTL formula:

$$\forall \hat{s}.\ \forall \hat{s}'.\ \Big(init_{\hat{s}} \wedge init_{\hat{s}'}\Big) \ \rightarrow\ \Big(\mathbb{P}(\Diamond\, a_{\hat{s}}) = \mathbb{P}(\Diamond\, a_{\hat{s}'})\Big) \tag{1}$$

which means that the probability of reaching proposition a from any pair of states $\hat{s}$ and $\hat{s}'$ labeled by *init* should be equal. Other works on probabilistic hyperproperties for DTMCs include parameter synthesis [8] and statistical model checking [9, 5].

An important gap in the spectrum is the verification of probabilistic hyperproperties with regard to models that allow *nondeterminism*, in particular, *Markov decision processes* (MDPs). Nondeterminism plays a crucial role in many probabilistic systems. For instance, nondeterministic queries can be exploited in order to make targeted attacks to databases with private information [10]. To motivate the idea, consider the MDP in Fig. 1b, where h is a high secret input and l is a low publicly observable output. To protect the secret, there should be no probabilistic dependencies between observations of the low variable l and the value of h. However, an attacker choosing a scheduler that always takes action α from states s_0 and s_1, can learn whether or not $h \leq 0$ by observing the probability of obtaining $l = 1$ (or $l = 2$). On the other hand, a scheduler that always chooses action β, does not leak any information about the value of h. Thus, a natural question to ask is whether a certain property holds for all or some schedulers.

With the above motivation, in this paper, we focus on probabilistic hyperproperties in the context of MDPs. Such hyperproperties inherently need to consider different nondeterministic choices in different executions, and naturally call for quantification over schedulers. There are several challenges to achieve this. In general, there are MDPs whose reachability probabilities cannot be achieved by any memoryless non-probabilistic scheduler, and, hence finding a scheduler is not reducible to checking non-probabilistic memoryless schedulers, as it is done in PCTL model checking for MDPs.

Consider for example the MDP in Fig. 2, for which we want to know whether there is a scheduler such that the probability to reach s_1 from s_0 equals 0.5. There are two non-probabilistic memoryless schedulers, one choosing action α and the other, action β in s_0. The first one is the maximal scheduler for which s_1 is reached with probability 1, and the second one is the minimal scheduler leading to probability 0. However, the probability 0.5 cannot be achieved by any non-probabilistic scheduler. *Memoryless* probabilistic schedulers cannot achieve probability 0.5 either : if a memoryless scheduler would take action α with any positive probability, then the probability to reach s_1 is always 1. The only way to achieve the reachability probability 0.5 (or any value strictly between 0 and 1) is by a probabilistic scheduler with memory, e.g., taking α and β in s_0 with probabilities 0.5 each when this is the first step on a path, and β with probability 1 otherwise.

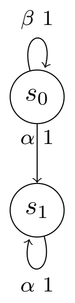


Figure 2: MDP where we cannot reach s_1 with 0.5 probability.

Our contributions in this paper are as follows. We first extend the temporal logic HyperPCTL from [7] to the context of MDPs. To this end, we augment the syntax and semantics of HyperPCTL to quantify over schedulers and relate probabilistic computation trees for different schedulers. For example, the following formula generalizes (1) by requiring that the respective property should hold for all computation trees starting in any states $\hat{s}$ and $\hat{s}'$ of the DTMC induced by any scheduler $\hat{\sigma}$:

$$\forall\hat{\sigma}(\hat{\mathcal{M}}).\ \forall\hat{s}(\hat{\mathcal{M}}^{\hat{\sigma}}).\ \forall\hat{s}'(\hat{\mathcal{M}}^{\hat{\sigma}}).\ \Big(init_{\hat{s}} \wedge init_{\hat{s}'}\Big) \rightarrow \Big(\mathbb{P}(\Diamond a_{\hat{s}}) = \mathbb{P}(\Diamond a_{\hat{s}'})\Big) \qquad (2)$$

On the negative side, we show that the problem to check HyperPCTL properties for MDPs is in general undecidable. On the positive side, we show that the problem becomes decidable when we restrict the scheduler quantification domain to memoryless non-probabilistic schedulers. We also show that this restricted problem is already NP-complete (respectively, coNP-complete) in the size of the given MDP for HyperPCTL formulas with a single existential (respectively, universal) scheduler quantifier. Subsequently, we propose an SMT-based encoding to solve the restricted model checking problem. We have implemented our method (available at https://github.com/TART-MSU/HyperProb), and used it to analyze three case studies: probabilistic scheduling attacks, side-channel timing attacks, and probabilistic conformance. We have further published a tool-paper that appeared in FM'21 [11] explaining the working of the tool.

We believe there is a deep connection between HyperPCTL model checking and verification of partially observable MPDs (POMDPs). In fact, we have successfully verified the well-known Monty-Hall problem (which is a POMDP) using our HyperPCTL solution. A similar analogy also exists between HyperLTL and distributed synthesis [12]. While an interesting problem, we leave it to future work to show whether it is possible to reduce HyperPCTL model checking to verification of POMDPs.

Comparison with the conference version. A preliminary version of this work appeared in ATVA'20 [13]. In this version, we have made several new contributions and improvements as compared to [13]. First, we have revised the syntax and semantics of HyperPCTL. The new logic is now *multi-model*, where different quantifiers can range over different Markov models. This feature allows reasoning about computations of different models more elegantly. Furthermore, we present the detailed proofs of our undecidability and complexity results. Perhaps, most importantly, our SMT-based model checking algorithm is now significantly more efficient. The average improvement of our algorithm is an order of magnitude. This allowed us to verify larger models for a verification problem that is computationally intractable.

Organization. Preliminary concepts are discussed in Section 2. We present the syntax and semantics of HyperPCTL for MDPs and discuss its applications in Sections 3 and 4, respectively. Section 5 studies the expressive power of HyperPCTL. Sections 6 and 7 present our model checking algorithm for memoryless non-probabilistic schedulers and its evaluation, respectively. Related work is discussed in Section 8 before concluding in Section 9.

2. Preliminaries

In this section, we recall the modeling formalisms of discrete-time Markov chains (DTMCs) in Section 2.1 and Markov decision processes in Section 2.2, and introduce some formalisms which we will need in later sections to define our HyperPCTL logic.

2.1. Discrete-Time Markov Chains

Definition 1. *A* discrete-time Markov chain (DTMC) *is a tuple* $\mathcal{D}=(S, \mathbf{P}, \mathsf{AP}, L)$ *specifying:*

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

We define the empty DTMC $\mathcal{D}_\emptyset = (\emptyset, \mathbf{P}_\emptyset, \emptyset, L_\emptyset)$ *with* $\mathbf{P}_\emptyset\colon \emptyset \times \emptyset \to [0,1]$ *and* $L_\emptyset\colon \emptyset \to \{\emptyset\}$ *and use* $\mathbb{D}$ *to denote the set of all DTMCs with non-empty state sets.*

In the following, whenever we refer to a DTMC, we mean a *non-empty* DTMC from $\mathbb{D}$, and explicitly name a DTMC *empty* when we mean $\mathcal{D}_\emptyset$. Fig. 1a shows the graphical illustration of a simple DTMC, where each state is represented by a node, each positive transition probability $P(s,s')>0$ by an arrow from node s to node s' decorated with the probability $P(s,s')$, and each label set $L(s)$ depicted beside the node for state s.

An (*infinite*) *path* of a DTMC $\mathcal{D}=(S,\mathbf{P},\mathsf{AP},L)\in\mathbb{D}$ is an infinite sequence $\pi=s_0s_1s_2\ldots\in S^\omega$ of states with $\mathbf{P}(s_i,s_{i+1})>0$, for all $i\geq 0$; we define $\pi[i]=s_i$. Let $\mathit{Paths}_s^{\mathcal{D}}$ denote the set of all (infinite) paths of $\mathcal{D}$ starting in s, and $\mathit{fPaths}_s^{\mathcal{D}}$ denote the set of all non-empty finite prefixes of paths from $\mathit{Paths}_s^{\mathcal{D}}$, which we call *finite paths*. For a finite path $\pi=s_0\ldots s_k\in\mathit{fPaths}_{s_0}^{\mathcal{D}}$, $k\geq 0$, we define $|\pi|=k$. We will also use the notations $\mathit{Paths}^{\mathcal{D}}=\bigcup_{s\in S}\mathit{Paths}_s^{\mathcal{D}}$ and $\mathit{fPaths}^{\mathcal{D}}=\bigcup_{s\in S}\mathit{fPaths}_s^{\mathcal{D}}$. A state $t\in S$ is *reachable* from a state $s\in S$ in $\mathcal{D}$ if there exists a finite path in $\mathit{fPaths}_s^{\mathcal{D}}$ with last state t; we use $\mathit{fPaths}_{s,T}^{\mathcal{D}}$ to denote the set of all finite paths from $\mathit{fPaths}_s^{\mathcal{D}}$ with last state in $T\subseteq S$. A state $s\in S$ is *absorbing* if $\mathbf{P}(s,s)=1$.

The *cylinder set* $\mathit{Cyl}^{\mathcal{D}}(\pi)$ of a finite path $\pi\in\mathit{fPaths}_s^{\mathcal{D}}$ is the set of all infinite paths of $\mathcal{D}$ with prefix π. The *probability space for $\mathcal{D}$ and state $s\in S$* is

$$\Big(\mathit{Paths}_s^{\mathcal{D}},\big\{\bigcup_{\pi\in R}\mathit{Cyl}^{\mathcal{D}}(\pi)\mid R\subseteq\mathit{fPaths}_s^{\mathcal{D}}\big\},\Pr_s^{\mathcal{D}}\Big)\ ,$$

where the *probability* of the cylinder set of $\pi\in\mathit{fPaths}_s^{\mathcal{D}}$ is $\Pr_s^{\mathcal{D}}(\mathit{Cyl}^{\mathcal{D}}(\pi))=\Pi_{i=1}^{|\pi|}\mathbf{P}(\pi[i-1],\pi[i])$. Note that the cylinder sets of two finite paths starting in the same state are either disjoint or one is contained in the other. According to the definition of the probability spaces, the total probability for a set of cylinder sets defined by the finite set of paths $R\subseteq\mathit{fPaths}_s^{\mathcal{D}}$ is $\Pr_s^{\mathcal{D}}(\bigcup_{\pi\in R}\mathit{Cyl}^{\mathcal{D}}(\pi))=\sum_{\pi\in R'}\Pr_s^{\mathcal{D}}(\mathit{Cyl}^{\mathcal{D}}(\pi))$ with $R'=\{\pi\in R\mid \textit{no } \pi'\in R\setminus\{\pi\} \textit{ is a prefix of } \pi\}$. To improve readability, we sometimes omit the DTMC index $\mathcal{D}$ in the notations when it is clear from the context.

The following definition of *parallel composition* of two DTMCs allows to formalize simultaneous runs in the composed models.

Definition 2. *For two DTMCs $\mathcal{D}_i=(S_i,\mathbf{P}_i,\mathsf{AP}_i,L_i)\in\mathbb{D}$, $i\in\{1,2\}$, with disjoint atomic proposition sets $\mathsf{AP}_1\cap\mathsf{AP}_2=\emptyset$, their* parallel composition *is the DTMC $\mathcal{D}_1\times\mathcal{D}_2=(S,\mathbf{P},\mathsf{AP},L)$ with the following components:*

- $S=S_1\times S_2$,
- $\mathbf{P}\colon S\times S\to[0,1]$ *with* $\mathbf{P}((s_1,s_2),(s_1',s_2'))=\mathbf{P}_1(s_1,s_1')\cdot\mathbf{P}_2(s_2,s_2')$, *for all states* $(s_1,s_2),(s_1',s_2')\in S$,
- $\mathsf{AP}=\mathsf{AP}_1\cup\mathsf{AP}_2$, *and*
- $L\colon S\to 2^{\mathsf{AP}}$ *with* $L((s_1,s_2))=L_1(s_1)\cup L_2(s_2)$. ■

For the empty DTMC, we define $\mathcal{D}_\emptyset\times\mathcal{D}_\emptyset=\mathcal{D}_\emptyset$ and $\mathcal{D}_\emptyset\times\mathcal{D}=\mathcal{D}\times\mathcal{D}_\emptyset=\mathcal{D}$ for all $\mathcal{D}\in\mathbb{D}$.

2.2. Markov Decision Processes

Markov decision processes extend DTMCs with non-deterministic action choices.

Definition 3. *A* Markov decision process *(*MDP*) is a tuple* $\mathcal{M} = (S, Act, \mathbf{P}, \mathsf{AP}, L)$ *with*

- *a finite set* S *of* states*,*
- *a non-empty finite set* Act *of* actions*,*
- *a* transition probability function $\mathbf{P}\colon S \times Act \times S \to [0,1]$*, such that for all* $s \in S$ *the set*
$$Act(s) = \Big\{\alpha \in Act \mid \sum_{s' \in S} \mathbf{P}(s, \alpha, s') = 1\Big\}$$
of enabled actions in s *is not empty and* $\sum_{s' \in S} \mathbf{P}(s, \alpha, s') = 0$ *for all* $\alpha \in Act \setminus Act(s)$*,*
- *a finite set* AP *of* atomic propositions*, and*
- *a* labeling function $L\colon S \to 2^{\mathsf{AP}}$. ■

We use $\mathbb{M}$ *to denote the set of all MDPs with non-empty state sets.*

Fig. 1b illustrates a simple MDP. *Schedulers* can be used to eliminate the non-determinism in MDPs, inducing DTMCs with well-defined probability spaces. A scheduler eliminates non-determinism by deciding which action to take at which point of execution. *Memoryless* schedulers make these decisions based only on the current MDP state, whereas the decisions of schedulers *with memory* might also depend on the executions leading to the current state.

In the definition below, observations about the execution are stored in the mode of the scheduler, which is updated for each step by the *mode* function. When defining schedulers formally, one option used e.g., in [14] is to use the full execution for decision making. We decided to make this notion more abstract and allow the scheduler mode to store arbitrary observations about the execution. Allowing a single mode implies the memoryless feature. A finite set of modes allows to store observations, but they still induce finite DTMC models. Allowing infinite memory, needed e.g., to store the full execution paths, might induce infinite-space DTMCs.

The action to be taken is chosen by the *act* function, dependent on the current MDP state and the current scheduler mode. If the scheduler chooses a unique action in each state-mode pair then we call it *non-probabilistic*, otherwise if the scheduler makes a random choice over the available actions then we call it *probabilistic*.

Definition 4. *A* scheduler *for an MDP* $\mathcal{M} = (S, Act, \mathbf{P}, \mathsf{AP}, L)$ *is a tuple* $\sigma = (Q, act, mode, init)$ *where*

- Q *is a countable set of* modes*,*
- $act\colon Q \times S \times Act \to [0,1]$ *is a function such that for all* $s \in S$ *and* $q \in Q$ *it holds that*
$$\sum_{\alpha \in Act(s)} act(q, s, \alpha) = 1 \quad \text{and} \quad \sum_{\alpha \in Act \setminus Act(s)} act(q, s, \alpha) = 0,$$

- *$mode\colon Q \times S \to Q$ is a* mode transition *function, and*
- *$init\colon S \to Q$ is a function selecting a starting mode for state s of $\mathcal{M}$.* ■

Let $\Sigma^{\mathcal{M}}$ denote the set of all schedulers for an MDP $\mathcal{M}$. 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 $q \in Q$, $s \in S$ and $\alpha \in Act$.

Definition 5. *Assume an MDP $\mathcal{M} = (S, Act, \mathbf{P}, \mathsf{AP}, L)$ and a scheduler $\sigma = (Q, act, mode, init) \in \Sigma^{\mathcal{M}}$ 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 $S^{\sigma} = Q \times S$,*

$$\mathbf{P}^{\sigma}((q,s),(q',s')) = \begin{cases} \sum_{\alpha \in Act(s)} act(q,s,\alpha) \cdot \mathbf{P}(s,\alpha,s') & \text{if } q' = mode(q,s) \\ 0 & \text{otherwise} \end{cases}$$

and $L^{\sigma}(q, s) = L(s)$ for all $s, s' \in S$ and for all $q, q' \in Q$. ■

A state s' is *reachable* from $s \in S$ in MDP $\mathcal{M}$ if there exists a scheduler σ for $\mathcal{M}$ such that s' is reachable from s in $\mathcal{M}^{\sigma}$. A state $s \in S$ is *absorbing* in $\mathcal{M}$ if s is absorbing in $\mathcal{M}^{\sigma}$ for all schedulers σ for $\mathcal{M}$.

3. The Temporal Logic HyperPCTL

We now define the temporal logic HyperPCTL to specify probabilistic hyperproperties for MDPs. Compared to our previous work [7, 13], we re-design the semantics to extend more naturally, the standard PCTL semantics. Furthermore, while in the previous definition we allowed specification of a single model, the logic we present here enables simultaneous behavioral observations across different models. This is a feature often required, since hyperproperties can express the requirements and then relation between multiple models (e.g., conformance of one model with another).

Note that our DTMC and MDP model definitions unfold from initial states, as we are interested in observable differences caused by different initial conditions. As usual, our notion of observability of a path $s_0 s_1 \ldots$ is based on its *trace* $L(s_0)L(s_1)\ldots$. Standard probabilistic temporal logics can argue about the probabilities of certain observations, i.e., statistical conclusions that we can draw when we run an uncertain system over and over again, starting from a fixed, single initial state. By contrast, hyperproperties are more expressive and allow to argue about the probabilities of observations made on synchronous executions in different models or in the same model but rooted in potentially different initial states.

3.1. HyperPCTL Syntax

To formalize probabilistic hyperproperties, we define the temporal logic HyperPCTL. HyperPCTL formulas start with *scheduler quantifiers*, followed by *state quantifiers*, and then an inner formula based on an extended PCTL syntax.

Intuitively, HyperPCTL formulas will be evaluated in the context of some MDP models. Each scheduler quantifier refers to one of these MDP models and gets instantiated with concrete schedulers for this MDP. Each such scheduler instantiation provides an induced DTMC model; universal scheduler quantification requires the property in the quantifier's scope to hold in all possible induced DTMCs, whereas existential quantification requires the existence of an induced DTMC satisfying the property.

In each such induced DTMC model we can start a new "experiment" to observe its behavior by using a state quantifier; universal state quantification requires the property in the quantifier's scope to hold independently of the initial state, whereas existential quantification requires only the existence of a state from where the property is satisfied.

Having several state quantifiers in a formula, each of them starts a new experiment in one of the induced DTMCs. Since the state quantifiers are all in one block at the beginning of the formula, all experiments start simultaneously and run in parallel, executing their transition steps synchronously, but without having any possibility to influence each other's executions, i.e., maintaining stochastic independence. In order to be able to refer to each experiment in a unique way, in HyperPCTL formulas we annotate atomic propositions with a lower index that identifies the respective experiment by its corresponding state variable (used in the state quantifier that started the experiment). Using these indexed atomic propositions, HyperPCTL offers the possibility to relate observations made for different experiments, e.g., comparing reachability probabilities in different MDP models, in the same MDP under different schedulers, or in the same MDP under the same scheduler but starting from different states.

Besides indexing atomic propositions, the inner non-quantified formula part is more or less a PCTL formula, but we allow more general arithmetic expressions over probabilities.

Note that we restrict ourselves to quantifying first the schedulers then the states. Thereby, different state variables can share the same scheduler. One could consider also *local* schedulers when different experiments cannot explicitly share the same scheduler, or in other words, each scheduler quantifier belongs to exactly one of the quantified states. Furthermore, technically it would also be possible to quantify first over states and then over schedulers, or allow arbitrary quantification order with some flexible mechanism to bind state and scheduler quantification. We decided for the below presented approach because it is expressive enough to formalize relevant security properties but simple enough to have an easily understandable semantics close to standard temporal logics.

Formally, HyperPCTL formulas are built according to the following abstract grammar:

$$
\begin{array}{lll}
\begin{array}{l}\textit{scheduler} - \textit{quantified}\\ \textit{formula}:\end{array} & \varphi^{sch} & ::= \ \forall\hat{\sigma}(\hat{\mathcal{M}}).\varphi^{sch} \mid \exists\hat{\sigma}(\hat{\mathcal{M}}).\varphi^{sch} \mid \varphi^{st} \\
\begin{array}{l}\textit{state} - \textit{quantified}\\ \textit{formula}:\end{array} & \varphi^{st} & ::= \ \forall\hat{s}(\hat{\mathcal{M}}^{\hat{\sigma}}).\varphi^{st} \mid \exists\hat{s}(\hat{\mathcal{M}}^{\hat{\sigma}}).\varphi^{st} \mid \varphi^{nq} \\
\begin{array}{l}\textit{non} - \textit{quantified}\\ \textit{formula}:\end{array} & \varphi^{nq} & ::= \ \texttt{true} \mid a_{\hat{s}} \mid \varphi^{nq} \wedge \varphi^{nq} \mid \neg\varphi^{nq} \mid \varphi^{pr} \sim \varphi^{pr} \\
\textit{probability expression}: & \varphi^{pr} & ::= \ \mathbb{P}(\varphi^{path}) \mid f(\varphi^{pr}, \dots, \varphi^{pr}) \\
\textit{path formula}: & \varphi^{path} & ::= \ \bigcirc\varphi^{nq} \mid \varphi^{nq}\ \mathcal{U}\ \varphi^{nq} \mid \varphi^{nq}\ \mathcal{U}^{[k_1,k_2]}\ \varphi^{nq}
\end{array}
$$

where $\hat{\sigma}$ is a *scheduler variable*, $\hat{\mathcal{M}}$ is an *MDP reference*, $\hat{s}$ is a *state variable*, a is an atomic proposition, $f\colon [0,1]^k \to \mathbb{R}$ are k-ary arithmetic operators (e.g., binary addition, unary/binary subtraction, binary multiplication) over probabilities where constants are viewed as 0-ary functions, $\sim$ is $\{\leq, <, =, \neq, >, \geq\}$, and $k_1, k_2 \in \mathbb{Z}_{\geq 0}$ with $k_1 \leq k_2$. The probability operator $\mathbb{P}$ denotes the probability that its operand is true on paths starting in a given state, and '$\bigcirc$' and '$\mathcal{U}$' are the usual temporal operators 'next' and 'until', respectively.

Note that we use $\hat{X}$ to refer to *variables* ranging over objects X. For example, $\hat{\sigma}$, $\hat{s}$, and $\hat{\mathcal{M}}$ denote scheduler variables, state variables, and MDP references, respectively, whereas σ, s, and $\mathcal{M}$ represent concrete schedulers, states, and MDPs. We define *models*(φ) to be the set of all MDP references occurring in φ.

A HyperPCTL *construct* φ is any HyperPCTL expression of the form φ^{sch}, φ^{st}, φ^{nq}, φ^{pr}, or φ^{path}. A HyperPCTL scheduler-quantified formula is *well-formed* if each occurrence of any $a_{\hat{s}}$ is in the scope of a *state quantifier* for $\hat{s}(\hat{\mathcal{M}}^{\hat{\sigma}})$, and any state quantifier for $\hat{s}(\hat{\mathcal{M}}^{\hat{\sigma}})$ is in the scope of a *scheduler quantifier* for $\hat{\sigma}(\hat{\mathcal{M}})$.

HyperPCTL formulas are well-formed HyperPCTL scheduler-quantified formulas, where we additionally allow standard syntactic sugar like $\texttt{false} = \neg\texttt{true}$, $\varphi_1 \vee \varphi_2 = \neg(\neg\varphi_1 \wedge \neg\varphi_2)$, $\varphi_1 \to \varphi_2 = \neg(\varphi_1 \wedge \neg\varphi_2)$, $\Diamond\varphi = \texttt{true}\,\mathcal{U}\,\varphi$, $\varphi_1 \leftrightarrow \varphi_2 = \neg(\varphi_1 \wedge \neg\varphi_2) \bigwedge \neg(\varphi_2 \wedge \neg\varphi_1)$, and $\mathbb{P}(\Box\varphi) = 1 - \mathbb{P}(\Diamond\neg\varphi)$.

3.2. HyperPCTL Semantics

Assume a HyperPCTL formula φ which refers to the MDPs *models*$(\varphi) = \{\hat{\mathcal{M}}_1, \hat{\mathcal{M}}_2, \ldots, \hat{\mathcal{M}}_k\}$. To define the satisfaction relation for φ, we first need to define the models for which we want to evaluate φ. This is done by a function $m\colon$ *models*$(\varphi) \to \mathbb{M}$ assigning to each MDP reference a (non-empty) MDP. To apply the model instantiation m to φ, we syntactically and simultaneously replace occurrences of each $\hat{\mathcal{M}}_i$, where $1 \leq i \leq k$, in φ by $\mathcal{M}_i = m(\hat{\mathcal{M}}_i)$, whose result we denote by $\varphi\big[m(\mathit{models}(\varphi))/\mathit{models}(\varphi)\big]$. The model instantiation turns all sub-constructs for state quantification of the form $\forall\hat{s}(\hat{\mathcal{M}}_i).\varphi'$ to $\forall\hat{s}(\mathcal{M}_i).\varphi'$ and similarly for the existential case. Refer to Example 1 for an illustration of these evaluations.

Initially, the evaluation starts with $n = 0$, the empty DTMC $\mathcal{D}_\emptyset$ and the "empty" state sequence $()$:

$$m \models \varphi \qquad \textit{iff} \qquad 0, \mathcal{D}_\emptyset, () \models \varphi[m(\mathit{models}(\varphi))/\mathit{models}(\varphi)] \tag{3}$$

Note that the evaluation now proceeds in the context of a *DTMC*, not an MDP. The reason is the scheduler order: we quantify first over schedulers and then over states in the induced DTMCs. Consequently, when instantiating $\forall\hat{\sigma}(\mathcal{M}).\varphi$ or $\exists\hat{\sigma}(\mathcal{M}).\varphi$ by a scheduler $\sigma \in \Sigma^{\mathcal{M}}$, we syntactically replace in φ each $\mathcal{M}^{\hat{\sigma}}$ that is not in the scope of a quantifier for $\hat{\sigma}$, by the induced DTMC $\mathcal{M}^{\sigma}$, and denote the result by $\varphi[\hat{\sigma} \rightsquigarrow \sigma]$.

For each state quantification $\forall\hat{s}(\mathcal{M}^\sigma).\varphi$ and $\exists\hat{s}(\mathcal{M}^\sigma).\varphi$, the DTMC in the evaluation context will be parallelly composed with a new "experiment", i.e., a copy of the induced DTMC $\mathcal{M}^\sigma$ with renamed atomic propositions in order to be able to individually refer to the observations in this "experiment". Furthermore, we extend the state sequence $\vec{s}$ from the current context, by appending the state s with which we instantiate the state quantifier as initial state for the new experiment in $\mathcal{M}^\sigma$.

To get a unique renaming, upon instantiating the nth state quantifier with state s in DTMC $\mathcal{M}^\sigma$, we create a copy of $\mathcal{M}^\sigma$ in which we rename each atomic proposition a to a_n; we denote the result of this renaming by $\mathcal{M}^\sigma[n]$. Formally, for a DTMC $\mathcal{D} = (S, \mathbf{P}, \mathsf{AP}, L)$ we define $\mathcal{D}[n] = (S, \mathbf{P}, \mathsf{AP}[n], L[n])$ with $\mathsf{AP}[n] = \{a_n \mid a \in \mathsf{AP}\}$ and $L[n](s) = \{a_n \mid a \in L(s)\}$.

Besides renaming in the model for unique observations, we also need a corresponding renaming in the formula to link the relevant atomic propositions to this copy with identity n. We do so by replacing in φ each freely occurring[1] $a_{\hat{s}}$ by a_n, whose result we denote by $\varphi[\hat{s} \rightsquigarrow n]$.

Formally, the satisfaction relation is defined for a non-negative integer n, a DTMC $\mathcal{D} = (S, \mathbf{P}, \mathsf{AP}, L)$, and a state sequence $\vec{s} \in S^n$ by the following rules:

Table 1: Semantic rules

(1)	$n, \mathcal{D}, \vec{s} \models \forall\hat{\sigma}(\mathcal{M}).\varphi$	*iff*	$n, \mathcal{D}, \vec{s} \models \varphi[\hat{\sigma} \rightsquigarrow \sigma]$ *for all* $\sigma \in \Sigma^{\mathcal{M}}$
(2)	$n, \mathcal{D}, \vec{s} \models \exists\hat{\sigma}(\mathcal{M}).\varphi$	*iff*	$n, \mathcal{D}, \vec{s} \models \varphi[\hat{\sigma} \rightsquigarrow \sigma]$ *for some* $\sigma \in \Sigma^{\mathcal{M}}$
(3)	$n, \mathcal{D}, \vec{s} \models \forall\hat{s}(\mathcal{M}^\sigma).\varphi$	*iff*	$n{+}1, \mathcal{D} \times \mathcal{M}^\sigma[n{+}1], \vec{s} \circ (init(s'), s')$ $\models \varphi[\hat{s} \rightsquigarrow n{+}1]$, *for all states* s' *of* $\mathcal{M}^\sigma$
(4)	$n, \mathcal{D}, \vec{s} \models \exists\hat{s}(\mathcal{M}^\sigma).\varphi$	*iff*	$n{+}1, \mathcal{D} \times \mathcal{M}^\sigma[n{+}1], \vec{s} \circ (init(s'), s')$ $\models \varphi[\hat{s} \rightsquigarrow n{+}1]$, *for some state* s' *of* $\mathcal{M}^\sigma$
(5)	$n, \mathcal{D}, \vec{s} \models \texttt{true}$		
(6)	$n, \mathcal{D}, \vec{s} \models a_i$	*iff*	$a_i \in L(\vec{s})$
(7)	$n, \mathcal{D}, \vec{s} \models \varphi_1^{nq} \wedge \varphi_2^{nq}$	*iff*	$n, \mathcal{D}, \vec{s} \models \varphi_1^{nq}$ *and* $n, \mathcal{D}, \vec{s} \models \varphi_2^{nq}$
(8)	$n, \mathcal{D}, \vec{s} \models \neg\varphi^{nq}$	*iff*	$n, \mathcal{D}, \vec{s} \not\models \varphi^{nq}$
(9)	$n, \mathcal{D}, \vec{s} \models \varphi_1^{pr} \sim \varphi_2^{pr}$	*iff*	$[\![\varphi_1^{pr}]\!]_{n,\mathcal{D},\vec{s}} \sim [\![\varphi_2^{pr}]\!]_{n,\mathcal{D},\vec{s}}$
(10)	$[\![\mathbb{P}(\varphi^{path})]\!]_{n,\mathcal{D},\vec{s}}$	$=$	$\Pr_{\vec{s}}^{\mathcal{D}}\big(\bigcup_{\pi \in \{\pi' \in Paths_{\vec{s}}^{\mathcal{D}} \mid n,\mathcal{D},\pi' \models \varphi^{path}\}} Cyl^{\mathcal{D}}(\pi)\big)$
(11)	$[\![f(\varphi_1^{pr}, \ldots \varphi_k^{pr})]\!]_{n,\mathcal{D},\vec{s}}$	$=$	$f\big([\![\varphi_1^{pr}]\!]_{n,\mathcal{D},\vec{s}}, \ldots, [\![\varphi_k^{pr}]\!]_{n,\mathcal{D},\vec{s}}\big)$
(12)	$n, \mathcal{D}, \pi \models \bigcirc\varphi^{nq}$	*iff*	$n, \mathcal{D}, \pi[1] \models \varphi^{nq}$
(13)	$n, \mathcal{D}, \pi \models \varphi_1^{nq}\,\mathcal{U}\,\varphi_2^{nq}$	*iff*	*exists* $j \geq 0$ *such that* $n, \mathcal{D}, \pi[j] \models \varphi_2^{nq}$ *and* $n, \mathcal{D}, \pi[i] \models \varphi_1^{nq}$ *for all* $i \in [0, j)$
(14)	$n, \mathcal{D}, \pi \models \varphi_1^{nq}\,\mathcal{U}^{[k_1,k_2]}\,\varphi_2^{nq}$	*iff*	*exists* $j \in [k_1, k_2]$ *such that* $n, \mathcal{D}, \pi[j] \models \varphi_2^{nq}$ *and* $n, \mathcal{D}, \pi[i] \models \varphi_1^{nq}$ *for all* $i \in [0, j)$

In Table 1, $\mathcal{M}$ is an MDP, $\sigma = (Q, act, mode, init) \in \Sigma^{\mathcal{M}}$ is a scheduler for $\mathcal{M}$, and $init(s)$ is the function selecting the starting mode for s. When we instantiate each state quantifier with states from an induced DTMC $\mathcal{D}$, we use $\vec{s} \circ (init(s'), s')$ to append the existing sequence of states and initial modes, namely $\vec{s}$, with the new state s' and its initial mode $init(s')$. Additionally, $a_i \in \mathsf{AP}$ is an atomic proposition in $\mathcal{D}$; φ^{st} are $\mathsf{HyperPCTL}$ state-quantified formulas; $\varphi^{nq}, \varphi_1^{nq}, \varphi_2^{nq}$ are $\mathsf{HyperPCTL}$ non-quantified formulas; $\varphi_1^{pr}, \ldots, \varphi_k^{pr}$ are

[1]That is, not in the scope of any quantifier for $\hat{s}$ in φ.

HyperPCTL probability expressions; and φ^{path} is a HyperPCTL path formula which is evaluated in the context of a non-negative integer n, a DTMC $\mathcal{D}$ and a path $\pi = s_0 s_1 \cdots$ of $\mathcal{D}$ as in rows (12), (13), and (14) where $k_1, k_2 \in \mathbb{Z}_{\geq 0}$ with $k_1 \leq k_2$.

Example 1. *We illustrate the semantical evaluation of the following HyperPCTL formula on the MDP in Fig. 1b. Note that in this example, $h{>}0$, $h{\leq}0$, $l{=}1$ and $l{=}2$ are atomic propositions.*

$$\forall\hat{\sigma}(\hat{\mathcal{M}}).\forall\hat{s}(\hat{\mathcal{M}}^{\hat{\sigma}}).\forall\hat{s}'(\hat{\mathcal{M}}^{\hat{\sigma}}).\ \Big((h{>}0)_{\hat{s}}\wedge(h{\leq}0)_{\hat{s}'}\Big) \to \Big(\mathbb{P}(\Diamond\,(l{=}1)_{\hat{s}})=\mathbb{P}(\Diamond\,(l{=}1)_{\hat{s}'})\Big)$$

Let us consider the model assignment $m\colon \{\hat{\mathcal{M}}\} \to \mathbb{M}$ with MDP $m(\hat{\mathcal{M}}) = \mathcal{M} = (S, Act, \mathbf{P}, \mathsf{AP}, L)$. Each scheduler σ for $\mathcal{M}$ induces a DTMC $\mathcal{M}^\sigma$, whose state set we denote by S^σ. To illustrate a simple case, we assume our schedulers are memoryless, i.e., they have only one possible mode. Hence, we have removed any mention of mode of the scheduler at each state.

$$\begin{aligned}
m \models \forall\hat{\sigma}(\hat{\mathcal{M}}).\forall\hat{s}(\hat{\mathcal{M}}^{\hat{\sigma}}).\forall\hat{s}'(\hat{\mathcal{M}}^{\hat{\sigma}}).\ & \Big((h{>}0)_{\hat{s}} \wedge (h{\leq}0)_{\hat{s}'}\Big) \to \\
& \Big(\mathbb{P}(\Diamond\,(l{=}1)_{\hat{s}})=\mathbb{P}(\Diamond\,(l{=}1)_{\hat{s}'})\Big) \qquad \textit{iff} \\
0, \mathcal{D}_\emptyset, () \models \forall\hat{\sigma}(\mathcal{M}).\forall\hat{s}(\mathcal{M}^{\hat{\sigma}}).\forall\hat{s}'(\mathcal{M}^{\hat{\sigma}}).\ & \Big((h{>}0)_{\hat{s}} \wedge (h{\leq}0)_{\hat{s}'}\Big) \to \\
& \Big(\mathbb{P}(\Diamond\,(l{=}1)_{\hat{s}})=\mathbb{P}(\Diamond\,(l{=}1)_{\hat{s}'})\Big) \qquad \textit{iff} \\
0, \mathcal{D}_\emptyset, () \models \forall\hat{s}(\mathcal{M}^{\sigma}).\forall\hat{s}'(\mathcal{M}^{\sigma}).\ & \Big((h{>}0)_{\hat{s}} \wedge (h{\leq}0)_{\hat{s}'}\Big) \to \\
\Big(\mathbb{P}(\Diamond\,(l{=}1)_{\hat{s}})=\mathbb{P}(\Diamond\,(l{=}1)_{\hat{s}'})\Big),\ & \textit{for all } \sigma \in \Sigma^{\mathcal{M}} \qquad \textit{iff} \\
1, \mathcal{M}^\sigma[1], \big((init(s), s)\big) \models \forall\hat{s}'(\mathcal{M}^{\sigma}).\ & \Big((h{>}0)_{1}\wedge(h{\leq}0)_{\hat{s}'}\Big) \to \\
\Big(\mathbb{P}(\Diamond\,(l{=}1)_{1})=\mathbb{P}(\Diamond\,(l{=}1)_{\hat{s}'})\Big),\ & \textit{for all } \sigma{\in}\Sigma^{\mathcal{M}},\ \textit{for all } s{\in}S^\sigma \qquad \textit{iff} \\
2, \mathcal{M}^\sigma[1]{\times}\mathcal{M}^\sigma[2], \big((init(s), s), (init(s'), s')\big){\models}\ & \Big((h{>}0)_{1} \wedge (h{\leq}0)_{2}\Big) \to \\
\Big(\mathbb{P}(\Diamond\,(l{=}1)_{1})=\mathbb{P}(\Diamond\,(l{=}1)_{2})\Big),\ & \textit{for all } \sigma{\in}\Sigma^{\mathcal{M}},\ \textit{for all } s, s'{\in}S^\sigma
\end{aligned}$$

The first statement expresses that the model assignment satisfies the given property. The second statement follows from the use of Eq. (3) to initialize the necessary parameters for the evaluation, including the instantiation of the MDP reference $\hat{\mathcal{M}}$ by the concrete MDP $\mathcal{M}$. The third statement is achieved by using the first semantics rule from Table 1, instantiating the scheduler variable $\hat{\sigma}$ by a concrete scheduler σ for $\mathcal{M}$. In the final two steps, we instantiate the state quantifiers by the MDP states s and s', in this order. From this point on, the evaluation follows the standard PCTL evaluation in the state (s, s') of the composed DTMC $\mathcal{M}^\sigma[1] \times \mathcal{M}^\sigma[2]$.

4. Applications of HyperPCTL

We now put HyperPCTL into action by formulating probabilistic hyperproperties from different application areas. We start with examples without nondeterminism, followed by examples where non-determinism plays a role.

4.1. Application in DTMCs

To simplify the formalism, we handle deterministic MDP models as DTMCs, i.e., we skip the (unique) actions, and skip scheduler quantification from the formulas, e.g., we write $\forall \hat{s}(\hat{\mathcal{D}}).\varphi$ instead of $\forall \hat{\sigma}(\hat{\mathcal{M}}).\forall \hat{s}(\hat{\mathcal{M}}^{\hat{\sigma}}).\varphi$.

4.1.1. Probabilistic noninterference

Probabilistic noninterference [15] establishes connection between information theory and information flow by employing probabilities to address covert channels. *Noninterference* is an information-flow security policy that enforces that a low-privileged user (e.g., an attacker) should not be able to distinguish two computations from their publicly observable outputs, if they only vary in their inputs by a high-privileged user (e.g., a secret). Intuitively, it requires that the probability of every low-observable trace pattern is the same for every low-equivalent initial state. Probabilistic noninterference can be expressed in HyperPCTL as follows, where l denotes a low-observable atomic proposition:

$$\forall \hat{s}(\hat{\mathcal{D}}).\forall \hat{s}'(\hat{\mathcal{D}}).\Big(l_{\hat{s}} \wedge l_{\hat{s}'}\Big) \rightarrow \mathbb{P}\Big(\Box\big(\mathbb{P}(\bigcirc l_{\hat{s}}) = \mathbb{P}(\bigcirc l_{\hat{s}'})\big)\Big) = 1$$

We emphasize that our logic in its current form is not able to reason about formulas that are invariant under stuttering. Since we consider discrete-time models where time evolves synchronously, we believe stuttering is not relevant here. In classic probabilistic noninterference, however, the probability of reachability is computed for stutter-equivalent traces. Since HyperPCTL has synchronous semantics, traces have to be analyzed step by step. For example, consider the model in Fig. 3 and the noninterference hyperproperty above. The two DTMCs have the same probabilistic language in terms of absence and presence of l, but the next step probabilities of moving to a state labeled l differs for s_1 and s_2 (for the DTMC on the right). Given our current logic, the above hyperproperty would not hold in this example as all states labeled l in the DTMC on the right do not move to a state labeled l with equal probability. To ensure that the hyperproperty holds for all states labeled l, we would need to incorporate a stuttering equivalence mechanism in our logic. This would allow the traces for the DTMC on the right to advance, while the traces for the DTMC on the left stutter and ensure that the probabilistic distributions of reaching states labeled l are equal.

4.1.2. Differential Privacy

Differential privacy [16] is a commitment by a data holder to a data subject (normally an individual) that they will not be affected by allowing their data to be used in any study or analysis. Formally, let ϵ be a positive real number and $\mathcal{A}$ be a randomized algorithm that makes a query to an input database and produces an output. Algorithm $\mathcal{A}$ is called *ϵ-differentially private*, if for all databases D_1 and D_2 that differ on a single element, and all subsets S of

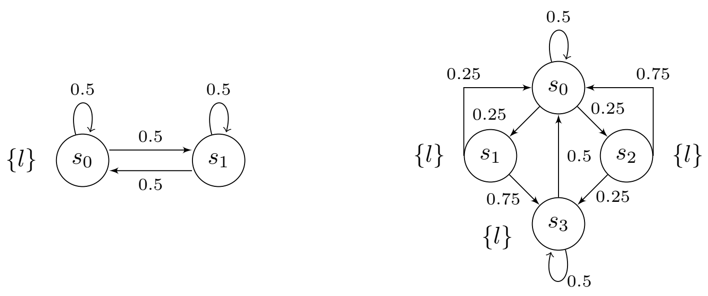


Figure 3: DTMCs that should satisfy *noninterference* hyperproperty in the presence of a stutter-equivalence mechanism.

possible outputs of $\mathcal{A}$, we have $\Pr[\mathcal{A}(D_1) \in S] \leq e^{\epsilon} \cdot \Pr[\mathcal{A}(D_2) \in S]$. Differential privacy can be expressed in HyperPCTL by the following formula:

$$\forall \hat{s}(\hat{\mathcal{D}}).\forall \hat{s}'(\hat{\mathcal{D}}).\Big[dbSim(\hat{s}, \hat{s}')\Big] \to \Big[\mathbb{P}\Big(\diamondsuit(qOut \in S)_{\hat{s}}\Big) \leq e^{\epsilon} \cdot \mathbb{P}\Big(\diamondsuit(qOut \in S)_{\hat{s}'}\Big)\Big]$$

where $dbSim(\hat{s}, \hat{s}')$ means that two different dataset inputs have all but one similarity and $qOut$ is the result of the query. For example, one way to provide differential privacy is through a *randomized response* protocol which adds noise to the response and provides plausible deniability. Let A be an embarrassing or illegal activity. In a social study, each participant is faced with the query, "Have you engaged in activity A in the past week?" and is instructed to respond by the following protocol:

1. Flip a fair coin.
2. If tail, then answer truthfully.
3. If head, then flip the coin again and respond "Yes" if head and "No" if tail.

Thus, a "Yes" response may have been offered because the first and second coin flips were both heads. This implies that there are no good or bad responses and an answer cannot be incriminating.

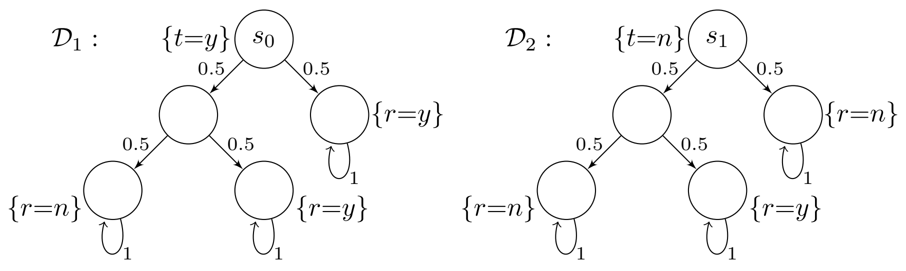


Figure 4: Markov chain of the randomized response protocol.

We now show that this social study is $(\ln 3)$-differentially private. For each participant in the study, Fig. 4 shows two Markov chains $\mathcal{D}_1$ and $\mathcal{D}_2$ of the response protocol for cases denoted by $\{t{=}y\}$ (respectively, $\{t{=}n\}$) which means the participant did (respectively, did not) engage in activity A. Also, $\{r{=}y\}$ (respectively, $\{r{=}n\}$) means that the participant responds "Yes" (respectively, "No"). The HyperPCTL formula to express $(\ln 3)$-differential privacy for this protocol is the following:

$$\forall \hat{s}(\hat{\mathcal{D}}_1).\forall \hat{s}'(\hat{\mathcal{D}}_2).\Big[\Big((t{=}n)_{\hat{s}} \wedge (t{=}y)_{\hat{s}'}\Big) \rightarrow \Big(\mathbb{P}\Big(\Diamond(r{=}n)_{\hat{s}}\Big) \leq e^{\ln 3}\cdot \mathbb{P}\Big(\Diamond(r{=}n)_{\hat{s}'}\Big)\Big)\Big]$$
$$\wedge \Big[\Big((t{=}y)_{\hat{s}} \wedge (t{=}n)_{\hat{s}'}\Big) \rightarrow \Big(\mathbb{P}\Big(\Diamond(r{=}y)_{\hat{s}}\Big) \leq e^{\ln 3}\cdot \mathbb{P}\Big(\Diamond(r{=}y)_{\hat{s}'}\Big)\Big)\Big]$$

We have decomposed $\mathit{dbSim}(\hat{s}, \hat{s}')$ into two cases of $\{t = y\}$ and $\{t = n\}$. Thus, in the left conjunct, the set S represents the case where the response is "No" and in the right conjunct, the set S represents the case where the response is "Yes". The DTMCs in Fig. 4 satisfy the formula for all instantiations of $\hat{s}$ and $\hat{s}'$. In the left conjunct, when $\hat{s}$ and $\hat{s}'$ are instantiated by s_1 and s_0, respectively, $\mathbb{P}\big(\Diamond(r{=}n)_{\hat{s}}\big) = 0.5 * 0.5 + 0.5 = 0.75$ and $\mathbb{P}\big(\Diamond(r{=}n)_{\hat{s}'}\big) = 0.5 * 0.5 = 0.25$, thus, satisfying the formula. In the right conjunct, when $\hat{s}$ and $\hat{s}'$ are instantiated by s_0 and s_1, respectively, $\mathbb{P}\big(\Diamond(r{=}y)_{\hat{s}}\big) = 0.5{*}0.5{+}0.5 = 0.75$ and $\mathbb{P}\big(\Diamond(r{=}y)_{\hat{s}'}\big) = 0.5 * 0.5 = 0.25$, thus, satisfying the formula. For all other state instantiations, antecedents of the implications are false, thus, vacuously satisfying the formula.

4.2. Applications in MDPs

4.2.1. Side-channel Timing Leaks

Side-channel timing leaks open a channel to an attacker to infer the value of a secret by observing the execution time of a function. For example, at the heart of the RSA public-key encryption algorithm is the modular exponentiation algorithm that computes ($a^b \mod n$), where a is an integer representing the plain text and b is the integer encryption key. A careless implementation can leak b through a probabilistic scheduling channel (see Fig. 5). This program is not secure since the two branches of the `if` have different timing behaviors.

```
1  void mexp(){
2   c = 0; d = 1; i = k;
3   while (i >= 0){
4    i = i-1;
5    c = c*2;
6    d = (d*d) % n;
7    if (b(i) = 1){
8     c = c+1;
9     d = (d*a) % n;
10   }
11  }
12 }
13 ...
14 t = new Thread(mexp());
15 j = 0; m = 2 * k;
16 while (j < m & !t.stop)
17  j++;
18
```

Figure 5: Modular exponentiation.

Under a fair execution scheduler for parallel threads, an attacker thread can infer the value of b by running in parallel to a modular exponentiation thread and iteratively incrementing a counter variable until the other thread terminates (lines 14-17). To model this program by an MDP, we can use two nondeterministic actions for the two branches of the `if` statement, such that the choice of different schedulers corresponds to the choice of different bit configurations `b(i)` for the key `b`. This algorithm should satisfy the following property: the probability of observing a concrete value in the counter `j` should be independent of the bit configuration of the secret key `b`:

$$\forall \hat{\sigma}_1(\hat{\mathcal{M}}).\forall \hat{\sigma}_2(\hat{\mathcal{M}}).\forall \hat{s}(\hat{\mathcal{M}}^{\hat{\sigma}_1}).\forall \hat{s}'(\hat{\mathcal{M}}^{\hat{\sigma}_2}).$$
$$\Big(init_{\hat{s}} \wedge init_{\hat{s}'}\Big) \rightarrow \bigwedge_{\ell=0}^{m} \Big(\mathbb{P}(\Diamond(j=\ell)_{\hat{s}}) = \mathbb{P}(\Diamond(j=\ell)_{\hat{s}'})\Big)$$

Another example of timing attacks that can be implemented through a probabilistic scheduling side channel is password verification which is typically implemented by comparing an input string with another confidential string (see Fig. 6). Also here, an attacker thread can measure the time necessary to break the loop, and use this information to infer the prefix of the input string matching the secret string.

```
int str_cmp(char * r){
 char * s = ‘Bg\$4\0’;
 i = 0;
 while(s[i]!=‘\0’){
  if(r[i]==‘\0’ ||
     s[i]!=r[i])
   return 0;
  i++;
 }
 if(r[i]==‘\0’)
  return 1;
 return 0;
}
```

Figure 6: String comparison.

4.2.2. Scheduler-Specific Observational Determinism Policy (SSODP)

SSODP [17] is a confidentiality policy in multi-threaded programs that defends against an attacker that chooses an appropriate scheduler to control the set of possible traces. In particular, given any scheduler and two initial states that are indistinguishable with respect to a secret input (i.e., low-equivalent), any two executions from these two states should terminate in low-equivalent states with equal probability. Formally, using a proposition h to represent a secret and propositions $l \in L$ to classify low-equivalent states:

$$\forall \hat{\sigma}(\hat{\mathcal{M}}).\forall \hat{s}(\hat{\mathcal{M}}^{\hat{\sigma}}).\forall \hat{s}'(\hat{\mathcal{M}}^{\hat{\sigma}}).\ \big(h_{\hat{s}} \oplus h_{\hat{s}'}\big) \rightarrow \bigwedge_{l \in L} \big(\mathbb{P}(\Diamond l_{\hat{s}}) = \mathbb{P}(\Diamond l_{\hat{s}'})\big)$$

where $\oplus$ is the exclusive-or operator. A stronger variation of this policy is that the executions are stepwise low-equivalent:

$$\forall \hat{\sigma}(\hat{\mathcal{M}}).\forall \hat{s}(\hat{\mathcal{M}}^{\hat{\sigma}}).\forall \hat{s}'(\hat{\mathcal{M}}^{\hat{\sigma}}).\ \big(h_{\hat{s}} \oplus h_{\hat{s}'}\big) \rightarrow \mathbb{P}\big(\Box \bigwedge_{l \in L} \big(\mathbb{P}(\bigcirc l_{\hat{s}}) = \mathbb{P}(\bigcirc l_{\hat{s}'})\big)\big) = 1.$$

4.2.3. Probabilistic Conformance

Probabilistic conformance describes how well a model and an implementation conform with each other with respect to a specification. As an example, consider a six-sided die. The probability to obtain one possible side of the die is 1/6. We would like to synthesize a protocol that simulates the 6-sided die behavior only by repeatedly tossing a fair coin. We know that such an implementation exists [18] (see Fig. 7), but we aim to find such a solution automatically. We model the die as a DTMC $\mathcal{D}$, and possible coin-implementations as an MDP $\mathcal{M}$

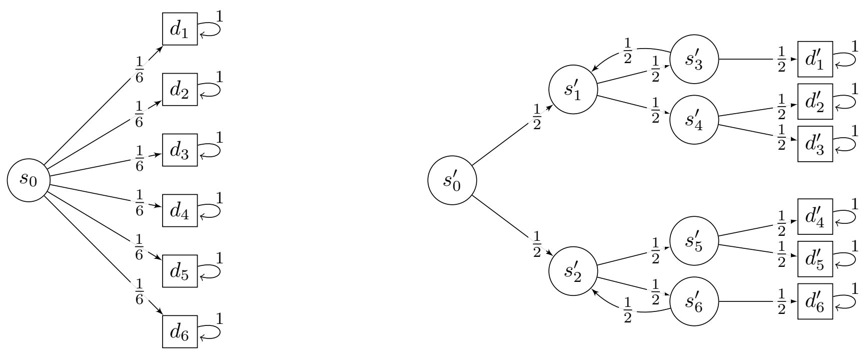


Figure 7: **Left:** DTMC model of a fair 6-sided die. **Right:** DTMC model of the Knuth-Yao Algorithm [18] to simulate a fair 6-sided die by fair coin tosses.

with a given number of states, including six absorbing final states to model the outcomes. Each non-final state of the MDP has a set of enabled actions, each of them choosing two successor states with equal probability 1/2. We want to know whether there is a scheduler whose induced outcome probabilities agree with the DTMC, giving us a particular implementation to simulate the die:

$$\exists\hat{\sigma}(\hat{\mathcal{M}}).\forall\hat{s}(\hat{\mathcal{D}}).\exists\hat{s}'(\hat{\mathcal{M}}^{\hat{\sigma}}).\ init_{\hat{s}} \rightarrow \Big(init_{\hat{s}'} \wedge \bigwedge_{\ell=1}^{6} \Big(\mathbb{P}(\Diamond(die{=}\ell)_{\hat{s}}) = \mathbb{P}(\Diamond(die{=}\ell)_{\hat{s}'})\Big)\Big)$$

5. The Expressive Power of HyperPCTL

In order to satisfy $\mathbb{P}_{\sim c}(\varphi)$ in a given MDP state s, the standard PCTL semantics requires that *all* schedulers should induce a DTMC that satisfies $\mathbb{P}_{\sim c}(\varphi)$ in s. Though it should hold for *all* schedulers, it is known that there exist minimal and maximal schedulers that are non-probabilistic and memoryless, therefore it is sufficient to restrict the reasoning to such schedulers. Since for MDPs with finite state and action spaces, the number of such schedulers is finite, PCTL model checking for MDPs is decidable. Given this analogy, one would expect that HyperPCTL model checking should be decidable, but it is not.

5.1. HyperPCTL Model Checking for Memoryful Probabilistic Schedulers

Theorem 1. *HyperPCTL model checking for MDPs is in general undecidable.*

Before we prove the above theorem, let us explore shortly, the source of increased expressiveness with respect to PCTL that makes HyperPCTL undecidable. State quantification cannot be the source, as the state space is finite and thus, there are finitely many possible state quantifier instantiations.

In PCTL, each probability bound needs to be satisfied under all schedulers. However, when a PCTL formula has several probability bounds, its satisfaction requires each bound to be satisfied by all schedulers *independently*. For example,

$\mathcal{M}, s \models_{\mathsf{PCTL}} \Big(\mathbb{P}_{<0.5}(a\,\mathcal{U}\,b) \;\vee\; \mathbb{P}_{>0.5}(a\,\mathcal{U}\,b)\Big)$ is equivalent to

$\mathcal{M} \models_{\mathsf{HyperPCTL}} \forall\hat{\sigma}.\forall\hat{s}(\hat{\sigma}).\ (init_{\hat{s}} \rightarrow \mathbb{P}(a_{\hat{s}}\,\mathcal{U}\,b_{\hat{s}}) < 0.5)$ *or*

$\mathcal{M} \models_{\mathsf{HyperPCTL}} \forall\hat{\sigma}.\forall\hat{s}(\hat{\sigma}).\ (init_{\hat{s}} \rightarrow \mathbb{P}(a_{\hat{s}}\,\mathcal{U}\,b_{\hat{s}}) > 0.5)$

but *not* equivalent to the HyperPCTL formula

$\mathcal{M} \models_{\mathsf{HyperPCTL}} \forall\hat{\sigma}.\forall\hat{s}(\hat{\sigma}).\ (init_{\hat{s}} \rightarrow (\mathbb{P}(a_{\hat{s}}\,\mathcal{U}\,b_{\hat{s}}) < 0.5 \vee \mathbb{P}(a_{\hat{s}}\,\mathcal{U}\,b_{\hat{s}}) > 0.5))$

which states that the probability is either less than or larger than 0.5 under all schedulers, which is true if *there exists no scheduler* under which the probability is 0.5 (see also [19]). Thus, even for a fragment restricted to universal scheduler quantification, combinations of probability bounds allow HyperPCTL to express existential *scheduler synthesis* problems.

Finally, consider a scheduler quantifier followed by state quantifiers, whose scope may contain probability expressions. This means that we start several "experiments" in parallel, each one represented by a state quantifier. However, we may use in all experiments the *same scheduler*. Informally, this allows us to express the existence or absence of schedulers with certain probabilistic hyperproperties for the induced DTMCs. It would however also make sense to flip this quantifier order, such that state quantifiers are followed by scheduler quantifiers. This would mean, that we can use different schedulers in the different concurrently running experiments. This would be meaningful e.g., when users can provide input to the system, i.e., when the scheduler choice lies with the "observers" of the individual experiments, and they can adapt their schedulers to observations made in the other concurrently running experiments.

Proof of Theorem 1

To prove Theorem 1, we reduce the *emptiness* problem in *probabilistic Büchi automata* (*PBA*), which is known to be undecidable [20], to our problem. PBA can be viewed as a nondeterministic Büchi automaton where the nondeterminism is resolved by a probabilistic choice. That is, for any state q and letter a in alphabet Σ, either q does not have any a-successor or there is a probabilistic distribution for the a-successors of q.

Definition 6. *A* probabilistic Büchi automaton *(PBA) over a finite alphabet* Σ *is a tuple* $\mathcal{P} = (Q, \delta, \Sigma, \iota_{init}, F)$*, where* Q *is a finite state space,* $\delta\colon Q \times \Sigma \times Q \rightarrow [0,1]$ *is the transition probability function, such that* $\sum_{q' \in Q}(q, a, q') \in \{0,1\}$ *for all* $q \in Q$ *and* $a \in \Sigma$*,* $\iota_{init} \in Q$ *is the initial state, and* $F \subseteq Q$ *is the set of* accepting states*.*

A *run* for an infinite word $w = a_1 a_2 \cdots \in \Sigma^\omega$ is an infinite sequence $\pi = q_0 q_1 q_2 \cdots$ of states in Q, such that $q_{i+1} \in \delta(q_i, a_{i+1}) = \{q' \mid \delta(q_i, a_{i+1}, q') > 0\}$ for all $i \in \mathbb{N}_{\geq 0}$. A *run* begins with the initial state ι_{init}. Let $\mathsf{Inf}(\pi)$ denote the set of states that are visited infinitely often in π. Run π is called *accepting* if $\mathsf{Inf}(\pi) \cap F \neq \emptyset$. Given an infinite input word $w \in \Sigma^\omega$, the behavior of $\mathcal{P}$ is given by the infinite Markov chain that is obtained by unfolding $\mathcal{P}$ into a tree using w. This is similar to an induced Markov chain from an MDP by a scheduler. Hence, standard concepts for Markov chains can be applied to define the acceptance probability of w in $\mathcal{P}$, denoted by $\Pr_{\mathcal{P}}(w)$ or briefly $\Pr(w)$, by the probability measure of the set of accepting runs for w in $\mathcal{P}$. We define the accepted language of $\mathcal{P}$ as: $\mathcal{L}(\mathcal{P}) = \{w \in \Sigma^\omega \mid \Pr_{\mathcal{P}}(w) > 0\}$. The *emptiness problem* is to decide whether or not $\mathcal{L}(\mathcal{P}){=}\emptyset$ for a given input $\mathcal{P}$.

Mapping Our idea of mapping the emptiness problem in PBA to HyperPCTL model checking for MDPs is as follows. We map a PBA to an MDP such that the words of the PBA are mimicked by the runs of the MDP. In other words, letters of the words in the PBA appear as propositions on states of the MDP. This way, the existence of a word in the language of the PBA corresponds to the existence of a scheduler that produces a satisfying computation tree in the induced Markov chain of the MDP.

MDP model: Let $\mathcal{P} = (Q, \delta, \Sigma, \iota_{init}, F)$ be a PBA with alphabet Σ. We obtain an MDP $\mathcal{M} = (S, Act, \mathbf{P}, \mathsf{AP}, L)$ as follows:

- The set of states $S = \big(Q \times (\Sigma \cup \{\epsilon\})\big) \bigcup \{q_{sink}\}$, such that, $\epsilon \notin \Sigma$. Corresponding to the initial state ι_{init} in the PBA, we add a state (q, ϵ) in the MDP and label it with proposition *init*. Additionally, we add a sink state, q_{sink}, to ensure our MDP is complete.
- The set of actions is $Act = \Sigma$.
- The transition probability function $\mathbf{P}\colon S \times Act \times S \to [0,1]$ is defined as follows:
  - If action b is *enabled* at state q in the PBA,
  $$\mathbf{P}\Big((q,a), b, (q', a')\Big) = \delta(q, b, q') \quad \text{where, } a' = b;$$
  - If action b is *not enabled* at state q in the PBA,
  $$\mathbf{P}\Big((q,a), b, (q_{sink})\Big) = 1;$$
  - q_{sink} is an absorbing state, thus,
  $$\mathbf{P}\Big((q_{sink}), b, (q_{sink})\Big) = 1, \ \text{ for all } b \in \Sigma.$$
- The set of atomic propositions is $\mathsf{AP} = \Sigma \cup \{f, init\}$, where f, $init \notin \Sigma$. We use f to label the accepting states and *init* to label the initial state.
- The labeling function L is defined for each $a \in \Sigma \cup \{\epsilon\}$ and $q \in Q$ as follows:
  - For every $(q, a) \in S$, where $a = \epsilon$:
  $$L(s) = \begin{cases} \{init, f\} & \text{if } q \in F, \\ \{init\} & \text{otherwise.} \end{cases}$$

– For every $(q, a) \in S$, where $a \in \Sigma$:

$$L(q,a) = \begin{cases} \{a, f\} & \text{if } q \in F, \\ \{a\} & \text{otherwise.} \end{cases}$$

– The sink state would be labeled by all propositions for completeness of the MDP.

$$L(q_{sink}) = \Sigma$$

HyperPCTL formula: The HyperPCTL formula in our mapping is

$$\varphi_{\mathsf{map}} = \exists \hat{\sigma}(\hat{\mathcal{M}}).\exists \hat{s}(\hat{\mathcal{M}}^{\hat{\sigma}}).\forall \hat{s}'(\hat{\mathcal{M}}^{\hat{\sigma}}).\ init_{\hat{s}} \wedge$$
$$\Big(init_{\hat{s}'} \rightarrow \mathbb{P}\big(\Box \bigwedge_{a \in \mathsf{AP}\setminus\{f\}} (a_{\hat{s}} \leftrightarrow a_{\hat{s}'})\big) = 1\Big) \wedge \Big(\mathbb{P}\Big(\Diamond \mathbb{P}\big(\Box \mathbb{P}(\Diamond f_{\hat{s}}) = 1\big) = 1\Big) > 0\Big)$$

Intuitively, the formula along with the above described mapping, establishes connection between the PBA emptiness problem and HyperPCTL model checking problem for MDPs. In particular:

- The existence of a scheduler $\hat{\sigma}(\mathcal{M})$ in φ_{map} corresponds to the existence of a word w in $\mathcal{L}(\mathcal{P})$;
- Every run of the PBA must begin from the initial state $\iota_{init} \in Q$. Hence, when arguing about these runs in the mapped MDP, we have to only consider runs that begin from the states corresponding to this initial state in the PBA. In the first and the second conjunct of the above hyperproperty, the condition to check for states labeled *init* serves this purpose.
- The state quantifiers along with the global fragment in the second conjunct ensure that all the paths in the induced DTMC and the PBA, follow the same sequence of actions (respectively, letters) in the witness to $\hat{\sigma}(\mathcal{M})$ (respectively, w). Note that, in the second conjunct, we enforce that the paths between the PBA and the MDP should match globally with probability one. If our PBA is incomplete, this fragment of the property would fail due to lack of enabled actions at certain states of the mapped MDP. To avoid this situation, we add transitions corresponding to these unenabled actions in the MDP and redirect them to a sink state q_{sink}.
- The third conjunct mimics that a state in F is visited with non-zero probability if and only if a state labeled by proposition f is visited infinitely often in the MDP with non-zero probability.

Reduction We now show that $\mathcal{L}(\mathcal{P}) \neq \emptyset$ if and only if $\mathcal{M} \models \varphi_{\mathsf{map}}$.

($\rightarrow$) Suppose we have $\mathcal{L}(\mathcal{P}) \neq \emptyset$. This means that there exists a word $w \in \Sigma^\omega$, such that $\Pr_{\mathcal{P}}(w) > 0$. We use w to eliminate the existential scheduler quantifier and instantiate $\hat{\sigma}(\hat{\mathcal{M}})$ in formula φ_{map}. This induces a DTMC and now, we show that the induced DTMC satisfies the following HyperPCTL formula as prescribed in [7]:

$$\exists \hat{s}(\hat{\mathcal{M}}^{\hat{\sigma}}).\forall \hat{s}'(\hat{\mathcal{M}}^{\hat{\sigma}}).\ init_{\hat{s}} \ \wedge \ \Big(init_{\hat{s}'} \rightarrow \ \mathbb{P}\big(\Box \bigwedge_{a \in \mathsf{AP}\setminus\{f\}} (a_{\hat{s}} \leftrightarrow a_{\hat{s}'})\big) = 1 \Big) \wedge$$
$$\Big(\mathbb{P}\Big(\Diamond \mathbb{P}\big(\Box \mathbb{P}(\Diamond f_{\hat{s}}) = 1\big) = 1 \Big) > 0 \Big)$$

To this end, observe that the third conjunct is trivially satisfied due to the fact that $\Pr_{\mathcal{P}}(w) > 0$, i.e., since a state in F is visited infinitely often with non-zero probability in $\mathcal{P}$, a state labeled by f in $\mathcal{M}$ is also visited infinitely often with non-zero probability. The first and second conjuncts are satisfied by construction of the mapped MDP, since the sequence of letters in w appear in all paths of the induced DTMC as propositions, when we start from the specific initial states.

($\leftarrow$) The reverse direction is pretty similar. Since the answer to the model checking problem is affirmative, a witness to scheduler quantifier $\hat{\sigma}$ exists. This scheduler induces a DTMC whose paths follow the same sequence of propositions. This sequence indeed provides us with the word w for $\mathcal{P}$. Finally, since the third conjunct in φ_{map} is satisfied by the MDP, we are guaranteed that w reaches an accepting state in F infinitely often with non-zero probability.

And this concludes the proof.

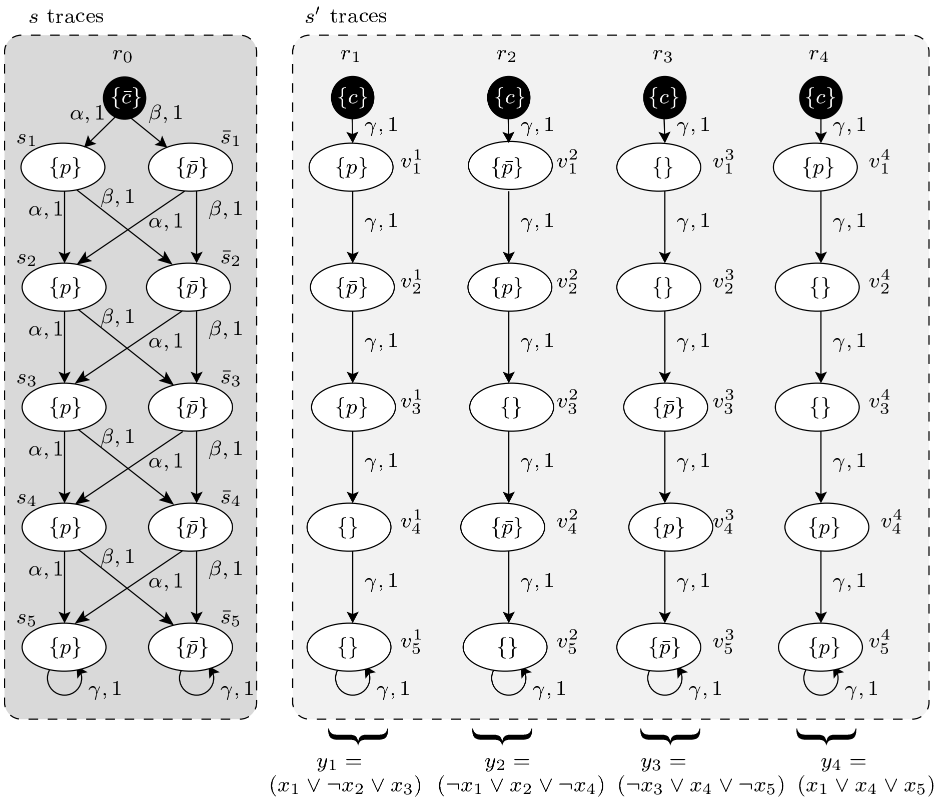
5.2. Decidability for Non-probabilistic Memoryless Schedulers

Due to the undecidability of HyperPCTL formulas for MDPs, we focus in this section on an alternative semantics, where scheduler quantification ranges over non-probabilistic memoryless schedulers only. It is easy to see that limiting ourselves to non-probabilistic memoryless schedulers makes the model checking problem decidable, as there are only finitely many such schedulers. Regarding complexity, we have the following property.

Theorem 2. *The problem to decide for MDPs the truth of HyperPCTL formulas with a single existential (respectively, universal) scheduler quantifier over non-probabilistic memoryless schedulers is NP-complete (respectively, coNP-complete) in the state set size of the given MDP.*

In order to show membership to NP, let $\mathcal{M}$ be an MDP and $\varphi = \exists\hat{\sigma}(\hat{\mathcal{M}}).\varphi'$ be a HyperPCTL formula, where φ' is a state quantified formula. We show that given a solution to the problem, we can verify the solution in polynomial time. Observe that given a non-probabilistic memoryless scheduler as a witness to the existential quantifier $\exists\hat{\sigma}(\hat{\mathcal{M}})$, one can compute the induced DTMC and then verify the DTMC against the resulting HyperPCTL formula in polynomial time in the size of the induced DTMC [7].

Inspired by the proof technique introduced in [21], for the lower bound, we reduce the SAT problem to our model checking problem. The SAT problem is as follows:



s traces
s' traces
r_0
$\{\bar{c}\}$
$\alpha, 1$
$\beta, 1$
s_1 $\{p\}$
$\bar{s}_1$ $\{\bar{p}\}$
s_2 $\{p\}$
$\bar{s}_2$ $\{\bar{p}\}$
s_3 $\{p\}$
$\bar{s}_3$ $\{\bar{p}\}$
s_4 $\{p\}$
$\bar{s}_4$ $\{\bar{p}\}$
s_5 $\{p\}$
$\bar{s}_5$ $\{\bar{p}\}$
$\gamma, 1$
r_1 r_2 r_3 r_4
$\{c\}$
v_1^1 $\{p\}$, v_2^1 $\{\bar{p}\}$, v_3^1 $\{p\}$, v_4^1 $\{\}$, v_5^1 $\{\}$
v_1^2 $\{\bar{p}\}$, v_2^2 $\{p\}$, v_3^2 $\{\}$, v_4^2 $\{\bar{p}\}$, v_5^2 $\{\}$
v_1^3 $\{\}$, v_2^3 $\{\}$, v_3^3 $\{\bar{p}\}$, v_4^3 $\{p\}$, v_5^3 $\{\bar{p}\}$
v_1^4 $\{p\}$, v_2^4 $\{\}$, v_3^4 $\{\}$, v_4^4 $\{p\}$, v_5^4 $\{p\}$
$y_1 = (x_1 \vee \neg x_2 \vee x_3)$
$y_2 = (\neg x_1 \vee x_2 \vee \neg x_4)$
$y_3 = (\neg x_3 \vee x_4 \vee \neg x_5)$
$y_4 = (x_1 \vee x_4 \vee x_5)$

Figure 8: Example of mapping SAT to HyperPCTL model checking.

> Let $y = y_1 \wedge y_2 \wedge \cdots \wedge y_m$ be a Boolean formula where each y_j, for $j \in [1, m]$, is a disjunction of at least three literals using propositions $\{x_1, x_2, \ldots, x_n\}$. Is y satisfiable, i.e., is there an assignment of truth values to $x_1, x_2, \ldots, x_n$, such that y evaluates to true?

Mapping We now present a mapping from an arbitrary SAT problem instance to the model checking problem of an MDP and a HyperPCTL formula of the form $\exists\hat{\sigma}(\hat{\mathcal{M}}).\ \exists\hat{s}(\hat{\mathcal{M}}^{\hat{\sigma}}).\ \forall\hat{s}'(\hat{\mathcal{M}}^{\hat{\sigma}}).\ \varphi$. Then, we show that the MDP satisfies this formula if and only if the answer to the SAT problem is affirmative. Fig. 8 shows an example.

MDP: For a given propositional logic formula in conjunctive normal form, we define the MDP $\mathcal{M} = (S, Act, \mathbf{P}, \mathsf{AP}, L)$ as follows.

- *(Atomic propositions* AP*)* We include four atomic propositions: p and $\bar{p}$ to mark the positive and negative literals in each clause and c and $\bar{c}$ to mark paths that correspond to clauses of the SAT formula. Thus, $\mathsf{AP} = \{p, \bar{p}, c, \bar{c}\}$.
- *(Set of states S)* We now identify the members of S:

– For each clause y_j, where $j \in [1, m]$, we include a state r_j, labeled by proposition c. We also include a state r_0 labeled by $\bar{c}$.

  - For each clause y_j, $j \in [1, m]$, we introduce n states $\left\{v_i^j \mid i \in [1, n]\right\}$. Each state v_i^j is labeled with proposition p if x_i is a literal in y_j, or with $\bar{p}$ if $\neg x_i$ is a literal in y_j.
  - For each Boolean variable x_i, where $i \in [1, n]$, we include two states: a state s_i labeled with p and a state $\bar{s}_i$ labeled with $\bar{p}$.

- *(Set of actions Act)* The set of actions is $Act = \{\alpha, \beta, \gamma\}$. Intuitively, the scheduler chooses action α (respectively, β) at a state s_i or $\bar{s}_i$ to assign true (respectively, false) to variable x_{i+1}. Action γ is the sole action available at all other states.
- *(Transition probability function* $\mathbf{P}$*)* We now identify the members of P. All transitions have probability 1, so we only discuss the actions.
  - We add transitions (r_j, γ, v_1^j) for each $j \in [1, m]$, where from r_j, the probability of reaching v_1^j is 1.
  - We also add transitions $(v_i^j, \gamma, v_{i+1}^j)$ for each $i \in [1, n)$, connecting the states representing literals in each clause y_j, $j \in [1, m]$.
  - For each $i \in [1, n)$, we include four transitions (s_i, α, s_{i+1}), $(s_i, \beta, \bar{s}_{i+1})$, $(\bar{s}_i, \alpha, s_{i+1})$, and $(\bar{s}_i, \beta, \bar{s}_{i+1})$. The intuition here is that when the scheduler chooses action α at state s_i or $\bar{s}_i$, variable x_{i+1} evaluates to true and when the scheduler chooses action β at state s_i or $\bar{s}_i$, variable x_{i+1} evaluates to false in the SAT instance. We also include two transitions (r_0, α, s_1) and $(r_0, \beta, \bar{s}_1)$ with the same intended meaning.
  - Finally, we include self-loops (s_n, γ, s_n), $(\bar{s}_n, \gamma, \bar{s}_n)$, and (v_n^j, γ, v_n^j), for each $j \in [1, m]$.

HyperPCTL formula: The $\mathsf{HyperPCTL}$ formula in our mapping is:

$$\varphi_{\mathsf{map}} = \exists \hat{\sigma}(\hat{\mathcal{M}}).\exists \hat{s}(\hat{\mathcal{M}}^{\hat{\sigma}}).\forall \hat{s}'(\hat{\mathcal{M}}^{\hat{\sigma}}).\ \bar{c}_{\hat{s}} \wedge \left(c_{\hat{s}'} \rightarrow \mathbb{P}\left(\Diamond\left((p_{\hat{s}} \wedge p_{\hat{s}'}) \vee (\bar{p}_{\hat{s}} \wedge \bar{p}_{\hat{s}'})\right)\right) = 1\right)$$

The intended meaning of the formula is that if there exists a scheduler that makes the formula true by choosing the α and β actions, this scheduler gives us the assignment to the Boolean variables in the SAT instance. This is achieved by making all clauses true, hence, the $\forall \hat{s}'(\hat{\mathcal{M}}^{\hat{\sigma}})$ subformula.

Reduction We now show that the given SAT formula is satisfiable if and only if the MDP obtained by our mapping satisfies the $\mathsf{HyperPCTL}$ formula φ_{map}.

($\rightarrow$) Suppose that y is satisfiable. Then, there is an assignment that makes each clause y_j, where $j \in [1, m]$, true. We now use this assignment to instantiate a scheduler for the formula φ_{map}. If $x_i = \texttt{true}$, then we instantiate scheduler $\hat{\sigma}$ such that in state s_{i-1} or $\bar{s}_{i-1}$, it chooses action α. Likewise, if $x_i = \texttt{false}$, then we instantiate scheduler $\hat{\sigma}$, such that in state s_{i-1} or $\bar{s}_{i-1}$, it chooses action β. We now show that this scheduler instantiation evaluates formula φ_{map} to true. First observe that $\hat{s}(\hat{\mathcal{M}}^{\hat{\sigma}})$ can only be instantiated with state r_0 and $\hat{s}'(\hat{\mathcal{M}}^{\hat{\sigma}})$

can only be instantiated with states r_j, where $j \in [1, m]$. Otherwise, the left side of the implication in φ_{map} becomes false, making the formula vacuously true. Since each y_j is true, there is at least one literal in y_j that is true. If this literal is of the form x_i, then we have $x_i = \texttt{true}$ and the path that starts from r_0 will include s_i, which is labeled by p. Hence, the values of p, in both paths that start from $\hat{s}(\hat{\mathcal{M}}^{\hat{\sigma}})$ and $\hat{s}'(\hat{\mathcal{M}}^{\hat{\sigma}})$ are eventually equal. If the literal in y_j is of the form $\neg x_i$, then $x_i = \texttt{false}$ and the path that starts from $\hat{s}(\hat{\mathcal{M}}^{\hat{\sigma}})$ will include $\bar{s}_i$. Again, the values of $\bar{p}$ are eventually equal. Finally, since all clauses are true, all paths that start from $\hat{s}'(\hat{\mathcal{M}}^{\hat{\sigma}})$ reach a state where the right side of the implication becomes true.

($\leftarrow$) Suppose our mapped MDP satisfies formula φ_{map}. This means that there exists a scheduler and state $\hat{s}(\hat{\mathcal{M}}^{\hat{\sigma}})$ that makes the subformula $\forall \hat{s}'(\hat{\mathcal{M}}^{\hat{\sigma}})$ true, i.e., since $\hat{s}$ can uniquely be instantiated by r_0 due to its labeling by $\bar{c}$, the path that starts from r_0 results in making the inner PCTL formula true for all paths that start from r_j, where $1 \leq j \leq m$, as the left of the implication is false for all other states. We obtain the truth assignment to the SAT problem as follows. If the scheduler chooses action α at state s_i, then we assign $x_i = \texttt{true}$. Likewise, if the scheduler chooses action β at state $\bar{s}_i$, then we assign $x_i = \texttt{false}$. Observe that since in no state p and $\bar{p}$ are simultaneously true and no path includes both s_i and $\bar{s}_i$, variable x_i will have only one truth value. Similar to the forward direction, it is straightforward to see that this valuation makes every clause y_j of the SAT instance true.

This concludes the proof. We note that in [7], we showed that HyperPCTL model checking for DTMCs is PSPACE-hard in the size of the input *formula*. Observe that our NP-completeness result in Theorem 2 is in the size of the input MDP. These results are not contradicting each other.

6. HyperPCTL Model Checking for Non-probabilistic Memoryless Schedulers

Next, we describe our SMT-based technique for solving the model checking problem for non-probabilistic memoryless scheduler domains, and for the simplified case of having a single scheduler quantifier; the general case for an arbitrary number of scheduler quantifiers is similar, but a bit more involved, so the simplified setting might be more suitable for understanding the basic ideas.

6.1. The Model Checking Algorithm

The main method listed in Algorithm 1 constructs a formula E that is satisfiable if and only if the input MDP $\mathcal{M}$ satisfies the input HyperPCTL formula with a single scheduler quantifier over the non-probabilistic memoryless scheduler domain. In line 2 we encode possible instantiations σ for the scheduler variable $\hat{\sigma}$. We use a variable σ_s for each MDP state $s \in S$ to encode which action is chosen in that state. Let us first deal with the case that the scheduler quantifier is *existential*. In line 5 we encode the meaning of the quantifier-free

Algorithm 1: Main SMT encoding algorithm

```
Input  : $\mathcal{M}=(S, Act, \mathbf{P}, \mathsf{AP}, L)$:  model specification;
         $\varphi^{sch} = Q\hat{\sigma}(\hat{\mathcal{M}}).Q_1\hat{s}_1(\hat{\mathcal{M}}^{\hat{\sigma}}).\dots Q_n\hat{s}_n(\hat{\mathcal{M}}^{\hat{\sigma}}).\varphi^{nq}$: HyperPCTL formula.
Output: Whether $\mathcal{M} \models \varphi^{sch}$.
1  Function $Main(\varphi^{sch}, \mathcal{M})$
2      $E := \bigwedge_{s\in S}(\bigvee_{\alpha\in Act(s)} \sigma_s = \alpha)$; // scheduler choice
3      if $Q$ is existential then
4          $E := E \wedge \text{Truth}(\mathcal{M},\ \exists\hat{\sigma}(\hat{\mathcal{M}}).\ Q_1\hat{s}_1(\hat{\mathcal{M}}^{\hat{\sigma}}).\dots Q_n\hat{s}_n(\hat{\mathcal{M}}^{\hat{\sigma}}).\ \varphi^{nq})$
5          $E := E \wedge \text{Semantics}(\mathcal{M}, \varphi^{nq}, n)$
6          if check($E$) = SAT then return $TRUE$ else return $FALSE$
7      else if $Q$ is universal then
           // $\overline{Q}_i$ is $\forall$ if $Q_i = \exists$ and $\exists$ else
8          $E := E \wedge \text{Truth}(\mathcal{M},\ \exists\hat{\sigma}(\hat{\mathcal{M}}).\overline{Q}_1\hat{s}_1(\hat{\mathcal{M}}^{\hat{\sigma}}).\dots\overline{Q}_n\hat{s}_n(\hat{\mathcal{M}}^{\hat{\sigma}}).\neg\varphi^{nq})$
9          $E := E \wedge \text{Semantics}(\mathcal{M}, \neg\varphi^{nq}, n)$
10         if check($E$) = SAT then return $FALSE$ else return $TRUE$
```

inner part φ^{nq} of the input formula, whereas line 4 encodes the meaning of the state quantifiers, i.e., for which sets of composed states φ^{nq} needs to hold in order to satisfy the input formula. In lines 6 we check the satisfiability of the encoding and return the corresponding answer. Formulas with a *universal* scheduler quantifier $\forall\hat{\sigma}(\hat{\mathcal{M}}).\varphi$ are semantically equivalent to $\neg\exists\hat{\sigma}(\hat{\mathcal{M}}).\neg\varphi$. We make use of this fact in lines 7–10 to check, first the satisfaction of an encoding for $\exists\hat{\sigma}(\hat{\mathcal{M}}).\neg\varphi$ and then return the inverted answer.

The Semantics method, shown in Algorithm 2, applies structural recursion to encode the meaning of the quantifier-free part of the input formula. As variables, the encoding uses (1) propositions $isTrue_{\vec{s},\varphi^{nq}} \in \{\texttt{true}, \texttt{false}\}$ to encode the truth of each Boolean subformula φ^{nq} of the input formula in each state $\vec{s} \in S^n$ of the parallel composition of n copies of $\mathcal{M}$, (2) numeric variables $pr_{\vec{s},\varphi^{pr}} \in [0,1] \subseteq \mathbb{R}$ to encode the value of each probability expression φ^{pr} in the input formula in the context of each composed state $\vec{s} \in S^n$, (3) variables $boolToInt_{\vec{s},\varphi^{pr}} \in \{0,1\}$ to encode truth values in a pseudo-Boolean form, i.e., we set $boolToInt_{\vec{s},\varphi^{pr}} = 1$ if $isTrue_{\vec{s},\varphi^{nq}} = \texttt{true}$ and $pr_{\vec{s},\varphi^{pr}} = 0$ otherwise, and (4) variables $d_{\vec{s},\varphi}$ to encode the existence of a loop-free path from state $\vec{s}$ to a state satisfying φ.

In Algorithm 2, there are two base cases: the Boolean constant `true` holds in all states (line 3), whereas atomic propositions hold in exactly those states that are labeled by them (line 4). For conjunction (line 8) we recursively encode the truth values of the operands and state that the conjunction is true if and only if both operands are true. For negation (line 14) we again encode the meaning of the operand recursively and flip its truth value. For the comparison of two probability expressions (line 19), we recursively encode the probability values of the operands and state the respective relation between them for satisfaction of the comparison. The remaining cases encode the semantics of probability expressions. The cases for constants (line 34) and arithmetic operations (line 35)

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

```
Input  : $\mathcal{M} = (S, Act, \mathbf{P}, \mathsf{AP}, L)$: MDP;  $\varphi$: quantifier-free HyperPCTL
         construct;  $n$: number of state variables in $\varphi$.
Output: SMT encoding of the meaning of $\varphi$ under $\mathcal{M}$.
Function $Semantics(\mathcal{M}, \varphi, n)$
    $\overrightarrow{r_Q} := \{\}$;   $\overrightarrow{r} := [s_1]^n$
    if $\varphi$ is true then $E := isTrue_{\overrightarrow{r},\varphi}$
    else if $\varphi$ is $a_{\hat{s}_i}$ then
        $\overrightarrow{r_Q} := \overrightarrow{r_Q} \cup \{i\}$
        $S^n_{rel} :=$ ComputeRelevantStatesSet$(S, n, \overrightarrow{r_Q})$
        $E := (\bigwedge_{\vec{s}\in S^n_{rel},\ a\in L(s_i)}(isTrue_{\vec{s},\varphi})) \wedge (\bigwedge_{\vec{s}\in S^n_{rel},\ a\notin L(s_i)}(\neg isTrue_{\vec{s},\varphi}))$
    else if $\varphi$ is $\varphi_1 \wedge \varphi_2$ then
        $E_{\varphi_1}, \overrightarrow{r_{\varphi_1}} :=$ Semantics$(\mathcal{M}, \varphi_1, n)$; $E_{\varphi_2}, \overrightarrow{r_{\varphi_2}} :=$ Semantics$(\mathcal{M}, \varphi_2, n)$
        $\overrightarrow{r_Q} := \overrightarrow{r_Q} \cup \overrightarrow{r_{\varphi_1}} \cup \overrightarrow{r_{\varphi_2}}$
        $S^n_{rel} :=$ ComputeRelevantStatesSet$(S, n, \overrightarrow{r_Q})$
        $E := E_{\varphi_1} \wedge E_{\varphi_2} \wedge \bigwedge_{\vec{s}\in S^n_{rel}}((isTrue_{\vec{s},\varphi} \wedge isTrue_{\vec{s},\varphi_1} \wedge isTrue_{\vec{s},\varphi_2}) \vee$
                $(\neg isTrue_{\vec{s},\varphi} \wedge (\neg isTrue_{\vec{s},\varphi_1} \vee \neg isTrue_{\vec{s},\varphi_2})))$
    else if $\varphi$ is $\neg\varphi'$ then
        $E_{\varphi'}, \overrightarrow{r_\varphi} :=$ Semantics$(\mathcal{M}, \varphi', n)$
        $\overrightarrow{r_Q} := \overrightarrow{r_Q} \cup \overrightarrow{r_\varphi}$
        $S^n_{rel} :=$ ComputeRelevantStatesSet$(S, n, \overrightarrow{r_Q})$
        $E := E_{\varphi'} \wedge \bigwedge_{\vec{s}\in S^n_{rel}}(isTrue_{\vec{s},\varphi} \oplus isTrue_{\vec{s},\varphi'})$
    else if $\varphi$ is $\varphi_1 < \varphi_2$ then
        $E_{\varphi_1}, \overrightarrow{r_{\varphi_1}} :=$ Semantics$(\mathcal{M}, \varphi_1, n)$; $E_{\varphi_2}, \overrightarrow{r_{\varphi_2}} :=$ Semantics$(\mathcal{M}, \varphi_2, n)$
        $\overrightarrow{r_Q} := \overrightarrow{r_Q} \cup \overrightarrow{r_{\varphi_1}} \cup \overrightarrow{r_{\varphi_2}}$
        $S^n_{rel} :=$ ComputeRelevantStatesSet$(S, n, \overrightarrow{r_Q})$
        $E := E_{\varphi_1} \wedge E_{\varphi_2} \wedge \bigwedge_{\vec{s}\in S^n_{rel}}[(isTrue_{\vec{s},\varphi} \wedge pr_{\vec{s},\varphi_1} < pr_{\vec{s},\varphi_2}) \vee$
                $(\neg isTrue_{\vec{s},\varphi} \wedge pr_{\vec{s},\varphi_1} \geq pr_{\vec{s},\varphi_2})]$
    else if $\varphi$ is $\mathbb{P}(\bigcirc\varphi')$ then
        $E, \overrightarrow{r_\varphi} :=$ SemanticsNext$(\mathcal{M}, \varphi, n)$
        $\overrightarrow{r_Q} := \overrightarrow{r_Q} \cup \overrightarrow{r_\varphi}$
    else if $\varphi$ is $\mathbb{P}(\varphi_1 \mathcal{U} \varphi_2)$ then
        $E, \overrightarrow{r_\varphi} :=$ SemanticsUnboundedUntil$(\mathcal{M}, \varphi, n)$
        $\overrightarrow{r_Q} := \overrightarrow{r_Q} \cup \overrightarrow{r_\varphi}$
    else if $\varphi$ is $\mathbb{P}(\varphi_1 \mathcal{U}^{[k_1,k_2]} \varphi_2)$ then
        $E, \overrightarrow{r_\varphi} :=$ SemanticsBoundedUntil$(\mathcal{M}, \varphi, n)$
        $\overrightarrow{r_Q} := \overrightarrow{r_Q} \cup \overrightarrow{r_\varphi}$
    else if $\varphi$ is $c$ then $E := (pr_{\overrightarrow{r},\varphi} = c)$
    else if $\varphi$ is $\varphi_1\ op\ \varphi_2$   /* $op \in \{+, -, *\}$ */   then
        $E_{\varphi_1}, \overrightarrow{r_{\varphi_1}} :=$ Semantics$(\mathcal{M}, \varphi_1, n)$; $E_{\varphi_2}, \overrightarrow{r_{\varphi_2}} :=$ Semantics$(\mathcal{M}, \varphi_2, n)$
        $\overrightarrow{r_Q} := \overrightarrow{r_Q} \cup \overrightarrow{r_{\varphi_1}} \cup \overrightarrow{r_{\varphi_2}}$
        $S^n_{rel} :=$ ComputeRelevantStatesSet$(S, n, \overrightarrow{r_Q})$
        $E := E_{\varphi_1} \wedge E_{\varphi_2} \wedge \bigwedge_{\vec{s}\in S^n_{rel}}(pr_{\vec{s},\varphi} = (pr_{\vec{s},\varphi_1}\ op\ pr_{\vec{s},\varphi_2}))$
    return $E$, $\overrightarrow{r_Q}$
```

are straightforward. For the probability $\mathbb{P}(\bigcirc\varphi')$ (line 25 and Algorithm 3), we encode the Boolean value of φ' in the variables $\mathit{isTrue}_{\vec{s},\varphi'}$ (line 2), turn them into pseudo-Boolean values $\mathit{boolToInt}_{\vec{s},\varphi'}$ (1 for true and 0 for false, line 6), and state that for each composed state, the probability value of $\mathbb{P}(\bigcirc\varphi')$ is the sum of the probabilities to get to a successor state where the operand φ' holds; since the successors and their probabilities are scheduler-dependent, we need to iterate over all scheduler choices and use $\mathit{supp}(\alpha_i)$ to denote the support $\{s \in S \mid \alpha_i(s) > 0\}$ of the distribution α_i (line 7).

For the probability $\mathbb{P}(\varphi_1 \,\mathcal{U}\, \varphi_2)$ to satisfy an unbounded until formula, the method SemanticsUnboundedUntil shown in Algorithm 4 first encodes the meaning of the until operands (line 6). For each composed state $\vec{s} \in S^n$, the probability of satisfying the until formula in $\vec{s}$ is encoded in the variable $\mathit{pr}_{\vec{s},\mathbb{P}(\varphi_1 \,\mathcal{U}\, \varphi_2)}$. If the second until-operand φ_2 holds in $\vec{s}$ then this probability is 1 and if none of the operands are true in $\vec{s}$ then it is 0 (line 9). Otherwise, depending on the scheduler σ of $\mathcal{M}$ (line 10), the value of $\mathit{pr}_{\vec{s},\mathbb{P}(\varphi_1 \,\mathcal{U}\, \varphi_2)}$ is a sum, adding up for each successor state $\vec{s}'$ of $\vec{s}$, the probability to get from $\vec{s}$ to $\vec{s}'$ in one step times the probability to satisfy the until-formula on paths starting in $\vec{s}'$ (line 12). However, these encodings work only when at least one state satisfying φ_2 is reachable from $\vec{s}$ with a positive probability: for any bottom strongly connected component (SCC) whose states all violate φ_2, the probability $\mathbb{P}(\varphi_1 \,\mathcal{U}\, \varphi_2)$ is 0. However, assigning any fixed value from $[0, 1]$ to all states of this bottom SCC would yield a fixed-point for the underlying equation system. To ensure correctness, in line 13 we enforce smallest fixed-points by requiring that if $\mathit{pr}_{\vec{s},\mathbb{P}(\varphi_1 \,\mathcal{U}\, \varphi_2)}$ is positive then there exists a loop-free path from $\vec{s}$ to any state satisfying φ_2. In the encoding of this property we use fresh variables $d_{\vec{s},\varphi_2}$ and require a path over states with strong monotonically decreasing $d_{\vec{s},\varphi_2}$-values to a φ_2-state (where the decreasing property serves to exclude loops). The domain of the distance-variables $d_{\vec{s},\varphi_2}$ can be integers, rationals or reals; the only restriction is that it should contain at least $|S|^n$ ordered values. Especially, it does not need to be lower bounded (note that each solution assigns to $d_{\vec{s},\varphi_2}$ a fixed value, leading to a finite number of distance values).

The SemanticsBoundedUntil method, listed in Algorithm 5, encodes the probability $\mathbb{P}(\varphi_1 \,\mathcal{U}^{[k_1,k_2]} \varphi_2)$ of a bounded until formula in the numeric variables $\mathit{pr}_{\vec{s},\mathbb{P}(\varphi_1 \,\mathcal{U}^{[k_1,k_2]} \varphi_2)}$ for all (composed) states $\vec{s} \in S^n$ and recursively reduced time bounds. There are three main cases: (i) the satisfaction of $\varphi_1 \,\mathcal{U}^{[0,k_2-1]} \varphi_2$ requires to satisfy φ_2 immediately (lines 2–8); (ii) $\varphi_1 \,\mathcal{U}^{[0,k_2-1]} \varphi_2$ can be satisfied by either satisfying φ_2 immediately or satisfying it later, but in the latter case φ_1 needs to hold currently (lines 9–17); (iii) φ_1 has to hold and φ_2 needs to be satisfied some time later (line 18–25). To avoid the repeated encoding of the semantics of the operands, we do it only when we reach case (i) where recursion stops (line 6). For the other cases, we recursively encode the probability to reach a φ_2-state over φ_1 states where the deadlines are reduced by one step (lines 10 resp. 19) and use these to fix the values of the variables $\mathit{pr}_{\vec{s},\mathbb{P}(\varphi_1 \,\mathcal{U}^{[k_1,k_2]} \varphi_2)}$, similar to the unbounded case but under additional consideration of time bounds.

The Truth method listed in Algorithm 6 encodes the meaning of state quan-

Algorithm 3: SMT encoding for the meaning of next formulas

```
Input  : $\mathcal{M} = (S, Act, \mathbf{P}, \mathsf{AP}, L)$: MDP;  $\varphi = \mathbb{P}(\bigcirc \varphi')$: HyperPCTL formula;
         $n$: number of state variables in $\varphi$.
Output: SMT encoding of $\varphi$'s meaning under $\mathcal{M}$.
Function SemanticsNext($\mathcal{M}, \varphi, n$)
    $E, \overrightarrow{r_Q}$ := Semantics($\mathcal{M}, \varphi', n$)
    $S^n_{rel}$ := ComputeRelevantStatesSet($S, n, \overrightarrow{r_Q}$)
    foreach $\vec{s} = (s_1, \ldots, s_n) \in S^n_{rel}$ do
        $E := E \wedge$
        $\big((\mathit{boolToInt}_{\vec{s},\varphi'} = 1 \wedge \mathit{isTrue}_{\vec{s},\varphi'}) \vee (\mathit{boolToInt}_{\vec{s},\varphi'} = 0 \wedge \neg \mathit{isTrue}_{\vec{s},\varphi'})\big)$
        foreach $\vec{\alpha} = (\alpha_1, \ldots, \alpha_n) \in Act(s_1) \times \ldots \times Act(s_n)$ do
            $E := E \wedge \big(\big[\bigwedge_{i=1}^n \sigma_{s_i} = \alpha_i\big] \rightarrow \big[\mathit{pr}_{\vec{s},\varphi} =$
            $\sum_{\vec{s}' \in supp(\alpha_1) \times \ldots \times supp(\alpha_n)} ((\Pi_{i=1}^n \mathbf{P}(s_i, \alpha_i, s'_i)) \cdot \mathit{boolToInt}_{\vec{s}',\varphi'})\big]\big)$
    return $E, \overrightarrow{r_Q}$
```

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

```
Input  : $\mathcal{M} = (S, Act, \mathbf{P}, \mathsf{AP}, L)$: MDP;  $\varphi = \mathbb{P}(\varphi_1 \mathcal{U} \varphi_2)$: HyperPCTL
         formula;  $n$: number of state variables in $\varphi$.
Output: SMT encoding of $\varphi$'s meaning under $\mathcal{M}$.
Function SemanticsUnboundedUntil($\mathcal{M}, \varphi, n$)
    $E_{\varphi_1}, \overrightarrow{r_{\varphi_1}}$ := Semantics($\mathcal{M}, \varphi_1, n$)
    $E_{\varphi_2}, \overrightarrow{r_{\varphi_2}}$ := Semantics($\mathcal{M}, \varphi_2, n$)
    $\overrightarrow{r_Q} := \overrightarrow{r_{\varphi_1}} \cup \overrightarrow{r_{\varphi_2}}$
    $S^n_{rel}$ := ComputeRelevantStatesSet($S, n, \overrightarrow{r_Q}$)
    $E := E_{\varphi_1} \wedge E_{\varphi_2}$;
    foreach $\vec{s} = (s_1, \ldots, s_n) \in S^n_{rel}$ do
        $E := E \wedge (\mathit{isTrue}_{\vec{s},\varphi_2} \rightarrow \mathit{pr}_{\vec{s},\varphi}{=}1) \wedge$
                    $((\neg \mathit{isTrue}_{\vec{s},\varphi_1} \wedge \neg \mathit{isTrue}_{\vec{s},\varphi_2}) \rightarrow \mathit{pr}_{\vec{s},\varphi}{=}0)$;
        foreach $\vec{\alpha} = (\alpha_1, \ldots, \alpha_n) \in Act(s_1) \times \ldots \times Act(s_n)$ do
            $E := E \wedge \Big(\big[\mathit{isTrue}_{\vec{s},\varphi_1} \wedge \neg \mathit{isTrue}_{\vec{s},\varphi_2} \wedge \bigwedge_{i=1}^n \sigma_{s_i} = \alpha_i\big] \rightarrow$
            $\big[\mathit{pr}_{\vec{s},\varphi} = \sum_{\vec{s}' \in supp(\alpha_1) \times \ldots \times supp(\alpha_n)} ((\Pi_{i=1}^n \mathbf{P}(s_i, \alpha_i, s'_i)) \cdot \mathit{pr}_{\vec{s}',\varphi}) \wedge$
            $(\mathit{pr}_{\vec{s},\varphi} > 0 \rightarrow$
              $(\bigvee_{\vec{s}' \in supp(\alpha_1) \times \ldots \times supp(\alpha_n)} (\mathit{isTrue}_{\vec{s}',\varphi_2} \vee d_{\vec{s},\varphi_2} > d_{\vec{s}',\varphi_2})))\big]\Big)$
    return $E, \overrightarrow{r_Q}$
```

tification: it states for each universal quantifier that instantiating it with any MDP state should satisfy the formula (conjunction over all states in line 3), and for each existential state quantification that at least one state should lead to satisfaction (disjunction in line 3).

Optimization We have added an important optimization to the solution presented in [13]. Our earlier algorithm encoded, for each state in the parallel composition of the models, the semantical value of each subformula. However,

Algorithm 5: SMT encoding for the meaning of bounded until formulas

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

Output: SMT encoding of φ's meaning under $\mathcal{M}$.

```
1  Function SemanticsBoundedUntil($\mathcal{M}, \varphi, n$)
2      if $k_2 = 0$ then
3          $E_{\varphi_1}, \overrightarrow{r_{\varphi_1}}$ := Semantics($\mathcal{M}, \varphi_1, n$); $E_{\varphi_2}, \overrightarrow{r_{\varphi_2}}$ := Semantics($\mathcal{M}, \varphi_2, n$)
4          $\overrightarrow{r_Q} := \overrightarrow{r_{\varphi_1}} \cup \overrightarrow{r_{\varphi_2}}$
5          $S^n_{rel}$ := ComputeRelevantStatesSet($S, n, \overrightarrow{r_Q}$)
6          $E := E_{\varphi_1} \wedge E_{\varphi_2}$
7          foreach $\vec{s} = (s_1, \ldots, s_n) \in S^n_{rel}$ do
8              $E := E \wedge (isTrue_{\vec{s},\varphi_2} \rightarrow pr_{\vec{s},\varphi}=1) \wedge (\neg isTrue_{\vec{s},\varphi_2} \rightarrow pr_{\vec{s},\varphi}=0)$
9      else if $k_1 = 0$ then
10         $E, \overrightarrow{r_Q}$ := SemanticsBoundedUntil($\mathcal{M}$, $\mathbb{P}(\varphi_1 \,\mathcal{U}^{[0,k_2-1]} \varphi_2), n$)
11         $S^n_{rel}$ := ComputeRelevantStatesSet($S, n, \overrightarrow{r_Q}$)
12         foreach $\vec{s} = (s_1, \ldots, s_n) \in S^n_{rel}$ do
13             $E := E \wedge (isTrue_{\vec{s},\varphi_2} \rightarrow pr_{\vec{s},\varphi}=1) \wedge$
14                     $((\neg isTrue_{\vec{s},\varphi_1} \wedge \neg isTrue_{\vec{s},\varphi_2}) \rightarrow pr_{\vec{s},\varphi}=0)$
15             foreach $\vec{\alpha} = (\alpha_1, \ldots, \alpha_n) \in Act(s_1) \times \ldots \times Act(s_n)$ do
16                 $E := E \wedge \Big( \big[ isTrue_{\vec{s},\varphi_1} \wedge \neg isTrue_{\vec{s},\varphi_2} \wedge \bigwedge_{i=1}^{n} \sigma_{s_i}=\alpha_i \big] \rightarrow \big[ pr_{\vec{s},\varphi} =$
17                 $\sum_{\vec{s}' \in supp(\alpha_1) \times \ldots \times supp(\alpha_n)} ((\Pi_{i=1}^{n} \mathbf{P}(s_i, \alpha_i, s'_i)) \cdot pr_{\vec{s}', \mathbb{P}(\varphi_1 \mathcal{U}^{[0,k_2-1]} \varphi_2)}) \big] \Big)$;
18     else if $k_1 > 0$ then
19         $E, \overrightarrow{r_Q}$ := SemanticsBoundedUntil($\mathcal{M}$, $\mathbb{P}(\varphi_1 \,\mathcal{U}^{[k_1-1,k_2-1]} \varphi_2), n$)
20         $S^n_{rel}$ := ComputeRelevantStatesSet($S, n, \overrightarrow{r_Q}$)
21         foreach $\vec{s} = (s_1, \ldots, s_n) \in S^n_{rel}$ do
22             $E := E \wedge (\neg isTrue_{\vec{s},\varphi_1} \rightarrow pr_{\vec{s},\varphi} = 0)$
23             foreach $\vec{\alpha} = (\alpha_1, \ldots, \alpha_n) \in Act(s_1) \times \ldots \times Act(s_n)$ do
24                 $E := E \wedge \Big( \big[ isTrue_{\vec{s},\varphi_1} \wedge \bigwedge_{i=1}^{n} \sigma_{s_i} = \alpha_i \big] \rightarrow \big[ pr_{\vec{s},\varphi} =$
25                 $\sum_{\vec{s}' \in supp(\alpha_1) \times .. \times supp(\alpha_n)} ((\Pi_{i=1}^{n} \mathbf{P}(s_i, \alpha_i, s'_i)) \cdot pr_{\vec{s}', \mathbb{P}(\varphi_1 \mathcal{U}^{[k_1-1,k_2-1]} \varphi_2)}) \big] \Big)$
26     return $E, \overrightarrow{r_Q}$
```

these semantical values often do not depend on all the copies of the model involved, but only a subset of them. For example, in the context of the MDP described in Fig. 1b, let us consider the following formula,

$$\exists \hat{\sigma}(\hat{\mathcal{M}}).\forall \hat{s}(\hat{\mathcal{M}}^{\hat{\sigma}}).\exists \hat{s}'(\hat{\mathcal{M}}^{\hat{\sigma}}). \Big((h{>}0)_{\hat{s}} \wedge (h{\leq}0)_{\hat{s}'} \Big) \rightarrow \Big(\mathbb{P}(\Diamond (l{=}1)_{\hat{s}}) {=} \mathbb{P}(\Diamond (l{=}2)_{\hat{s}'}) \Big)$$

The evaluation composes two copies of $\mathcal{M}$, one for each state quantifier and computes recursively, the encoding for each state $(s, s') \in S \times S$ of the parallel composition. However, the values of the subformulas $(h{>}0)_{\hat{s}}$ and $(\mathbb{P}(\Diamond (l{=}1)_{\hat{s}})$ depend on the first state component s only; here we say that the first com-

Algorithm 6: SMT encoding of the truth of the input formula

```
Input  : $\mathcal{M} = (S, Act, \mathbf{P}, \mathsf{AP}, L)$: MDP;
         $\varphi^{sch} = \exists\hat{\sigma}(\hat{\mathcal{M}}).Q_1\hat{s}_1(\hat{\mathcal{M}}^{\hat{\sigma}}).\dots Q_n\hat{s}_n(\hat{\mathcal{M}}^{\hat{\sigma}}).\varphi^{nq}$: HyperPCTL
formula.
Output: Encoding of the truth of the input formula under $\mathcal{M}$.
1 Function $Truth(\mathcal{M}, \varphi^{sch})$
2     foreach $i = 1, \dots, n$ do
3         if $Q_i = \forall$ then $B_i :=$"$\bigwedge_{s_i \in S}$" else $B_i :=$"$\bigvee_{s_i \in S}$"
4     return $B_1 \ \dots \ B_n \ isTrue_{(s_1,\dots,s_n),\varphi^{nq}}$
```

Algorithm 7: Calculating set of relevant state combinations

```
Input  : $S$: set of states in the input MDP;  $n$: number of quantifiers in $\varphi^{sch}$;
         $rel_Q$: set of relevant quantifiers carried over from previous recursion.
Output: Set of state combinations relevant to the current subformula.
1 Function $ComputeRelevantStatesSet(S, n, rel_Q)$
2     $\overrightarrow{r} := [s_1]^n$; $\overrightarrow{index} := [1]^n$; $i := \|S\|$; $S^n_{rel} := \{\}$
3     while $i > 0$ do
4         $S^n_{rel} := S^n_{rel} \cup \overrightarrow{r}$
5         while $(i \geq 1)$ and $(index_i = n$ or $i \notin rel_Q)$ do
6             $r_i = s_1$
7             $index_i = 1$
8             $i = i - 1$
9         if $i > 0$ then
10            $index_i := index_i + 1$
11            $r_i := s_{index_i}$
12    return $S^n_{rel}$
```

ponent is *relevant*, whereas the second one is not. Similarly, for $(h{\leq}0)_{\hat{s}'}$ and $\mathbb{P}(\diamondsuit\,(l{=}2)_{\hat{s}'})$ only the second component s' is relevant but not the first. To reduce computational effort as well as memory usage, we do not encode the semantical values for all composed states, but limit the encoding to the parallel composition of the *relevant* components. Technically, we choose an arbitrary state $s^* \in S$ of $\mathcal{M}$ as a "placeholder" state for non-relevant components. Instead of encoding the semantical value of $(h{>}0)_{\hat{s}}$ for all $(s, s') \in S \times S$, we encode it only for each $(s) \in S \times \{s^*\}$. During recursion, it might happen that the semantical value of $h_{\hat{s}}$ is needed for some $(s, s') \in S \times S$ with $s' \neq s^*$; in this case we re-direct the query and get the value of (s, s^*). For example, let us consider the MDP in Fig. 1b with 4 states. In our previous approach to encode $(h > 0)_{\hat{s}}$, we would have considered all 16 state combinations $((s_0, s'_0), (s_0, s'_1), \dots, (s_1, s'_2), \dots, (s_3, s'_3))$. However, in our optimized approach, we consider only four state combinations $((s_0, s'_0), \dots, (s_3, s'_0))$. This optimization has a huge impact for large state spaces. For a multi-model system, with n and m states, our previous approach would need us to consider $(n + m)^2$ state combinations, whereas the current optimization considers only $(n \times m)$ state combinations.

Algorithm 7 describes how we avoid computing unwanted state combinations. State quantifiers $(Q_1,\ldots,Q_n)$ that are *relevant* to the subformula, are stored in the array named $\overrightarrow{r_Q}$. Here, $\overrightarrow{r}$ stores the current composed state and $\overrightarrow{index}$ keeps track of the indices of the states in $\overrightarrow{r}$. The enumeration works for quantifiers inside-out, i.e., for three quantifiers and a four-state DTMC, the enumeration is $(s_1, s_1', s_1''), \ldots, (s_1, s_1', s_4''), (s_1, s_2', s_1''), \ldots, (s_1, s_4', s_4''), \ldots, (s_4, s_4', s_4'')$. Our initial $\overrightarrow{r}$ consists of the first state, i.e., s_1 for all quantifiers. This initialization is done in line 2. We start increasing the index of the state associated with the innermost quantifier (lines 9-11). Once all states have been covered for that quantifier, in (lines 6-8), we reset the state index of the current quantifier to s_1. We also try to find the index of the next relevant quantifier closest to the current one, i.e., Q_j with highest j from right end of $Q_1,\ldots,Q_n$ and increase its index by one. This extra search helps us to store only the relevant state combinations in S^n_{rel}.

Theorem 3. *Algorithm 1 returns an encoding that is true if and only if its input HyperPCTL formula is satisfied by the input MDP.*

We note that the satisfiability of the generated SMT encoding for a formula with an existential scheduler quantifier does not only prove the truth of the formula but also provides a scheduler as witness, encoded in the solution of the SMT encoding. Conversely, unsatisfiability of the SMT encoding for a formula with a universal scheduler quantifier provides a counterexample scheduler.

6.2. Example of the Encoding

Consider the problem where we are trying to verify the following property on the MDP in Fig. 1b.

$$\exists\hat{\sigma}(\hat{\mathcal{M}}).\forall\hat{s}(\hat{\mathcal{M}}^{\hat{\sigma}}).\exists\hat{s}'(\hat{\mathcal{M}}^{\hat{\sigma}}).\ \Big((h{>}0)_{\hat{s}} \wedge (h{\le}0)_{\hat{s}'}\Big) \rightarrow \Big(\mathbb{P}(\Diamond\,(l{=}1)_{\hat{s}}) {=} \mathbb{P}(\Diamond\,(l{=}2)_{\hat{s}'})\Big)$$

We encode the actions in the MDP as described in line 2 of Algorithm 1. Here, σ_i refers to the action in s_i.

$$E_{sch} = (\sigma_0 = \alpha \vee \sigma_0 = \beta) \wedge (\sigma_1 = \alpha \vee \sigma_1 = \beta) \wedge (\sigma_2 = \tau) \wedge (\sigma_3 = \tau)$$

We handle the encoding of the state quantifiers using Algorithm 6. We use φ_{nq} to represent the quantifier-free part of the above property and $\mathit{isTrue}_{s_i,s_j,\varphi_{nq}}$ refers to the encoding to ensure φ_{nq} holds in the composed state of (s_i, s_j).

$$\begin{aligned} E_{truth} = (&\mathit{isTrue}_{s_0,s_0,\varphi_{nq}} \vee \ldots \vee \mathit{isTrue}_{s_0,s_3,\varphi_{nq}}) \wedge \\ &\ldots \wedge (\mathit{isTrue}_{s_3,s_0,\varphi_{nq}} \vee \ldots \vee \mathit{isTrue}_{s_3,s_3,\varphi_{nq}}) \end{aligned}$$

We handle atomic propositions as described in line 4 in Algorithm 2. For example, we have the encoding of $(h{>}0)_{\hat{s}}$ below. Please note here that the first quantifier is relevant and the second is not. Also, $(h{>}0)_{\hat{s}}$ is true only in s_0, hence we encode the atomic proposition with a negation for all other states.

$$E_{(h>0)_{\hat{s}}} = (\mathit{isTrue}_{\,s_0,s_0,(h>0)_{\hat{s}}}) \wedge (\neg \mathit{isTrue}_{\,s_1,s_0,(h>0)_{\hat{s}}} \wedge \ldots \wedge \neg \mathit{isTrue}_{\,s_3,s_0,(h>0)_{\hat{s}}})$$

To encode operators, we include both the satisfaction and dissatisfaction clauses in the encoding. Depending on the atomic propositions involved, one of the clauses would be satisfied. For example, the encoding for conjunction of $\Big((h{>}0)_{\hat{s}} \wedge (h{\le}0)_{\hat{s}'}\Big)$ for (s_0, s_0) would be as follows.

$$E_{conj} = (\mathit{isTrue}_{\,s_0,s_0,(h>0)_{\hat{s}}} \wedge \mathit{isTrue}_{\,s_0,s_0,(h\le 0)_{\hat{s}'}} \wedge \mathit{isTrue}_{\,s_0,s_0,(h>0)_{\hat{s}} \wedge (h\le 0)_{\hat{s}'}}) \vee$$
$$((\neg \mathit{isTrue}_{\,s_0,s_0,(h>0)_{\hat{s}}} \vee \neg \mathit{isTrue}_{\,s_0,s_0,(h\le 0)_{\hat{s}'}}) \wedge \neg \mathit{isTrue}_{\,s_0,s_0,(h>0)_{\hat{s}} \wedge (h\le 0)_{\hat{s}'}})$$

The encoding of $\Diamond$ is similar to Algorithm 4 except that we consider φ_1 to be `true` and ignore its encoding. For example, the encoding of $\mathbb{P}(\Diamond\, (l{=}1)_{\hat{s}})$ for (s_0, s_0) and action $(\alpha_{\hat{s}}, \alpha_{\hat{s}'})$ would be as below. Note that since the first quantifier is relevant, we will only encode (s_2, s_0) and (s_3, s_0) as the successor states.

$$\begin{aligned}
E_{\Diamond} = &(\mathit{isTrue}_{\,s_0,s_0,(l=1)_{\hat{s}}} \rightarrow pr_{\,s_0,s_0,\mathbb{P}(\Diamond\,(l=1)_{\hat{s}})} = 1) \wedge (pr_{\,s_0,s_0,\mathbb{P}(\Diamond\,(l=1)_{\hat{s}})} \ge 0) \wedge \\
&(\neg \mathit{isTrue}_{\,s_0,s_0,(l=1)_{\hat{s}}} \wedge \sigma_0 = \alpha \wedge \sigma_1 = \alpha) \rightarrow \Big(pr_{\,s_0,s_0,\mathbb{P}(\Diamond\,(l=1)_{\hat{s}})} = \\
&(3/4 \times 1 \times pr_{\,s_2,s_0,\mathbb{P}(\Diamond\,(l=1)_{\hat{s}})}) + (1/4 \times 1 \times pr_{\,s_3,s_0,\mathbb{P}(\Diamond\,(l=1)_{\hat{s}})})\Big) \wedge \\
&\Big(pr_{\,s_0,s_0,\mathbb{P}(\Diamond\,(l=1)_{\hat{s}})} > 0 \rightarrow (\mathit{isTrue}_{\,s_2,s_0,(l=1)_{\hat{s}}} \vee d_{s_0,s_0,(l=1)_{\hat{s}}} > d_{s_2,s_0,(l=1)_{\hat{s}}}) \\
&\vee (\mathit{isTrue}_{\,s_3,s_0,(l=1)_{\hat{s}}} \vee d_{s_0,s_0,(l=1)_{\hat{s}}} > d_{s_3,s_0,(l=1)_{\hat{s}}})\Big)
\end{aligned}$$

For Multiple Scheduler Quantifiers To extend this algorithm to work for multiple schedulers, we firstly have to encode the combination of actions for every state combination. Hence in line 2 of Algorithm 1, we will need n nested loops to encode all the possible scheduler choices of n scheduler quantifiers. In the rest of the algorithm, we have to do similar changes to account for the combinations of actions possible from each state $\vec{s} \in S^n$. So we will need similar nested loops in cases like line 10 of Algorithm 4.

7. Evaluation

7.1. Implementation

We have prototypically implemented our algorithm in Python, with the help of several libraries. We have initially parsed the hyperproperty into an abstract syntax tree which helped us to manage the precedence of operators. We have also used STORMPY [22, 23] to parse our model, given as a PRISM file. STORMPY provides an efficient solution to parsing, building, and storage of MDPs. Together, they are used to encode the SMT constraints for the scheduler actions available, the scheduler and state quantifiers, and finally, the semantics of the hyperproperty, according to our algorithm. We used the SMT-solver Z3 [24] to solve the logical encoding generated by Algorithm 1. As a result, we return a Boolean value representing the satisfaction or dissatisfaction of the formula. Additionally, we return a set of actions $\{\alpha_{s_1}, \ldots, \alpha_{s_n}\}$ for specific cases where additional information about the result can be given. In case the scheduler quantifier is $\forall$, and the hyperproperty is dissatisfied, the set of actions would induce a DTMC that is a counterexample. In case the scheduler quantifier is $\exists$, and the hyperproperty is satisfied, the set of actions would induce a DTMC that is a witness.

Exact computations For our experiments, we specified high-level models in PRISM language [25] and used STORMPY to parse them. STORMPY, by default, reads and stores the transition probabilities as float datatype, potentially losing accuracy. This caused erroneous results when comparing two such approximated values for equality. Hence, in [13], some of the reported results can be wrong due to this numerical bug in STORMPY. We came across this when scaling up our previous case studies. Therefore, after parsing, we stored the model details and utilized all its information except the transition probabilities, to rebuild the models with exact fractional probabilities in a rational format (introduced in the latest version of STORMPY), to ensure consistency throughout the implementation. The model rebuilding step has been automated and successfully incorporated into our implementation.

7.2. Experimental Data

All our experiments were run on a MacBook Pro laptop with a 2.3GHz i7 processor with 32GB of RAM. The results are presented in Table 3.

We used four benchmark families, whose complexities are indicated in Table 2, listing the depth of the property we have verified along with the number of states and transitions of the model. The depth of a property refers to the total number of atomic propositions and Boolean and temporal operations contained in the respective hyperproperty.

For the first case study, we modeled and analyzed information leakage in the modular exponentiation algorithm (function `modexp` in Fig. 5); the corresponding results in Table 3 are marked by **TA**. We experimented with 1, 2, 3, and 4 bits for the encryption key (hence, $m \in \{2, 4, 6, 8\}$). The specification checks for the absence of a timing channel for all possible schedulers, which is not the case for the implementation in `modexp`.

Our second case study **PW** is verification of password leakage through the string comparison algorithm (function `str_cmp` in Fig. 6). Here, we experimented with $m \in \{2, 4, 6, 8\}$.

In our third case study **TS**, we assume two concurrent processes. The first process decrements the value of a secret h by 1 as long as the value is still positive, and sets the low variable l to 1 outside the loop. A second process just sets the value of the same low variable l to 2. The two threads run in parallel until one of them terminates and a fair scheduler chooses the next executing thread for each CPU cycle. This opens a probabilistic thread scheduling channel and leaks the value of h. We compare observations for executions with different secret values h_1 and h_2 (denoted as $h = (h_1, h_2)$). There is an interesting relation between the data for **TS**. Both the encoding and running time for the experiment is proportional to the higher value in the tuple h. We believe this is because, while encoding, the parser has to traverse the maximum depth of the model corresponding to the higher value of h, irrespective of the lower value of h in the tuple.

Our last case study **PC** is on probabilistic conformance. The input is a DTMC modeling a fair 6-sided die and an MDP whose actions model single fair coin tosses with two successor states each. We are interested in finding a scheduler that induces a DTMC that simulates the die outcomes using a fair coin. Given a fixed state space, we experiment with different numbers of actions. In particular, we started from the implementation in [18] and for the state space of the coin section of the protocol, we added all the possible nondeterministic transitions from the first state to all the other states (denoted as s=0 in the data tables), from the first and second states to all the others (s=0..1), and, similarly scaled it step wise to include transitions from all states to all others (s=0..6). Each time, we were not only able to satisfy the formula, but also obtain the witness corresponding to the scheduler satisfying the property. A noteworthy observation in this experiment data is the fact that the number of variables created remains constant in spite of scaling the model up by introducing more transitions at each stage. This is attributed to the fact that we have a single variable representing the action chosen at each state. Hence, no matter how many actions are introduced in a state of the MDP, it just increases the number of options we have to choose a value from for the action variable in the respective state, and the size of the encoding, but not the number of variables. As illustrated in Fig. 9, our approach could find, as witnesses, two new DTMC models with respect to the original Knuth-Yao solution [18] and one of the two even has a state less than the original solution. This shows how our approach can be exploited for the synthesis of probabilistic programs satisfying certain program sketches.

Table 2: Case Studies. **TA:** Timing attack. **PW:** Password leakage. **TS:** Thread scheduling. **PC:** Probabilistic conformance. #op: Formula size (number of operators). #st: Number of states. #tr: Number of transitions.

Case Study		#op	#st	#tr
TA	$m = 2$	14	24	46
	$m = 4$		60	136
	$m = 6$		112	274
	$m = 8$		180	460
PW	$m = 2$	14	24	46
	$m = 4$		70	146
	$m = 6$		140	302
	$m = 8$		234	514
TS	$h = (0, 1)$	28	7	13
	$h = (0, 15)$		35	83
	$h = (4, 8)$		21	48
	$h = (8, 15)$		35	83
	$h = (10, 20)$		45	108
PC	$s = (0)$	44	20	188
	$s = (0..1)$		20	340
	$s = (0..2)$		20	494
	$s = (0..3)$		20	648
	$s = (0..4)$		20	802
	$s = (0..5)$		20	956
	$s = (0..6)$		20	1110

Regarding the running times in Table 3, we note that our previous naive implementation as well as our new optimized implementation (see Section 7.1) are only prototypical and there is still scope of optimizations. Nevertheless, the new implementation already shows strong improvements due to our optimiza-

Table 3: Experimental results and comparison. **TA:** Timing attack. **PW:** Password leakage. **TS:** Thread scheduling. **PC:** Probabilistic conformance. **TO:** Timeout. **N:** Naive. **O:** Optimized. **SE:** SMT encoding. **SS:** SMT solving

Case Study		**Running time(s)**						**#SMT variables**	
		SE		**SS**		**Total**			
		N	**O**	**N**	**O**	**N**	**O**	**N**	**O**
TA	$m = 2$	5	2	< 1	< 1	5	2	8088	2520
	$m = 4$	114	18	20	1	134	19	50460	14940
	$m = 6$	1721	140	865	45	2586	185	175728	51184
	$m = 8$	12585	952	**TO**	426	**TO**	1378	388980	131220
PW	$m = 2$	5	2	< 1	< 1	6	3	8088	2520
	$m = 4$	207	26	40	1	247	27	68670	20230
	$m = 6$	3980	331	1099	41	5079	372	274540	79660
	$m = 8$	26885	2636	**TO**	364	**TO**	3000	657306	221130
TS	$h = (0, 1)$	< 1	< 1	< 1	< 1	1	1	1379	441
	$h = (0, 15)$	60	8	1607	< 1	1667	8	34335	8085
	$h = (4, 8)$	12	3	17	< 1	29	3	12369	3087
	$h = (8, 15)$	60	8	1606	< 1	1666	8	34335	8085
	$h = (10, 20)$	186	19	13707	1	13893	20	52695	13095
PC	s=(0)	277	10	1996	5	2273	15	21220	6780
	s=(0,1)	822	13	5808	5	6630	18	21220	6780
	s=(0..2)	1690	15	**TO**[a]	5	**TO**	20	21220	6780
	s=(0..3)	4631	16	**TO**	7	**TO**	23	21220	6780
	s=(0..4)	7353	22	**TO**	21	**TO**	43	21220	6780
	s=(0..5)	10661	19	**TO**	61	**TO**	80	21220	6780
	s=(0..6)	13320	18	**TO**	41	**TO**	59	21220	6780

[a]Without timeout, this case needed 58095s for solving

tion. The timing data is the average for five runs per experiment, although we noticed no considerable change in the timing among individual runs of experiments. In our naive implementation, due to encoding of all formulas for all composed states, both the encoding as well as SMT solving time were significantly higher. Hence, we opted for a timeout for cases where the timing did not seem practically useful. For **TA**, **PW**, and **PC**, we used a timeout of 10000s for the SMT solving. Just to get an idea of how long it actually might take to solve the encoding for a moderately big model, we let **PC** for s=(0..2) run without a timeout: it took 58095s for solving, running a total of 59785s for both solving and encoding.

8. Related Work

The first attempt on developing a temporal logic for probabilistic hyperproperties was our work in [7], where we proposed HyperPCTL for DTMCs. We later extended HyperPCTL to MDPs in the conference version of this paper [13]. The work in [26] is perhaps the closest to our approach in this paper. The authors propose the temporal logic PHL. Similar to HyperPCTL, PHL also allows quantification over schedulers, but path quantification of the induced DTMC is

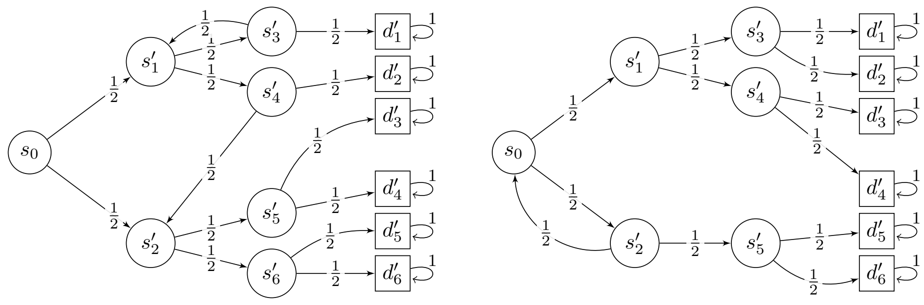


Figure 9: Two alternative DTMC models (w.r.t. Knuth-Yao Algorithm [18]) with seven (on the left) and six (on the right) intermediate states.

achieved by using HyperCTL*. Both papers show that the model checking problem is undecidable for the respective logics. The difference, however, is in our approaches to deal with the undecidability result, which leads two complementary and orthogonal techniques. For both logics, the problem is decidable for non-probabilistic memoryless schedulers. We provide an SMT-based verification procedure for HyperPCTL for this class of schedulers. The work in [26] presents two approximate methods for proving and for refuting only universally quantified formulas in PHL for memoryful schedulers. The two papers offer disjoint case studies for evaluation.

Other efforts in dealing with probabilistic hyperproperties include works on *statistical model checking* (SMC). In [9], the authors propose an SMC algorithm based on sequential probability ration test (SPRT) for an extension of HyperPCTL. This extension allows explicit path quantification over probability operators. SMC has the advantage of being more scalable and providing statistical guarantees of accuracy over bounded temporal operators. SMC in the context of continuous stochastic signals was studied in [5].

In the context of non-probabilistic hyperproperties, there has been a lot of recent progress in automatically verifying [27, 28, 29, 30] and monitoring [31, 32, 33, 6, 34, 35, 36] HyperLTL specifications. HyperLTL is also supported by a growing set of tools, including the model checker MCHyper [27, 30], the satisfiability checkers EAHyper [37] and MGHyper [38], and the runtime monitoring tool RVHyper [34]. The complexity of *model checking* for HyperLTL for tree-shaped, acyclic, and general graphs was rigorously investigated in [21]. The first algorithms for model checking HyperLTL and HyperCTL* using alternating automata were introduced in [27]. A bounded model checking technique for HyperLTL has been introduced in [39]. The *satisfiability* problem for HyperLTL is shown to be undecidable in general but decidable for the $\exists^*\forall^*$ fragment and for any fragment that includes a $\forall\exists$ quantifier alternation [40]. The hierarchy of hyperlogics beyond HyperLTL was studied in [41]. The synthesis problem for HyperLTL has been studied in [42] in the form of *program repair*, in [43] in the form of *controller synthesis*, and in [12] for the general case.

9. Conclusion and Future Work

We investigated the problem of specifying and model checking probabilistic hyperproperties of Markov decision processes (MDPs). Our study is motivated by the fact that many systems have probabilistic nature and are influenced by nondeterministic actions of their environment. We extended the temporal logic HyperPCTL for DTMCs [7] to the context of MDPs by allowing formulas to quantify over schedulers. This additional expressive power leads to undecidability of the HyperPCTL model checking problem on MDPs, but we also showed that the undecidable fragment becomes decidable for non-probabilistic memoryless schedulers. Indeed, all applications discussed in this paper only require this type of schedulers.

Due to the high complexity of the problem, more efficient model checking algorithms are greatly needed. An orthogonal solution is offered by less accurate and/or approximate algorithms such as statistical model checking that scale better and provide certain probabilistic guarantees about the correctness of verification. Another interesting direction is using counterexample-guided techniques to manage the size of the state space.

10. Acknowledgments

We would like to deeply thank Sebastian Junges, Radboud University and Matthias Volk, RWTH Aachen for their help with linking the STORMPY API with our implementation.

This research has been partially supported by the United States NSF SaTC Award 2100989, by the Vienna Science and Technology Fund ProbInG Grant ICT19-018 and by the DFG Research and Training Group UnRAVeL.

References

[1] M. R. Clarkson, F. B. Schneider, Hyperproperties, Journal of Computer Security 18 (6) (2010) 1157–1210.

[2] B. Alpern, F. B. Schneider, Defining liveness, Information Processing Letters 21 (1985) 181–185.

[3] J. A. Goguen, J. Meseguer, Security policies and security models, in: IEEE Symp. on Security and Privacy, 1982, pp. 11–20.

[4] S. Zdancewic, A. C. Myers, Observational determinism for concurrent program security, in: Proc. of CSFW'03, 2003, p. 29.

[5] Y. Wang, M. Zarei, B. Bonakdarpour, M. Pajic, Statistical verification of hyperproperties for cyber-physical systems, ACM Trans. on Embedded Computing systems 18 (5s) (2019) 92:1–92:23.

[6] B. Bonakdarpour, C. Sánchez, G. Schneider, Monitoring hyperproperties by combining static analysis and runtime verification, in: Proc. of ISoLA'18, 2018, pp. 8–27.

[7] E. Ábrahám, B. Bonakdarpour, HyperPCTL: A temporal logic for probabilistic hyperproperties, in: Proc. of QEST'18, 2018, pp. 20–35.

[8] E. Ábrahám, E. Bartocci, B. Bonakdarpour, O. Dobe, Parameter synthesis for probabilistic hyperproperties, in: Proc. of LPAR-23, Vol. 73 of EPiC Series in Computing, EasyChair, 2020, pp. 12–31.

[9] Y. Wang, S. Nalluri, B. Bonakdarpour, M. Pajic, Statistical model checking for hyperproperties, in: Proc. of CSF'21, 2021, to appear.

[10] M. Guarnieri, S. Marinovic, D. Basin, Securing databases from probabilistic inference, in: Proc. of CSF'17, 2017, pp. 343–359.

[11] O. Dobe, E. Ábrahám, E. Bartocci, B. Bonakdarpour, Hyperprob: A model checker for probabilistic hyperproperties, in: Formal Methods - 24th International Symposium, FM 2021, Virtual Event, November 20-26, 2021, Proceedings, 2021, pp. 657–666.

[12] B. Finkbeiner, C. Hahn, P. Lukert, M. Stenger, L. Tentrup, Synthesis from hyperproperties, Acta Informatica 57 (1-2) (2020) 137–163.

[13] E. Ábrahám, E. Bartocci, B. Bonakdarpour, O. Dobe, Probabilistic hyperproperties with nondeterminism, in: Proc. of ATVA'20, Vol. 12302 of LNCS, 2020, pp. 518–534.

[14] C. Baier, J.-P. Katoen, Principles of Model Checking, The MIT Press, 2008.

[15] J. W. G. III, Toward a mathematical foundation for information flow security, Journal of Computer Security 1 (3-4) (1992) 255–294.

[16] C. Dwork, A. Roth, The algorithmic foundations of differential privacy, Foundations and Trends in Theoretical Computer Science 9 (3-4) (2014) 211–407.

[17] T. M. Ngo, M. Stoelinga, M. Huisman, Confidentiality for probabilistic multi-threaded programs and its verification, in: Proc. of ESSoS'13, 2013, pp. 107–122.

[18] D. Knuth, A. Yao, Algorithms and Complexity: New Directions and Recent Results, Academic Press, 1976, Ch. The complexity of nonuniform random number generation.

[19] C. Baier, T. Brázdil, M. Größer, A. Kucera, Stochastic game logic, Acta Informatica 49 (4) (2012) 203–224.

[20] C. Baier, N. Bertrand, M. Größer, On decision problems for probabilistic Büchi automata, in: Proc. of FOSSACS'08, 2008, pp. 287–301.

[21] B. Bonakdarpour, B. Finkbeiner, The complexity of monitoring hyperproperties, in: Proc. of CSF'18, 2018, pp. 162–174.

[22] STORMPY, https://moves-rwth.github.io/stormpy/.

[23] C. Dehnert, S. Junges, J. Katoen, M. Volk, A Storm is coming: A modern probabilistic model checker, in: Proc. of CAV'17, 2017, pp. 592–600.

[24] L. M. de Moura, N. Bjørner, Z3: An efficient SMT solver, in: Proc. of TACAS'08, 2008, pp. 337–340.

[25] The PRISM language, https://www.prismmodelchecker.org/manual/ThePRISMLanguage/Introduction.

[26] R. Dimitrova, B. Finkbeiner, H. Torfah, Probabilistic hyperproperties of Markov decision processes, in: Proc. of ATVA'20, Vol. 12302 of LNCS, Springer, 2020, pp. 484–500.

[27] B. Finkbeiner, M. N. Rabe, C. Sánchez, Algorithms for model checking HyperLTL and HyperCTL*, in: Proc. of CAV'15, 2015, pp. 30–48.

[28] B. Finkbeiner, C. Müller, H. Seidl, E. Zalinescu, Verifying Security Policies in Multi-agent Workflows with Loops, in: Proc. of CCS'17, 2017.

[29] B. Finkbeiner, C. Hahn, H. Torfah, Model checking quantitative hyperproperties, in: Proc. of CAV'18, 2018, pp. 144–163.

[30] N. Coenen, B. Finkbeiner, C. Sánchez, L. Tentrup, Verifying hyperliveness, in: Proc. of CAV'19, 2019, pp. 121–139.

[31] S. Agrawal, B. Bonakdarpour, Runtime verification of k-safety hyperproperties in HyperLTL, in: Proc. of CSF'16, 2016, pp. 239–252.

[32] B. Finkbeiner, C. Hahn, M. Stenger, L. Tentrup, Monitoring hyperproperties, Formal Methods in System Design 54 (3) (2019) 336–363.

[33] N. Brett, U. Siddique, B. Bonakdarpour, Rewriting-based runtime verification for alternation-free HyperLTL, in: Proc. of TACAS'17, 2017, pp. 77–93.

[34] B. Finkbeiner, C. Hahn, M. Stenger, L. Tentrup, RVHyper: A runtime verification tool for temporal hyperproperties, in: Proc. of TACAS'18, 2018, pp. 194–200.

[35] S. Stucki, C. Sánchez, G. Schneider, B. Bonakdarpour, Graybox monitoring of hyperproperties, in: Proc. of FM'19, 2019, pp. 406–424.

[36] C. Hahn, M. Stenger, L. Tentrup, Constraint-based monitoring of hyperproperties, in: Proc. of TACAS'19, 2019, pp. 115–131.

[37] B. Finkbeiner, C. Hahn, M. Stenger, Eahyper: Satisfiability, implication, and equivalence checking of hyperproperties, in: Proc. of CAV'17, 2017, pp. 564–570.

[38] B. Finkbeiner, C. Hahn, T. Hans, MGHyper: Checking satisfiability of HyperLTL formulas beyond the $\exists^*\forall^*$ fragment, in: Proc. of ATVA'18, 2018, pp. 521–527.

[39] T.-H. Hsu, C. Sánchez, B. Bonakdarpour, Bounded model checking for hyperproperties, in: Proc. of TACAS'21, 2021, to appear.

[40] B. Finkbeiner, C. Hahn, Deciding hyperproperties, in: Proc. of CONCUR'16, 2016, pp. 13:1–13:14.

[41] N. Coenen, B. Finkbeiner, C. Hahn, J. Hofmann, The hierarchy of hyperlogics, in: Proc. of LICS'19, 2019, pp. 1–13.

[42] B. Bonakdarpour, B. Finkbeiner, Program repair for hyperproperties, in: Proc. of ATVA'19, 2019, pp. 423–441.

[43] B. Bonakdarpour, B. Finkbeiner, Controller synthesis for hyperproperties, in: Proc. of CSF'20, 2020, pp. 366–379.