# Parameter Synthesis for Probabilistic Hyperproperties*

Erika Ábrahám[1], Ezio Bartocci[2], Borzoo Bonakdarpour[3], and Oyendrila Dobe[4]

[1] RWTH Aachen University, Aachen, Germany
abraham@informatik.rwth-aachen.de
[2] TU Wien, Vienna, Austria
ezio.bartocci@tuwien.ac.at
[3] Michigan State University, East Lansing, MI, USA
borzoo@msu.edu
[4] Iowa State University, Ames, IA, USA
odobe@iastate.edu

**Abstract**

In this paper, we study the *parameter synthesis problem* for *probabilistic hyperproperties*. A probabilistic hyperproperty stipulates quantitative dependencies among a set of executions. In particular, we solve the following problem: given a probabilistic hyperproperty $\psi$ and discrete-time Markov chain $\mathcal{D}$ with parametric transition probabilities, compute regions of parameter configurations that instantiate $\mathcal{D}$ to satisfy $\psi$, and regions that lead to violation. We address this problem for a fragment of the temporal logic HyperPCTL that allows expressing quantitative reachability relation among a set of computation trees. We illustrate the application of our technique in the areas of differential privacy, probabilistic nonintereference, and probabilistic conformance.

## 1 Introduction

We first motivate the problem studied in this paper through a simple example. Consider the concept of *differential privacy* [20,21], that is, a commitment by a data holder to a data subject that he/she will not be affected by allowing his/her data to be used in any study or analysis. More formally, let $\epsilon$ 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 *$\epsilon$-differentially private*, if for all databases $D_1$ and $D_2$ that differ on a single element, and all subsets $S$ of possible outputs of $\mathcal{A}$, we have:

$$Pr[\mathcal{A}(D_1) \in S] \leq e^{\epsilon} \cdot Pr[\mathcal{A}(D_2) \in S].$$

One way to guarantee differential privacy is by introducing *randomized response* to create noise and provide plausible deniability. For example, 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, and (3) if head, then flip the coin again and respond "Yes" if head and "No" if tail. Thus, there are no good or bad responses and an answer cannot be incriminating. The discrete-time Markov chain (DTMC) of this protocol conducted by a fair coin is shown in Fig. 1a, where $t = y$ (respectively, $t = n$) denotes that the truth is 'Yes' (respectively, 'No') and $r = y$ (respectively, $r = n$) denotes the fact that the response is 'Yes' (respectively, 'No'). It is straightforward to show that this protocol is $(\ln 3)$-differentially private.

*This research has been partially supported by the United States NSF SaTC Award 1813388. The order of authors is alphabetical and all authors made equal contribution.


(a) DTMC (b) Parametric DTMC

Figure 1: The randomized response protocol.

Now, let us imagine that we intend to change this protocol in order to make it $(\ln 2)$-differentially private. To this end, one can first transform the DTMC shown in Fig. 1a into a *parametric* DTMC (see Fig. 1b) that allows two different types of coins to be used during the protocol, hence, parameters $p$ and $q$. Then, we solve the *parameter synthesis problem* by finding a value for parameters $p$ and $q$ that result in an $(\ln 2)$-differentially private protocol. Differential privacy is a *probabilistic hyperproperty*, as it prescribes a probability relation between a set of independent executions. Although the parameter synthesis problem has been extensively studied in the context of conventional properties, to our knowledge, it has not yet been solved in the context of hyperproperties.

In this paper, our goal is to solve the parameter synthesis problem for a fragment of the temporal logic HyperPCTL [1]. HyperPCTL lifts the well-known probabilistic temporal logic PCTL by allowing to express stochastic relations between computations starting in different initial states. The fragment studied in this paper, called ReachHyperPCTL, is restricted to non-nested probability operators. For the above randomized response protocol the following ReachHyperPCTL formula states that whenever a computation starts in a state $\sigma$ and another computation in $\sigma'$ with a different truth value, the probabilities to get the same response satisfy the $(\ln 3)$-differential privacy condition:

$$\varphi_{\mathsf{dp}} = \forall\sigma.\forall\sigma'.\left[\Big((t{=}n)_\sigma \wedge (t{=}y)_{\sigma'}\Big) \Rightarrow \Big(\mathbb{P}\Big(\Diamond(r{=}n)_\sigma\Big) \leq e^{\ln 3}\cdot\mathbb{P}\Big(\Diamond(r{=}n)_{\sigma'}\Big)\Big)\right] \wedge \left[\Big((t{=}y)_\sigma \wedge (t{=}n)_{\sigma'}\Big) \Rightarrow \Big(\mathbb{P}\Big(\Diamond(r{=}y)_\sigma\Big) \leq e^{\ln 3}\cdot\mathbb{P}\Big(\Diamond(r{=}y)_{\sigma'}\Big)\Big)\right] \quad (1)$$

Given a parametric DTMC $\mathcal{D}$ and ReachHyperPCTL formula $\psi$, we solve the parameter synthesis problem for ReachHyperPCTL in two steps. In the first step, we compute for each possible instantiation of the quantified (initial) states an arithmetic formula over the model parameters that is true for exactly those parameter configurations that instantiate $\mathcal{D}$ to satisfy $\psi$. In a second step, we use those formulas to compute not only single solutions, but whole regions of satisfying parameter configurations: we decompose the configuration domain and identify smaller regions in which either all or none of the configurations lead to the satisfaction of $\psi$.

We illustrate the application of our technique by using four case studies. Our first example is differential privacy, as described above. The second example is *probabilistic noninterference* [32], which establishes a connection between information theory and information flow by employing probabilities to address covert channels. The third case study, is *probabilistic conformance*, where we want to find the parameter values of two systems (e.g., a model and an implementation) such that they conform with each other with respect to a specification. The last case study is the dining cryptographers problem [12], where we show that the anonymity of the cryptographers is assured also when using a biased coin in the protocol proposed in [12].

**Organization.** The rest of the paper is organized as follows. We recall the preliminary concepts in Section 2 and present our parameter synthesis algorithm in Section 3, followed by illustrative examples and case studies in Section 4. We discuss related work in Section 5 and conclude the paper in Section 6.

# 2 Preliminaries

As usual, $\mathbb{R}$, $\mathbb{Q}$, and $\mathbb{N}$ denote the real, rational, and natural (incl. zero) numbers, respectively. We set $\underline{n} = \{1, \ldots, n\}$ for $n \in \mathbb{N}$.

## 2.1 Parametric Discrete-time Markov Chains

We assume our system to be modeled as a parametric discrete-time Markov chain.

**Definition 1.** *A (labeled)* discrete-time Markov chain (DTMC) *is a tuple* $\mathcal{M} = (S, \mathbf{P}, \mathsf{AP}, L)$, *where (1)* $S$ *is a finite non-empty set of* states, *(2)* $\mathbf{P} : S \times S \to [0, 1]$ *is a* transition probability function *with*

$$\sum_{s' \in S} \mathbf{P}(s, s') = 1$$

*for all states* $s \in S$, *(3)* $\mathsf{AP}$ *is a set of* atomic propositions, *and (4)* $L : S \to 2^{\mathsf{AP}}$ *is a* labeling function. ■

A *path* of a DTMC $\mathcal{M} = (S, \mathbf{P}, \mathsf{AP}, L)$ is an infinite sequence $\pi = s_0 s_1 s_2 \cdots \in S^\omega$ of states with $\mathbf{P}(s_i, s_{i+1}) > 0$ for all $i \geq 0$; we write $\pi[i]$ for $s_i$. Let $Paths^s(\mathcal{M})$ denote the set of all (infinite) paths of $\mathcal{M}$ starting in $s$, and $Paths^s_{fin}(\mathcal{M})$ denote the set of all finite prefixes of paths from $Paths^s(\mathcal{M})$, which we sometimes call *finite paths*.

The *cylinder set* $Cyl(\pi)$ of a finite path $\pi \in Paths^s_{fin}(\mathcal{M})$ is the set of all (infinite) paths of $\mathcal{M}$ with prefix $\pi$. The probability space of $\mathcal{M}$ is defined by the smallest $\sigma$-algebra that contains the cylinder sets of all finite paths of $\mathcal{M}$ with probabilities $Pr(Cyl(s_0 \ldots s_k)) = \Pi_{i \in \underline{k-1}} \mathbf{P}(s_i, s_{i+1})$; for more details see [4].

To turn DTMCs parametric, we consider an ordered set $V$ of real-valued *parameters*, whose *configurations* are functions $u : V \to \mathbb{R}$.

**Definition 2.** *A* rational function $f$ *over a finite set* $V = \{v_1, \ldots, v_r\}$ *of parameters is a function that can be written as* $f = \frac{f_1}{f_2}$ *for some polynomials* $f_1$ *and* $f_2 \not\equiv 0$ *using variables from* $V$. ■

Let $\mathcal{F}_V$ be the field of real-valued rational functions. Given $f \in \mathcal{F}_V$ and a configuration $u : V \to \mathbb{R}$, we denote by $f[V/u]$ the value obtained by substituting in $f$ each occurrence of each $v \in V$ with $u(v)$.

**Definition 3.** *A* parametric DTMC (PDTMC) *is a tuple* $\mathcal{D} = (S, V, \mathbf{P}, \mathsf{AP}, L)$, *where* $S$, $\mathsf{AP}$ *and* $L$ *are as in Def. 1,* $V$ *is a finite set of parameters and* $\mathbf{P} : S \times S \to \mathcal{F}_V$. ■

A configuration $u$ for $V$ is *valid* if $\mathcal{D}_u = (S, \mathbf{P}_u, \mathsf{AP}, L)$ is a DTMC with $\mathbf{P}_u(s, s') = \mathbf{P}(s, s')[V/u]$ and

$$\sum_{s' \in S} \mathbf{P}_u(s, s') = 1$$

for all $s \in S$. We call $\mathcal{D}_u$ the *DTMC induced from* $\mathcal{D}$ *by* $u$.

## 2.2 The Logic ReachHyperPCTL

In a previous work, we have introduced HyperPCTL [1] to characterize quantitative dependencies between different independent executions of a DTMC model, and proposed a procedure for model checking such properties. In this paper, we consider *parametric* DTMCs and the problem to synthesize parameter configurations that satisfy a certain probabilistic hyperproperty. As this problem involves symbolic encodings of reachability probabilities and the truth values of hyperproperties, in order to provide an effective synthesis algorithm, we restrict ourselves to a fragment called ReachHyperPCTL, which excludes nested probability operators.

**Syntax.** ReachHyperPCTL is syntactically defined over a set $\mathsf{AP}$ of atomic propositions by the following abstract grammar:

$$\begin{array}{lcl} \psi & ::= & \forall\sigma.\psi \mid \exists\sigma.\psi \mid a_\sigma \mid \psi \wedge \psi \mid \neg\psi \mid p \sim p \\ p & ::= & \mathbb{P}(\bigcirc \varphi) \mid \mathbb{P}(\varphi \,\mathcal{U}\, \varphi) \mid f(p,\ldots,p) \\ \varphi & ::= & a_\sigma \mid \varphi \wedge \varphi \mid \neg\varphi \end{array}$$

where $a \in \mathsf{AP}$ is an atomic proposition, $\sim \in \{<, \leq, =, \geq, >\}$, $\sigma$ are *state variables* from a countably infinite set $\mathcal{V}$, and $f : [0,1]^k \to \mathbb{R}$ are $k$-ary elementary functions to express arithmetic operations over probabilities, where constants are viewed as 0-ary functions. We call $\varphi$ and $\psi$ *state formulas*, $a_\sigma$ an *indexed atomic proposition* and $p$ a *probability expression*. We denote by $\mathcal{F}$ the set of all ReachHyperPCTL state formulas and probability expressions (over $\mathsf{AP}$). The difference to HyperPCTL is that temporal operators may be applied to Boolean combinations of atomic propositions only (operands $\varphi$ instead of $\psi$). We use standard syntactic sugar $\texttt{false} = a_\sigma \wedge \neg a_\sigma$, $\texttt{true} = \neg\texttt{false}$, $\varphi_1 \vee \varphi_2 = \neg(\neg\varphi_1 \wedge \neg\varphi_2)$, $\Diamond\varphi = \texttt{true}\,\mathcal{U}\,\varphi$, $\Box\varphi = \neg\Diamond\neg\varphi$, etc.

An occurrence of an indexed atomic proposition $a_\sigma$ in a ReachHyperPCTL state formula $\psi$ is *free* if it is not in the scope of a quantifier bounding $\sigma$ and otherwise *bound*. ReachHyperPCTL *sentences* are ReachHyperPCTL state formulas in which all occurrences of all indexed atomic propositions are bound. ReachHyperPCTL (*quantified*) *formulas* are ReachHyperPCTL sentences.

Each ReachHyperPCTL quantified formula can be transformed into an equivalent formula in prenex normal form $\mathbb{Q}_1\sigma_1.\ldots\mathbb{Q}_n\sigma_n.\psi$, where each $\mathbb{Q}_i \in \{\forall, \exists\}$ is a quantifier, $\sigma_i$ is a state variable, and $\psi$ is a quantifier-free ReachHyperPCTL formula. In the following we assume all ReachHyperPCTL quantified formulas to be in prenex normal form.

**Example.** The formula

$$\forall\sigma_1.\exists\sigma_2.\Big(\mathbb{P}(\Diamond a_{\sigma_1}) = \mathbb{P}(\Diamond b_{\sigma_2})\Big)$$

holds if for each state $s_1$, there exists another state $s_2$, such that the probability to finally reach a state labeled with $a$ from $s_1$ equals the probability of reaching $b$ from $s_2$.

**Semantics.** We present the semantics of ReachHyperPCTL based on the $n$-ary *self-composition* of a DTMC. We emphasize that it is possible to define the semantics in terms of the non-self-composed DTMC, but it will essentially result in a very similar setting, but more difficult to understand.

**Definition 4.** *The* n-ary self-composition *of a PDTMC* $\mathcal{M} = (S, V, \mathbf{P}, \mathsf{AP}, L)$ *is the PDTMC* $\mathcal{M}^n = (S^n, V, \mathbf{P}^n, \mathsf{AP}^n, L^n)$ *with*

- $S^n = S \times \ldots \times S$ *is the n-ary Cartesian product of* $S$,

- $\mathbf{P}^n\big(s,s'\big) = \Pi_{i\in\underline{n}}\mathbf{P}(s_i,s'_i)$ *for all* $s=(s_1,\ldots,s_n)\in S^n$ *and* $s'=(s'_1,\ldots,s'_n)\in S^n$,
- $\mathsf{AP}^n = \cup_{i\in\underline{n}}\mathsf{AP}_i$, *where* $\mathsf{AP}_i=\{a_i \mid a\in\mathsf{AP}\}$ *for* $i\in\underline{n}$, *and*
- $L^n(s)=\cup_{i\in\underline{n}} L_i(s_i)$ *for all* $s=(s_1,\ldots,s_n)\in S^n$ *with* $L_i(s_i)=\{a_i \mid a\in L(s_i)\}$ *for* $i\in\underline{n}$. ■

The satisfaction relation for ReachHyperPCTL sentences by a DTMC $\mathcal{M}=(S,\mathbf{P},\mathsf{AP},L)$ is defined by:

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

where () is the empty sequence of states. Thus, the satisfaction relation $\models$ defines the values of ReachHyperPCTL quantified, state, and path formulas in the context of a DTMC $\mathcal{M}=(S,\mathbf{P},\mathsf{AP},L)$ and an $n$-tuple $s=(s_1,\ldots,s_n)\in S^n$ of states (which is () for $n=0$). Intuitively, the state sequence $s$ stores instantiations for quantified state variables. The semantics evaluates ReachHyperPCTL formulas by structural recursion. Quantifiers are instantiated and the instantiated values for state variables are stored in the state sequence $s$. To maintain the connection between a state in this sequence and the state variable which it instantiates, we introduce the auxiliary syntax $a_i$ with $a\in\mathsf{AP}$ and $i\in\mathbb{N}_{>0}$, and if we instantiate $\sigma$ in $\exists\sigma.\psi$ or $\forall\sigma.\psi$ by state $s$, then we append $s$ at the end of the state sequence and replace all $a_\sigma$ that is bound by the given quantifier by $a_i$ with $i$ being the index of $s$ in the state sequence. We will express the meaning of path formulas based on the $n$-ary self-composition of $\mathcal{M}$; the index $i$ for the instantiation of $\sigma$ also fixes the component index in which we keep track of the paths starting in $\sigma$. The semantics judgment rules to evaluate formulas in the context of a DTMC $\mathcal{M}=(S,\mathbf{P},\mathsf{AP},L)$ and an $n$-tuple $s=(s_1,\ldots,s_n)\in S^n$ of states are the following:

$$\begin{array}{lll}
\mathcal{M},s\models\forall\sigma.\psi & \textit{iff} & \mathcal{M},(s_1,\ldots,s_n,s_{n+1})\models\psi[\mathsf{AP}_{n+1}/\mathsf{AP}_\sigma] \textit{ for all } s_{n+1}\in S\\
\mathcal{M},s\models\exists\sigma.\psi & \textit{iff} & \mathcal{M},(s_1,\ldots,s_n,s_{n+1})\models\psi[\mathsf{AP}_{n+1}/\mathsf{AP}_\sigma] \textit{ for some } s_{n+1}\in S\\
\mathcal{M},s\models a_i & \textit{iff} & a\in L(s_i)\\
\mathcal{M},s\models\psi_1\wedge\psi_2 & \textit{iff} & \mathcal{M},s\models\psi_1 \textit{ and } \mathcal{M},s\models\psi_2\\
\mathcal{M},s\models\neg\psi & \textit{iff} & \mathcal{M},s\not\models\psi\\
\mathcal{M},s\models p_1\sim p_2 & \textit{iff} & [\![p_1]\!]_{\mathcal{M},s}\sim[\![p_2]\!]_{\mathcal{M},s}\\
[\![\mathbb{P}(\bigcirc\varphi)]\!]_{\mathcal{M},s} & = & Pr\Big(\{\pi\in Paths^s(\mathcal{M}^n)\mid\mathcal{M},\pi[1]\models\varphi\}\Big)\\
[\![\mathbb{P}(\varphi_1\,\mathcal{U}\,\varphi_2)]\!]_{\mathcal{M},s} & = & Pr\Big(\{\pi\in Paths^s(\mathcal{M}^n)\mid \textit{exists } j\geq 0 \textit{ such that } \mathcal{M},\pi[j]\models\varphi_2\\
 & & \quad \textit{and } \pi[i]\models\varphi_1 \textit{ for all } 0\leq i<j\}\Big)\\
[\![f(p_1,\ldots,p_k)]\!]_{\mathcal{M},s} & = & f([\![p_1]\!]_{\mathcal{M},s},\ldots,[\![p_k]\!]_{\mathcal{M},s})\\
\mathcal{M},s\models\varphi_1\wedge\varphi_2 & \textit{iff} & \mathcal{M},s\models\varphi_1 \textit{ and } \mathcal{M},s\models\varphi_2\\
\mathcal{M},s\models\neg\varphi & \textit{iff} & \mathcal{M},s\not\models\varphi
\end{array}$$

where $\psi$, $\psi_1$, and $\psi_2$ are ReachHyperPCTL state formulas; the substitution $\psi[\mathsf{AP}_{n+1}/\mathsf{AP}_\sigma]$ replaces for each atomic proposition $a\in\mathsf{AP}$ each free occurrence of $a_\sigma$ in $\psi$ by $a_{n+1}$; $a\in\mathsf{AP}$ is an atomic proposition and $1\leq i\leq n$; $p_1$ and $p_2$ are probability expressions and $\sim\in\{<,\leq,=,\geq,>\}$; $\varphi$ is a ReachHyperPCTL path formula.

**Example.** The ReachHyperPCTL formula

$$\psi=\forall\sigma.\forall\sigma'.(init_\sigma\wedge init_{\sigma'})\;\Rightarrow\;\Big(\mathbb{P}(\Diamond a_\sigma)=\mathbb{P}(\Diamond a_{\sigma'})\Big)$$

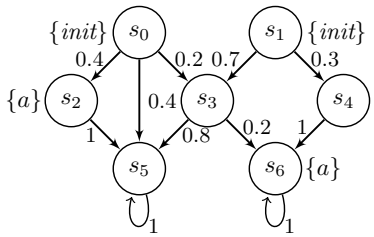



Figure 2: Semantics example.

is satisfied by the DTMC $\mathcal{M}$ in Figure 2 if for all pairs of *init*-labelled states, the probability to reach $a$ is the same, i.e., for each $(s_i, s_j) \in S^2$ with $init \in L(s_i)$ and $init \in L(s_j)$, it holds that $\mathcal{M}, (s_i, s_j) \models \mathbb{P}(\Diamond a_1) = \mathbb{P}(\Diamond a_2)$. The probability of reaching $a$ from $s_0$ is $0.4 + (0.2 \times 0.2) = 0.44$. Moreover, the probability of reaching $a$ from $s_1$ is $(0.7 \times 0.2) + (0.3 \times 1) = 0.44$. Hence, $\mathcal{M} \models \psi$.

# 3 Parameter Synthesis Algorithm for ReachHyperPCTL

**Algorithm 1:** Main parameter synthesis algorithm

**Input :** $\mathcal{D}$: PDTMC; $\psi$: ReachHyperPCTL formula;
$I$: a box of valid parameter configurations;
$max_{it}$: iteration limit.

**Output:** $(\mathcal{R}_{green}, \mathcal{R}_{white}, \mathcal{R}_{red})$: a decomposition of $I$ into boxes from which all $(\mathcal{R}_{green})$, none $(\mathcal{R}_{red})$ resp. some $(\mathcal{R}_{white})$ configurations make $\mathcal{D}$ satisfy $\psi$.

1 **Function** *Main*($\mathcal{D}$, $\psi$, $I$, $max_{it}$)
2 SymbolicEncoding($\mathcal{D}$, $\psi$, 0);
3 **return** checkParameterSpace($\mathcal{D}$, $\psi$, $I$, $max_{it}$);

Assume in the following a ReachHyperPCTL quantified formula (i.e. sentence) $\psi$ in prenex normal form $\psi = \mathbb{Q}_1\sigma_1 \ldots \mathbb{Q}_n\sigma_n.\psi'$ with quantifiers $\mathbb{Q}_i \in \{\forall, \exists\}$ for $i = 1, \ldots, n$. Assume furthermore, a parametric DTMC $\mathcal{D} = (S, V, \mathbf{P}, \mathsf{AP}, L)$ with valid parameter configuration domain $I$ and let $\mathcal{D}^n = (S^n, V, \mathbf{P}^n, \mathsf{AP}^n, L^n)$ be the $n$-ary self-composition of $\mathcal{D}$, defined over the same set of atomic propositions as $\psi$, where $n$ is the number of quantifiers in $\psi$. Our aim in this section is to provide an algorithm for the synthesis of valid parameter configurations for $\mathcal{D}$ such that $\psi$ is satisfied.

The problem is to decide whether a given fixed valid parameter configuration leads to the satisfaction of $\psi$ is decidable. Moreover, for a box $R$ of valid parameter configurations of $\mathcal{D}$, also the problem to decide whether all, none or some of the parameter configurations in $R$ lead to the satisfaction of $\psi$ is solvable. In the following, we propose a parameter synthesis algorithm that will use these computations to decompose a set of valid parameter configurations into a finite set of subsets, and for each of those subsets provide information whether all, none or some configuration in it leads to the satisfaction of the formula. This way, we provide not only single configurations that satisfy the requirements but even sets of them, and point also to regions that contain no satisfying configurations.

The main method is shown in Algorithm 1. In line 2, we first compute for each state $s = (s_1, \ldots, s_n) \in S^n$ of the $n$-ary self-composition $\mathcal{D}^n$ a real-arithmetic formula $Symb(\psi', s)$ over the model parameters that is true for exactly those parameter configurations under which

$\psi'$ holds in state $s$ of $\mathcal{D}^n$. Given a set description $R$ of parameter configurations, unsatisfiability of the formula $R \wedge \mathit{Symb}(\psi', s)$ will thus mean that there is no satisfying configuration in $R$, whereas unsatisfiability of $R \wedge \neg \mathit{Symb}(\psi', s)$ means that all configurations in $R$ are satisfying for $\psi'$. If both are satisfiable than some configurations in $R$ satisfy $\psi'$ and some violate it.

Once the symbolic truth values of the input formula are constructed, in line 3 of Algorithm 1 we try to determine regions in the parameter space such that the input formula either evaluates to true under all parameter values in the region or it evaluates to false for all of them. We will use the constants GREEN $= 1$ respectively RED $= -1$ to encode these properties, and we will use WHITE $= 0$ to express that none of these properties hold.

Next we explain the two above-mentioned computations. The symbolic values for $\psi'$ and all of its subformulas are computed by Algorithm 2. Besides the PDTMC model and a probability expression of a ReachHyperPCTL formula $F$ whose value needs to be computed, the algorithm receives as a third input how many quantifiers we have already processed; this is needed to be able to determine the position of quantifiers during recursive calls on sub-formulas. If $F$ is atomic then we can compute its value in a given state by looking at the state labelling. If $F$ is non-atomic then it is the application of an operator to some operands; the interesting case is when $F$ is the application of a probability operator, in all other cases we call the same method recursively to compute symbolic values for the operands and subsequently syntactically connect those by the respective operator.

There are two cases for the probability operator, one for the probability of a next-expression and one for the until. The symbolic encodings for them are computed by the Algorithms 3 and

**Algorithm 2:** Symbolic value encoding: Main algorithm

**Input :** $\mathcal{D}$=$(S, V, \mathbf{P}, \mathsf{AP}, L)$: PDTMC; $F$: ReachHyperPCTL formula or expression; $i$: number of already processed quantifiers.

1 **Function** *SymbolicEncoding*($\mathcal{D}$, $F$, $i$)
2 **if** $F$ *is* $(\forall\sigma.\ \psi)$ *or* $(\exists\sigma.\ \psi)$ **then**
3 $i := i + 1$;
4 SymbolicEncoding($\mathcal{D}$, $\psi[\mathsf{AP}_i/\mathsf{AP}_\sigma]$, $i$);
5 **else if** $F$ *is* $a_j$ **then**
6 **foreach** $s = (s_1, \ldots, s_n) \in S^n$ **do**
7 **if** $a \in L(s_j)$ **then** $\mathit{Symb}(F, s) :=$ true **else** $\mathit{Symb}(F, s) :=$ false;
8 **else if** $F$ *is* $\psi_1 \wedge \psi_2$ **then**
9 SymbolicEncoding($\mathcal{D}$, $\psi_1$, $i$); SymbolicEncoding($\mathcal{D}$, $\psi_2$, $i$);
10 **foreach** $s = (s_1, \ldots, s_n) \in S^n$ **do** $\mathit{Symb}(F, s) := \mathit{Symb}(\psi_1, s) \wedge \mathit{Symb}(\psi_2, s)$;
11 **else if** $F$ *is* $\neg\psi$ **then**
12 SymbolicEncoding($\mathcal{D}$, $\psi$, $i$);
13 **foreach** $s = (s_1, \ldots, s_n) \in S^n$ **do** $\mathit{Symb}(F, s) := \neg\mathit{Symb}(\psi, s)$;
14 **else if** $F$ *is* $\mathbb{P}(\bigcirc\varphi)$ **then** SymbolicEncodingNext($\mathcal{D}$, $F$);
15 **else if** $F$ *is* $\mathbb{P}(\varphi_1 \,\mathcal{U}\, \varphi_2)$ **then** SymbolicEncodingUntil($\mathcal{D}$, $F$);
16 **else if** $F$ *is* $p_1 \sim p_2$ **then**
17 SymbolicEncoding($\mathcal{D}$, $p_1$, $i$); SymbolicEncoding($\mathcal{D}$, $p_2$, $i$);
18 **foreach** $s = (s_1, \ldots, s_n) \in S^n$ **do** $\mathit{Symb}(F, s) := \mathit{Symb}(p_1, s) \sim \mathit{Symb}(p_2, s)$;
19 **else if** $F$ *is* $c$ **then foreach** $s = (s_1, \ldots, s_n) \in S^n$ **do** $\mathit{Symb}(F, s) := c$;
20 **else if** $F$ *is* $p_1$ *op* $p_2$ *with op* $\in \{+, -, *\}$ **then**
21 SymbolicEncoding($\mathcal{D}$, $p_1$, $i$); SymbolicEncoding($\mathcal{D}$, $p_2$, $i$);
22 **foreach** $s = (s_1, \ldots, s_n) \in S^n$ **do** $\mathit{Symb}(F, s) := \mathit{Symb}(p_1, s)$ *op* $\mathit{Symb}(p_2, s)$;

**Algorithm 3:** Symbolic value encoding: Computation for next

**Input** : $\mathcal{D} = (S, V, \mathbf{P}, \mathsf{AP}, L)$: PDTMC; ReachHyperPCTL expression $\mathbb{P}(\bigcirc \varphi)$

**Function** *SymbolicEncodingNext*($\mathcal{D}$, $\mathbb{P}(\bigcirc \varphi)$)
  $K = \{\ s \in S^n \ \mid\ \mathcal{D}^n, s \models \varphi\ \}$;
  **foreach** $s \in S^n$ **do** $\mathit{Symb}(\mathbb{P}(\bigcirc \varphi), s) := \sum_{s' \in K} \mathbf{P}(s, s')$;

**Algorithm 4:** Symbolic value encoding: Computation for until

**Input** : $\mathcal{D} = (S, V, \mathbf{P}, \mathsf{AP}, L)$: PDTMC; ReachHyperPCTL expression $\mathbb{P}(\varphi_1 \,\mathcal{U}\, \varphi_2)$

**Function** *SymbolicEncodingUntil*($\mathcal{D}$, $\mathbb{P}(\varphi_1 \,\mathcal{U}\, \varphi_2)$)
  $S_1 := \{s \in S^n \ \mid\ \mathcal{D}^n, s \models \varphi_2\}$;
  $S_0 := \{s \in S^n \ \mid\ \mathcal{D}^n, s \models \neg\varphi_1 \wedge \neg\varphi_2\}$;
  $S_? := S^n \setminus (S_1 \cup S_0)$;
  **foreach** $s \in S_1$ **do**
    $\mathit{Symb}(\mathbb{P}(\varphi_1 \,\mathcal{U}\, \varphi_2), s) := 1$; $\mathbf{P}^n(s, s) := 1$;
    **foreach** *successor* $s_2 \in S^n \setminus \{s\}$ *of* $s$ **do** $\mathbf{P}^n(s, s_2) := 0$;
  **foreach** $s \in S_0$ **do**
    $\mathit{Symb}(\mathbb{P}(\varphi_1 \,\mathcal{U}\, \varphi_2), s) := 0$; $\mathbf{P}^n(s, s) := 1$;
    **foreach** *successor* $s_2 \in S^n \setminus \{s\}$ *of* $s$ **do** $\mathbf{P}^n(s, s_2) := 0$;
  **foreach** $s \in S_?$ **do**
    **if** $\mathbf{P}^n(s, s) \notin \{0, 1\}$ **then**
      **foreach** *successor* $s_2 \in S^n \setminus \{s\}$ *of* $s$ **do** $\mathbf{P}^n(s, s_2)$ *$= \frac{1}{1-\mathbf{P}^n(s,s)}$;
      $\mathbf{P}^n(s, s) := 0$;
    **foreach** *predecessor* $s_1 \in S^n \setminus \{s\}$ *of* $s$ **do**
      **foreach** *successor* $s_2 \in S^n \setminus \{s\}$ *of* $s$ **do**
        $\mathbf{P}^n(s_1, s_2)$ += $\mathbf{P}^n(s_1, s) \cdot \mathbf{P}^n(s, s_2)$;
      $\mathbf{P}^n(s_1, s) := 0$;
  **foreach** $s \in S_?$ **do** $\mathit{Symb}(\mathbb{P}(\varphi_1 \,\mathcal{U}\, \varphi_2), s) := \sum_{s_2 \in S_1} \mathbf{P}(s, s_2)$;

4, respectively. The former is quite straightforward: to encode the value of $F = \mathbb{P}(\bigcirc \varphi)$ we first determine the set $K$ of those states of $\mathcal{D}^n$ that satisfy $\varphi$ and then for each $s \in S^n$ the value of $F$ can be encoded by summing up for each direct successor of $s$ that is included in $K$ the probability to move there (in one step). Note that $\varphi$ is a Boolean combination of atomic propositions, therefore its truth can be easily determined for each state.

The case for $F$ being a probability expression $\mathbb{P}(\varphi_1 \,\mathcal{U}\, \varphi_2)$ is a bit more involved. We use state elimination, similarly to the method used in [18] to symbolically express reachability probabilities by arithmetic expressions (rational functions). However, whereas in [18] such a reachability probability needs to be computed for a single initial state, for ReachHyperPCTL properties we need it for *all* states of a parametric DTMC. An algorithm that computes these probability expressions independently, repeatedly applying the method from [18] to each state, would work but it would re-do a lot of computations.

Instead, we apply a slight modification to the standard state elimination approach to make some additional bookkeeping. We first identify states for which the probability is known to be one (state set $S_1$) resp. zero ($S_0$), and make them absorbing (lines 2–10). Then for each remaining state $s$, we remove self-loops, connect pairs of predecessors and successors by direct transitions without visiting $s$ inbetween, and then remove the *incoming* edges of $s$. As illustrated in Fig. 3, the difference to the approach in [18] is that we do not remove the *outgoing* edges of

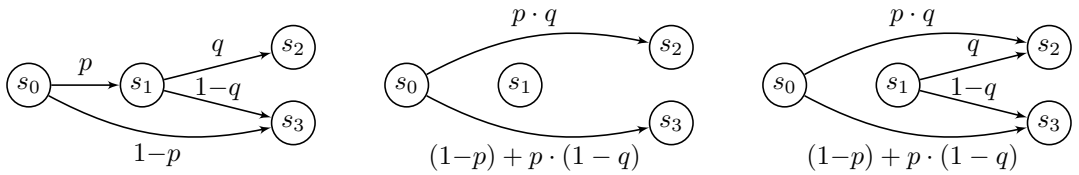



Figure 3: A PDTMC (left) and the result of eliminating $s_1$ in [18] (mid) and in Alg. 4 (right).

$s$, such that after having iterated over all states (lines 11–18), direct transitions from all states to the absorbing ones will remain that allow to express the reachability properties for all states similarly as it was done for the next operator (lines 19–19).

Once we have for all states $s \in S^n$ a symbolic description of the satisfaction of $\psi'$ in $s$, we can start to search for satisfying and violating parameter configurations using Algorithm 5. It maintains three sets, each of which contains zero or more boxes. Boxes from $\mathcal{R}_{green}$ contain only satisfying parameter configurations, boxes from $\mathcal{R}_{red}$ only unsatisfying ones, whereas boxes from $\mathcal{R}_{white}$ are mixed and contain both configuration types. We call the boxes from the respective sets accordingly green, red or white.

**Algorithm 5:** checkParameterSpace

**Input :** $\mathcal{D}$: PDTMC; $\psi$: ReachHyperPCTL formula;
$I$: a box of valid parameter configurations;
$max_{it}$: upper iteration limit.

**Output:** $(\mathcal{R}_{green}, \mathcal{R}_{white}, \mathcal{R}_{red})$: three sets of boxes decomposing $I$, such that each box from $\mathcal{R}_{green}$, $\mathcal{R}_{red}$ resp. $\mathcal{R}_{white}$ contains configurations from which all, none resp. some make $\mathcal{D}$ satisfy $\psi$.

```
1  Function checkParameterSpace(D, ψ, I, max_it)
2      R_green := ∅; R_red := ∅; R_white := ∅; R := {I}; l := 1;
3      while R ≠ ∅ do
4          let R ∈ R;  R := R\{R};
5          color:= checkRegion(D, ψ, R, ());
6          if color= GREEN then R_green := R_green ∪ {R}
7          else if color= RED then R_red := R_red ∪ {R}
8          else
9              if l < max_it then
10                 S := split(R);  l += |S|;  R := R ∪ S;
11             else R_white := R_white ∪ {R};
12     return (R_green, R_white, R_red);
```

A queue $\mathcal{R}$ contains at start the initial box. Iteratively, we take a box $R$ from $\mathcal{R}$ and determine with Algorithm 6 its color. If the color is green or red then we put the box into the corresponding set $\mathcal{R}_{green}$ resp. $\mathcal{R}_{red}$. Otherwise, if the color is white then we *split* $R$ into smaller boxes which are then added to $\mathcal{R}$ for further processing. For the split any heuristics can be used, in the hope that the smaller boxes will become conclusive in their color. To assure termination, after an upper limit of $max_{it}$ boxes have been scheduled for processing in $\mathcal{R}$, we finish by checking the remaining boxes in the queue without splitting and collect the inconclusive ones in $\mathcal{R}_{white}$.

Finally, the last module to discuss is Algorithm 6 which determines the color of a box $R$, i.e.

**Algorithm 6:** checkRegion

**Input :** $\mathcal{D} = (S, V, \mathbf{P}, \mathsf{AP}, L)$: PDTMC; $\psi$: ReachHyperPCTL formula; $R$: a box of valid parameter configurations; $(s_1, \ldots, s_{i-1}) \in S^{i-1}$.

**Output:** *color*: one of the colors GREEN=1, WHITE=0 or RED=-1 encoding whether $\psi$ is satisfied by $\mathcal{D}$ under all, some respectively none of the configurations in $R$.

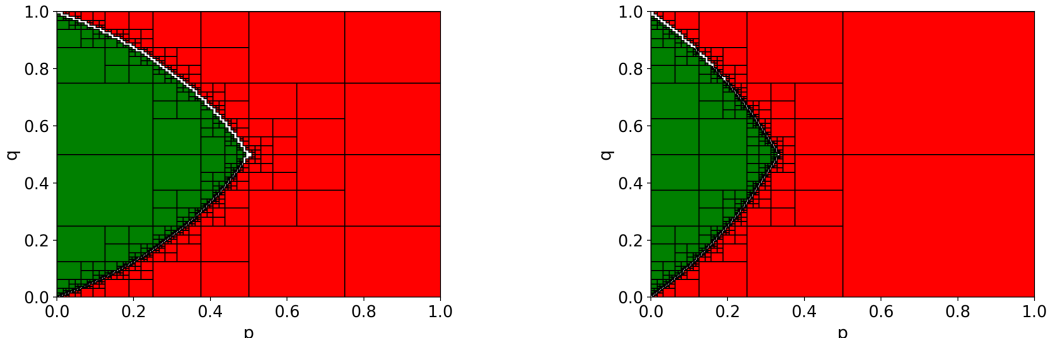
```
1  Function checkRegion(D, ψ, R, (s_1, ..., s_{i-1}))
2      if ψ is Q_i x_i. ... Q_n x_n.ψ' then
3          if Q_i = ∃ then color:= RED else color:= GREEN;
4          foreach s_i ∈ S do
5              color' := checkRegion(D, Q_{i+1} x_{i+1} ... Q_n x_n.ψ', R, (s_1, ..., s_{i-1}, s_i));
6              if Q_i = ∃ then color:= max{color, color'};
7              else if Q_i = ∀ then color:= min{color, color'};
8              if (Q_i = ∃ ∧ color = GREEN) or (Q_i = ∀ and color = RED) then break
9          return color
10     else
11         if R ∧ Symb(ψ, (s_1, ..., s_n)) is unsatisfiable then return RED
12         else if R ∧ ¬Symb(ψ, (s_1, ..., s_n)) is unsatisfiable then return GREEN
13         else return WHITE
```

the truth value of $\psi = Q_1 x_1 \ldots Q_n x_n.\psi'$ under configurations from $R$. Let us first have a look at the lines 11-13, where the truth value of $\psi'$ is checked for a fixed state $(s_1, \ldots, s_n)$ of the $n$-ary self-composition. Here, for a box $R = [l_1, u_1] \times \ldots \times [l_n, u_n]$ we overload notation and use $R$ also to denote its logical description $\bigwedge_{i=1}^{n} l_i \leq x_i \wedge x_i \leq u_i$. Since $Symb(\psi, (s_1, \ldots, s_n))$ encodes the value of $\psi$ in $(s_1, \ldots, s_n)$, the formula $R \wedge Symb(\psi, (s_1, \ldots, s_n))$ is true for all configurations in $R$ that satisfy $\psi$. If this formula is unsatisfiable then we know that no configuration in $R$ satisfies $\psi$ and return the color RED. In contrary, if $R \wedge \neg Symb(\psi, (s_1, \ldots, s_n))$ is unsatisfiable then we know that none of the configurations in $R$ violate $\psi$ and the color of the box is GREEN. Otherwise, if both formulas are satisfiable then some configurations in $R$ satisfy $\psi$ and some others not, therefore the color of the box is WHITE.

It depends on the quantifiers for which states we need to execute this check, as implemented in lines 2-9. For each existential quantifier $Q_i = \exists$ we need to find just a single state that makes the box GREEN in order to make the formula true, whereas for universal quantifiers $Q_i = \forall$ it needs to hold for each state. For the latter case it means also that if the box is red for one state than we know that According to this, quantifiers are instantiated from left to right and the previously described code in lines 11-13 is applied to check the color of the box for the chosen $n$-ary state vector.

As a result of the satisfiability checks in line 11 of Algorithm 6, for purely existentially quantified formulas we can also provide a satisfying configuration for each white box.

Given the soundness of [18], which we use basically unchanged (with the additional book-keeping shown in Fig. 3) to compute symbolic probabilities for the states, our computations in lines 11–13 are sound for each state. Furthermore, for universal state quantifiers we take the weakest satisfaction result under all states, and for existential state quantifiers the strongest one, such that soundness of our algorithm is easy to see assuming soundness of [18]).




(a) $(\ln 3)$-differential privacy. (b) $(\ln 2)$-differential privacy.

Figure 4: Synthesized probability distribution for randomized response.

# 4 Case Studies and Evaluation

We developed a prototypical implementation of our algorithm in python, with the help of several libraries that facilitate the handling of complex mathematical equations involved. There is an extensive use of STORMPY [42], which is a set of python bindings for the probabilistic model checker STORM [19]. It has provided efficient solution to parsing, building, and storage of parametric DTMC models. Internally, STORMPY uses pycarl [37], the python binding of CARL, an Open Source C++ Library for Computer Arithmetic and Logic. Several data structures and datatypes have been used from pycarl and STORMPY to handle complex polynomials, equations, and rational numbers. Finally, we have used the SMT solver Z3 [17] to implement lines 11 and 12 of Algorithm 6. All of our experiments are run on a MacBook Pro laptop with a 2.7Ghz i7 processor with 8GB of RAM. We set $max_{it} = 1500$ in Algorithm 5, process always the oldest inconclusive box in $\mathcal{R}$ (FIFO) and split inconclusive $d$-dimensional white boxes into $2^d$ new boxes by splitting in the middle in each dimension. We start with two smaller examples (randomized response and probabilistic conformance) and then switch to larger case studies (probabilistic noninterference and information leakage).

## 4.1 Randomized Response

We first synthesize configurations for the randomized response protocol described in Section 1. We experimented with the following scenarios. In the first scenario, we synthesized parameters to achieve $\ln 3$-differential privacy with parameters $p$ and $q$ as shown in Fig 1b. The green area in Fig. 4a includes the valid values ($p = 0.75$ and $q = 0.25$, or, $p = q = 0.5$). The white area consists of the values that remain unknown due to termination of the algorithm after 1500 rounds. In the second scenario, our goal is to achieve $\ln 2$-differential privacy. Again, the green area in Fig. 4b includes the valid values (e.g., $p = \frac{2}{3}$ and $q = 0.25$). The time spent to synthesize parameters in all the above scenarios was $0.1s$.

## 4.2 Probabilistic Conformance

The notion of *conformance* describes how well a system implements correctly a given specification in terms of observable behaviors, or, whether two systems (e.g., a model and an imple-

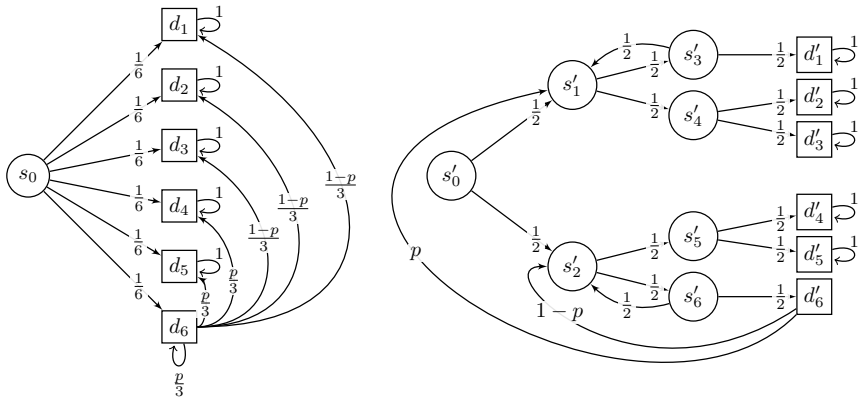



Figure 5: **Left:** a 6-sided die rolled as long as 6 is obtained (i.e., until 1–5). **Right:** a parametric adaptation of the Knuth-Yao algorithm [34] for obtaining the behavior of the 6-sided die protocol from fair coin tosses.

mentation) conform with each other with respect to a specification. In our setting, we model both specification and implementation as PDTMCs.

As an example, let us consider the specification of a protocol, where a 6-sided die is rolled as long as we get the number six (state $d_6$). Figure 5 (left) illustrates graphically our example. Our specification states also that the die, after behaving fairly the first time, can be bias only once according to a parameter $p$. Now, our goal is to implement this protocol using an adaptation of the Knuth-Yao algorithm [34] that was designed originally for simulating a 6-sided die by repeatedly tossing a fair coin, illustrated in Figure 5 (right). In the considered adaptation, we allow to bias the tossing of the coin according to the same parameter $p$, only when the state $d'_6$ (this state represents the number six in the simulated die) is encountered.

In this experiment, we are interested to find the value of $p$, such that the implementation of the protocol conforms with its specification according to the probability of terminating in each one of the five possible states that represent the numbers of the die from one to five. This property can be formally expressed using the following ReachHyperPCTL formula:

$$\varphi_{\mathsf{pc}} = \forall\sigma.\exists\sigma'.\Big(s_{0_\sigma} \wedge s'_{0_{\sigma'}}\Big) \;\Rightarrow\; \bigwedge_{i=1}^{5}\Big(\mathbb{P}(\Diamond\, d_{i_\sigma}) = \mathbb{P}(\Diamond\, d'_{i_{\sigma'}})\Big)$$

We synthesized parameter value $p = 0.5$, meaning that applying a fair coin ensures conformance of the implementation (the right model in Figure 5) with the specification (the left PDTMC in Figure 5). The synthesis for this experiment was $28s$, where $27s$ was spent in Algorithms 2-4 and $1s$ in Algorithms 5-6. The imbalance is mainly due to the fact that the PDTMCs have multiple nested cycles. We also note that $p = 0.5$ is the only valid solution.

## 4.3 Probabilistic Noninterference in Randomized Schedulers

*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).

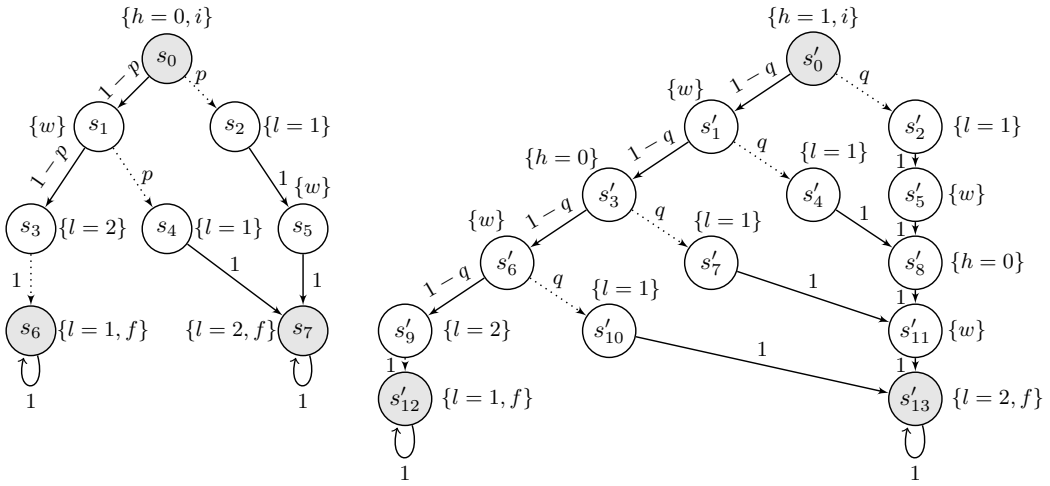



Figure 6: Parametric DTMC for the probabilistic noninterference example program with two threads $th_1$ and $th_2$, and secret inputs $h = 0$ (left) and $h = 1$ (right); solid (resp., dotted) transitions correspond to thread $th_1$ (resp., $th_2$), $i$ labels initial states, proposition, $w$ denotes execution of the while-loop condition checking, and $f$ denotes the terminating state.

*Probabilistic noninterference* [32] establishes connection between information theory and information flow by employing probabilities to address covert channels. Intuitively, it requires that the probability of every low-observable trace pattern is the same for every low-equivalent initial state. Consider the following example [40] of a program with two threads $th_1$ and $th_2$:

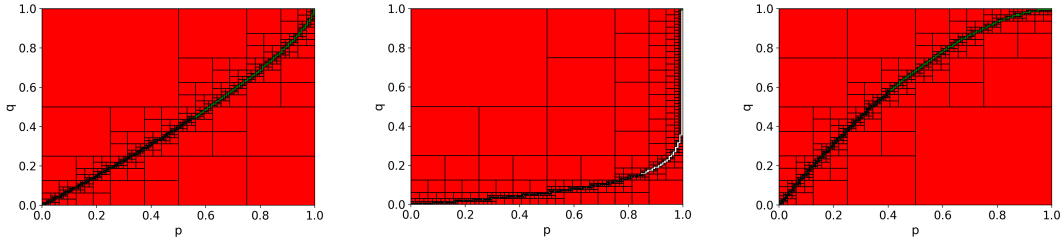
$$th_1 : \ \textbf{while}\ h > 0\ \textbf{do}\ \{h := h - 1\};\ l := 2 \qquad || \qquad th_2 : \ l := 1$$

where $h$ is a *secret* input by a high-privileged user and $l$ is a *public* output observable by low-privileged users. Figure 6 depicts PDTMC models of this program for secret input $h = 0$ (left) and $h = 1$ (right). The parameter configuration $p = 0.5$ resp. $q = 0.5$ models a fair scheduler that chooses each of the threads with equal probability for the execution of the next atomic statement. Probabilistic noninterference requires that $l$ obtains value of 1 (and 2) with equal probability, regardless of the initial value of $h$:

$$\varphi_{\mathsf{pni}} = \forall\sigma.\forall\sigma'.\Big(i_\sigma \wedge (h{=}0)_\sigma \wedge i_{\sigma'} \wedge (h{=}1)_{\sigma'}\Big) \Rightarrow \Big(\Big(\mathbb{P}\big(\Diamond(f_\sigma \wedge (l{=}1)_\sigma)\big) = \mathbb{P}\big(\Diamond(f_{\sigma'} \wedge (l{=}1)_{\sigma'})\big)\Big) \wedge \Big(\mathbb{P}\big(\Diamond(f_\sigma \wedge (l{=}2)_\sigma)\big) = \mathbb{P}\big(\Diamond(f_{\sigma'} \wedge (l{=}2)_{\sigma'})\big)\Big)\Big).$$

However, when using a fair scheduler, the likely outcome of the race between the two assignments $l := 1$ and $l := 2$ depends on the initial value of $h$: the larger the initial value of $h$, the greater the probability that the final value of $l$ is 2. For example, it is easy to recognize in Fig. 6 that for the secret input $h = 0$ the final value is $l = 1$ with probability 1/4 and $l = 2$ with probability 3/4, but for the input $h = 1$ the final value is $l = 1$ with probability 1/16 and $l = 2$ with probability 15/16. Thus, it holds that for two independent executions with initial $h$ values 0 resp. 1 the larger $h$ value leads to a lower probability for $l = 1$ upon termination. I.e., this program does not satisfy $\varphi_{\mathsf{pni}}$.

Now, let us repair this system by allowing the scheduler to use *biased coins* for different secret input values $h$. Table 1 shows experimental results for different input value combinations, for




(a) $(h = 3)_\sigma, (h = 5)_{\sigma'}$. (b) $(h = 0)_\sigma, (h = 10)_{\sigma'}$. (c) $(h = 14)_\sigma, (h = 8)_{\sigma'}$.

Figure 7: Synthesized probability distribution for randomized scheduler.

| **Input** $h$ | **Running time** ($s$) | | | #white | #red | red | #samples |
|---|---|---|---|---|---|---|---|
| | **Alg.2-4** | **Alg.5-6** | **Total** | boxes | boxes | area | |
| $(0, 1)$ | 2.90 | 100.03 | 102.93 | 378 | 748 | 0.79 | 477 |
| $(0, 5)$ | 15.61 | 143.58 | 159.2 | 374 | 752 | 0.815 | 421 |
| $(0, 10)$ | 55.73 | 259.3 | 315.06 | 374 | 752 | 0.8164 | 480 |
| $(0, 15)$ | 113.58 | 459.60 | 573.18 | 377 | 749 | 0.711 | 413 |
| $(1, 2)$ | 8.33 | 114.55 | 122.88 | 368 | 758 | 0.706 | 425 |
| $(3, 5)$ | 31.95 | 204.42 | 236.38 | 411 | 715 | 0.831 | 496 |
| $(4, 8)$ | 72.23 | 397.91 | 470.14 | 371 | 755 | 0.6622 | 481 |
| $(8, 14)$ | 213.96 | 2924.61 | 3138.07 | 378 | 748 | 0.825 | 496 |

Table 1: Experimental results for thread scheduling.

which we want to determine parameters that assure probabilistic noninterference. The table shows for each considered pair of $h$-values the time spent in Algorithms 1–4 and Algorithms 5–6, the total running time, the number of returned white and red boxes (no green boxes have been detected), the percentage of the configuration domain covered by red boxes (i.e. provably non-satisfying area), and the number of satisfying configurations (samples) detected. Figure 7 shows the 2-dimensional plot of synthesized valid values of parameters $p$ and $q$ for for different pair values of $h$. As can be seen, as the values of $h$ converge (e.g., in Fig. 7a), the probabilities of $p$ and $q$ also converge, as the resulting DTMC is balanced. On the contrary, as the values of $h$ diverge (e.g., in Fig. 7b), the probabilities of $p$ and $q$ also diverge, as the resulting DTMC is more imbalanced.

## 4.4 Information Leakage in Dining Cryptographers

Three cryptographers gather around a table for dinner. The waiter informs them that the meal has been paid for by someone, who could be either one of the three cryptographers or the master. The cryptographers respect each other's privacy, but want to find out whether the master paid. In order to decide this, they execute the following two-stage protocol [12]:

- Each cryptographer flips a coin and informs only the cryptographer on the right about the outcome.
- Each cryptographer who did not pay for the dinner announces whether the two coins that it can see (the own flipped one and the one the left-hand neighbour flipped) are the same ("agree") or different ("disagree").

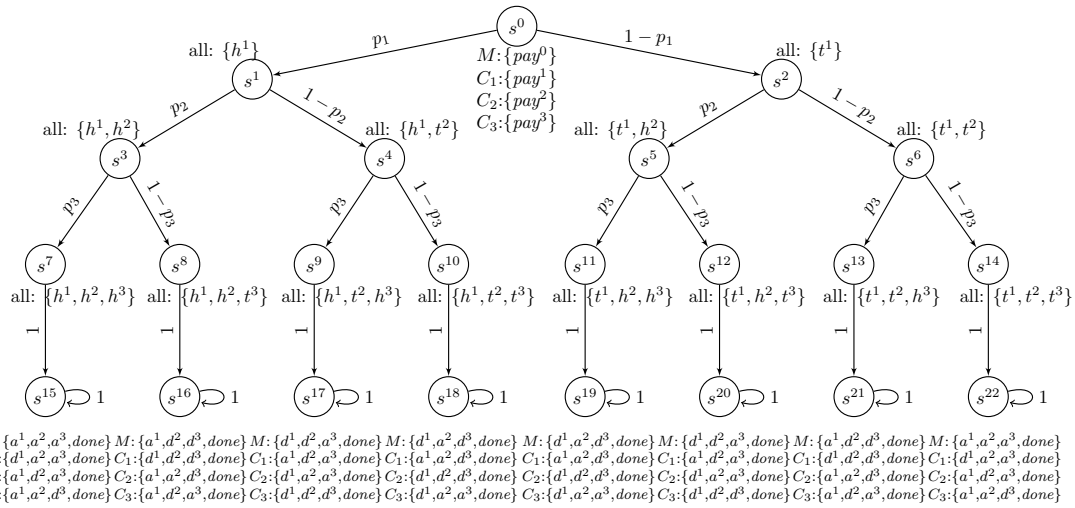



Figure 8: A parametric DTMC model for the dining cryptographers protocol with three cryptographers and three biased coins (with parameters $p_1$, $p_2$, and $p_3$, respectively) consists of four independent sub-PDTMCS, staying for the four cases who payed: the master ($M$), the first ($C_1$), second ($C_2$) resp. third ($C_3$) cryptographer. We depict all four PDTMCS in one illustration as they differ only in their state labeling. The labels $pay^M$ and $pay^i$, $i \in \underline{3}$ encode that the master resp. cryptographer $i$ payed; $t^i$ resp. $h^i$ encode tail resp. head flipped by cryptographer $i$; the labels $a^i$ resp. $d^i$ encode that cryptographer $i$ announced "agree" resp. "disagree"; finally, *done* stays for a terminated protocol. In the text, the above state identifiers $s^i$ are lower indexed with the cases to distinguish between $s^i_M$, $s^i_{C_1}$, $s^i_{C_2}$, $s^i_{C_3}$.

- However, if a cryptographer actually paid for dinner, then it instead states the opposite ("disagree" if the coins are the same and "agree" if the coins are different).

A parametric DTMC model of the protocol is illustrated in Figure 8. We are interested in deciding which parts of the valid parameter domain $[0,1]^3 \subseteq \mathbb{R}^3$ satisfy the following ReachHyperPCTL formula ($\oplus$ denotes the exclusive-or operator):

$$\varphi_{\mathsf{dc}} = \quad \forall\sigma.\, \forall\sigma'.\quad \Big((\bigvee_{i\in\underline{3}} pay^i_\sigma) \wedge (\bigvee_{i\in\underline{3}} pay^i_{\sigma'})\Big) \;\Rightarrow \tag{2}$$
$$\underbrace{\mathbb{P}\Big(\Diamond(done_\sigma \wedge (a^1_\sigma \oplus a^2_\sigma \oplus a^3_\sigma))\Big)}_{F_1} = \underbrace{\mathbb{P}\Big(\Diamond(done_{\sigma'} \wedge (a^1_{\sigma'} \oplus a^2_{\sigma'} \oplus a^3_{\sigma'}))\Big)}_{F_2}$$

In other words, if the master does not pay, then the different outcomes are observed with the same probabilities independently of the fact which cryptographer has paid. A careful reader recognizes that the above property holds for all parameters. Intuitively, independently of the flip outcomes, when ordered in a circle, the number of changes in the outcomes will be always even. Thus, the number of "agree"s will be even iff an even number of cryptographers lie. Therefore, when the master paid (zero lies) we have an even number of "agree"s, and when one of the cryptographers paid then one lies and we have an odd number.

To check this property, we follow Algorithm 1 and call first the SymbolicEncoding method from Algorithm 2 on the 2-ary self-composition of the PDTMC in Figure 8. The states of this

self-composition are pairs $(s^i_{T_1}, s^j_{T_2})$ with $i, j \in \underline{22}$ and $T_1, T_2 \in \{M, C_1, C_2, C_3\}$. Note that the self-composition is synchronous, i.e., each non-absorbing state has four successors; for example, state $(s^0_{C_1}, s^0_{C_2})$ has the successors (1) $(s^1_{C_1}, s^1_{C_2})$ with probability $p_1 \cdot p_1$, (2) $(s^1_{C_1}, s^2_{C_2})$ with probability $p_1 \cdot (1-p_1)$, (3) $(s^2_{C_1}, s^1_{C_2})$ with probability $(1-p_1) \cdot p_1$, and (4) $(s^2_{C_1}, s^2_{C_2})$ with probability $(1-p_1) \cdot (1-p_1)$.

Formula 2 is trivially satisfied by all states where the left-hand-side of the implication is false, i.e., the only relevant initial states (instantiating $\sigma$ and $\sigma'$) are $(s^0_{T_1}, s^0_{T_2})$ with $T_1, T_2 \in \{C_1, C_2, C_3\}$. Due to the synchronous nature of the self-composition, both runs starting in $\sigma$ resp. $\sigma'$ will execute the same number of steps, i.e. stay at the same "depth" in Figure 8. For all such state pairs, the probability expressions $F_1$ and $F_2$ in Formula 2 both simplify to 1, therefore the equality $F_1 = F_2$ holds independently from the parameter configuration.

Starting with the parameter domain $[0,1]^3$, our implementation reports that the whole box $[0,1]^3$ is green, without any splits. However, the (symbolic) transition matrix is quite large (we have 7744 states in the 2-ary self-composition and our implementation does not detect non-reachable states), it takes about 40 minutes running time to get this answer.

# 5 Related Work

The notion of hyperproperties was first introduced in [14] and temporal logics HyperLTL and HyperCTL$^*$ [13] have been proposed to capture particular classes of hyperproperties. There has been a lot of recent progress in automatically verifying [6, 15, 27–29] and monitoring [2, 9, 10, 25, 26, 30, 43] HyperLTL specifications. A growing set of tools support HyperLTL, including the model checker MCHyper [15, 29], the satisfiability checkers EAHyper [24] and MGHyper [22], and the runtime monitoring tool RVHyper [25]. Synthesis techniques for HyperLTL has been studied in [7, 8, 23]. HyperPCTL [1] is the first temporal logic proposed to express probabilistic hyperproperties, adding explicit and simultaneous quantification over multiple traces to PCTL. Statistical model checking techniques for probabilistic hyperproperties were proposed in [45].

While the work in [1] provides a model checking algorithm for HyperPCTL properties over DTMCs, here, we focus on parametric DTMCs and the parameter synthesis problem. A parametric DTMC [16,35] is a special class of Markov models, where some the transition probabilities (or rates) are not known a-priori and are parameter-dependent. These models can be adopted to analyze systems with stochastic uncertainty due to the impossibility to measure certain quantities (e.g., fault rates, packet loss ratios, etc.). In [16], Daws proposed an approach to express the probability to reach the target state as a rational function with the domain in the parameter space. This symbolic approach has been exploited in several model checking algorithms for parametric probabilistic Markov chains [5, 18, 31, 33, 36, 38] and efficiently implemented in PARAM1 and PARAM2 tools [31]. There are also *type-theoretic* approaches (e.g., [39]) for synthesizing protocols for differential privacy, which is out of scope of this paper.

The *parameter synthesis* problem consists in exploiting the generated rational function to find the parameter values that would maximize or minimize the probability to reach the target state [5]. The price to pay for these techniques is the increasing complexity of the rational functions in the presence of large models [35], causing the parameter synthesis to be also very computationally expensive. However, the introduction of new efficient heuristics [3,11,18,33,36, 38,41] has helped to alleviate this problem by supporting the parameter synthesis for quite large models. For example, PROPhESY [18] supports incremental automatic parameter synthesis for parametric Markov chains w.r.t. reachability properties expressed in PCTL and it exploits SMT techniques to determine *safe* and *unsafe* regions of the parameter space. In contrast with PROPhESY, our approach enables the parameter synthesis for the richer class of formal

specifications defined by ReachHyperPCTL, a fragment of HyperPCTL.

It is worth pointing out that other approaches based on *parameter lifting* [11,38], although scalable, cannot be adapted to our framework. For example, the work in [38] propose to: (a) relax the dependency among the parameters by introducing free variables, (b) replace parametric transitions by nondeterministic choices of extremal values, and (c) analyze the resulting parameter-free Markov decision process by computing lower and upper bounds on probabilities of regions in the parameter space. This approach is restricted to work only for probabilistic properties with an *eventually* operator, while our ReachHyperPCTL supports other operators such as the *until*. Furthermore, ReachHyperPCTL allows to compare the value of two probabilistic operators and this feature makes it infeasible to use parameter lifting that provides probability intervals.

# 6 Conclusion and Future Work

In this paper, we focused on *probabilistic hyperproperties*, which express stochastic relations between multiple execution traces of a probabilistic model. The *parameter synthesis* problem takes as input a parametric discrete-time Markov-chain and a probabilistic hyperproperty, and asks for valid parameter values for which the induced discrete-time Markov chain satisfies the probabilistic hyperproperty. Our synthesis algorithm works in two steps. In the first, it computes symbolic conditions for satisfying the formula, involving rational functions on the set of parameters. In the second step, identify regions of satisfying parameter configurations by decomposing the domain of parameter configurations and identify smaller regions in which either all or none of the configurations lead to the satisfaction of the input formula. We demonstrated how our algorithm works on several examples: randomized response, probabilistic conformance, probabilistic noninterference, and the dining cryptographers.

As for future work, making our prototypical implementation more efficient would clearly lead to better scalability. Further improvements could be achieved by e.g., exploiting the existence of symmetries. Another natural extension is to consider more expressive logics such as full HyperPCTL and HyperPCTL$^*$ [44].

# References

[1] E. Ábrahám and B. Bonakdarpour. HyperPCTL: A temporal logic for probabilistic hyperproperties. In *Proc. of the 15th Int. Conf. on Quantitative Evaluation of Systems (QEST'18)*, volume 11024 of *LNCS*, pages 20–35. Springer, 2018.

[2] S. Agrawal and B. Bonakdarpour. Runtime verification of $k$-safety hyperproperties in HyperLTL. In *Proc. of the IEEE 29th Computer Security Foundations Symp. (CSF'16)*, pages 239–252. IEEE, 2016.

[3] Sebastian Arming, Ezio Bartocci, Krishnendu Chatterjee, Joost-Pieter Katoen, and Ana Sokolova. Parameter-independent strategies for pmdps via pomdps. In *Proc. of the 15th Int. Conf. on Quantitative Evaluation of Systems (QEST'18)*, volume 11024 of *LNCS*, pages 53–70. Springer, 2018.

[4] C. Baier and J-.P. Katoen. *Principles of Model Checking*. MIT Press, 2008.

[5] E. Bartocci, R. Grosu, P. Katsaros, C. R. Ramakrishnan, and S. A. Smolka. Model repair for probabilistic systems. In *Proc. of the 17th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'11)*, volume 6605 of *LNCS*, pages 326–340. Springer, 2011.

[6] B. Bonakdarpour and B. Finkbeiner. The complexity of monitoring hyperproperties. In *Proc. of the 31st IEEE Computer Security Foundations Symp. (CSF'18)*, pages 162–174, 2018.

[7] B. Bonakdarpour and B. Finkbeiner. Program repair for hyperproperties. In *Proc. of the 17th Symp. on Automated Technology for Verification and Analysis (ATVA'19)*, volume 11781 of *LNCS*, pages 423–441. Springer, 2019.

[8] B. Bonakdarpour and B. Finkbeiner. Controller synthesis for hyperproperties. In *Proceedings of the IEEE 32th Computer Security Foundations (CSF)*, 2020. To appear.

[9] B. Bonakdarpour, C. Sánchez, and G. Schneider. Monitoring hyperproperties by combining static analysis and runtime verification. In *Proc. of the 8th Leveraging Applications of Formal Methods, Verification and Validation (ISoLA'18)*, volume 11245 of *LNCS*, pages 8–27. Springer, 2018.

[10] N. Brett, U. Siddique, and B. Bonakdarpour. Rewriting-based runtime verification for alternation-free HyperLTL. In *Proc. of the 23rd Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'17)*, volume 10206 of *LNCS*, pages 77–93. Springer, 2017.

[11] M. Ceska, F. Dannenberg, N. Paoletti, M. Kwiatkowska, and L. Brim. Precise parameter synthesis for stochastic biochemical systems. *Acta Informatica*, 54(6):589–623, 2017.

[12] D. Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1:65–75, 1988.

[13] M. R. Clarkson, B. Finkbeiner, M. Koleini, K. K. Micinski, M. N. Rabe, and C. Sánchez. Temporal logics for hyperproperties. In *Proc. of the 3rd Conf. on Principles of Security and Trust (POST'14)*, volume 8414 of *LNCS*, pages 265–284. Springer, 2014.

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

[15] N. Coenen, B. Finkbeiner, C. Sánchez, and L. Tentrup. Verifying hyperliveness. In *Proc. of the 31st Int. Conf. on Computer Aided Verification (CAV'19)*, volume 11561 of *LNCS*, pages 121–139. Springer, 2019.

[16] C. Daws. Symbolic and parametric model checking of discrete-time Markov chains. In *Proc. of the First Int. Conf. on Theoretical Aspects of Computing (ICTAC'04)*, volume 3407 of *LNCS*, pages 280–294. Springer, 2004.

[17] L. M. de Moura and N. Bjørner. Z3: An efficient SMT solver. In *Proc. of the 14th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'08)*, volume 4963 of *LNCS*, pages 337–340. Springer, 2008.

[18] C. Dehnert, S. Junges, N. Jansen, F. Corzilius, M. Volk, H. Bruintjes, J.-P. Katoen, and E. Ábrahám. PROPhESY: A probabilistic parameter synthesis tool. In *Proc. of the 27th Int. Conf. on Computer-Aided Verification (CAV'15)*, volume 9206 of *LNCS*, pages 214–231. Springer, 2015.

[19] C. Dehnert, S. Junges, J.-P. Katoen, and M. Volk. A Storm is coming: A modern probabilistic model checker. In *Proc. of the 29th Int. Conf. on Computer Aided Verification (CAV'17)*, volume 10427 of *LNCS*, pages 592–600. Springer, 2017.

[20] C. Dwork, F. McSherry, K. Nissim, and A. D. Smith. Calibrating noise to sensitivity in private data analysis. *Journal of Privacy Confidentiality*, 7(3):17–51, 2016.

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

[22] B. Finkbeiner, C. Hahn, and T. Hans. MGHyper: Checking satisfiability of HyperLTL formulas beyond the $\exists^*\forall^*$ fragment. In *Proc. of the 16th Int. Symp. on Automated Technology for Verification and Analysis (ATVA'18)*, volume 11138 of *LNCS*, pages 521–527. Springer, 2018.

[23] B. Finkbeiner, C. Hahn, P. Lukert, M. Stenger, and L. Tentrup. Synthesizing reactive systems from hyperproperties. In *Proc. of the 30th Int. Conf. on Computer Aided Verification (CAV'18)*, volume 10981 of *LNCS*, pages 289–306. Springer, 2018.

[24] B. Finkbeiner, C. Hahn, and M. Stenger. EAHyper: Satisfiability, implication, and equivalence checking of hyperproperties. In *Proc. of the 29th Int. Conf. on Computer Aided Verification (CAV'17)*, volume 10427 of *LNCS*, pages 564–570. Springer, 2017.

[25] B. Finkbeiner, C. Hahn, M. Stenger, and L. Tentrup. RVHyper: A runtime verification tool for temporal hyperproperties. In *Proc. of the 24th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'18)*, volume 10806 of *LNCS*, pages 194–200. Springer, 2018.

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

[27] B. Finkbeiner, C. Hahn, and H. Torfah. Model checking quantitative hyperproperties. In *Proc. of the 30th Int. Conf. on Computer Aided Verification (CAV'18)*, volume 10981 of *LNCS*, pages 144–163. Springer, 2018.

[28] B. Finkbeiner, Ch. Müller, H. Seidl, and E. Zalinescu. Verifying security policies in multi-agent workflows with loops. In *Proc. of the 2017 ACM SIGSAC Conf. on Computer and Communications Security (CCS'17)*, pages 633–645. ACM, 2017.

[29] B. Finkbeiner, M. N. Rabe, and C. Sánchez. Algorithms for model checking HyperLTL and HyperCTL*. In *Proc. of the 27th Int. Conf. on Computer Aided Verification (CAV'15)*, volume 9206 of *LNCS*, pages 30–48. Springer, 2015.

[30] C. Hahn, M. Stenger, and L. Tentrup. Constraint-based monitoring of hyperproperties. In *Proc. of the 25th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'19)*, volume 11428 of *LNCS*, pages 115–131. Springer, 2019.

[31] E. M. Hahn, T. Han, and L. Zhang. Probabilistic reachability for parametric Markov models. *Int. Journal on Software Tools for Technology Transfer*, 13(1):3–19, 2011.

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

[33] N. Jansen, F. Corzilius, M. Volk, R. Wimmer, E. Ábrahám, J.-P. Katoen, and B. Becker. Accelerating parametric probabilistic verification. In *Proc. of the 11th Int. Conf. on Quantitative Evaluation of Systems (QEST'14)*, volume 8657 of *LNCS*, pages 404–420. Springer, 2014.

[34] D. E. Knuth and A. C.-C. Yao. *Algorithms and Complexity: New Directions and Recent Results*, chapter The Complexity of Nonuniform Random Number Generation. Academic Press, 1976.

[35] R. Lanotte, A. Maggiolo-Schettini, and A. Troina. Parametric probabilistic transition systems for system design and analysis. *Formal Aspects of Computing*, 19(1):93–109, 2007.

[36] S. Pathak, E. Ábrahám, N. Jansen, A. Tacchella, and J.-P. Katoen. A greedy approach for the efficient repair of stochastic models. In *Proc. of the 7th Int. NASA Formal Methods Symp. (NFM'15)*, volume 9058 of *LNCS*, pages 295–309. Springer, 2015.

[37] pycarl. https://moves-rwth.github.io/pycarl/.

[38] T. Quatmann, C. Dehnert, N. Jansen, S. Junges, and J.-P. Katoen. Parameter synthesis for Markov models: Faster than ever. In *Proc. of the 14th Int. Symp. on Automated Technology for Verification and Analysis (ATVA'16)*, volume 9938 of *LNCS*, pages 50–67. Springer, 2016.

[39] C. Smith and A. Albarghouthi. Synthesizing differentially private programs. *Proc. of the ACM on Programming Languages (PACMPL'19)*, 3(ICFP):94:1–94:29, 2019.

[40] G. Smith. Probabilistic noninterference through weak probabilistic bisimulation. In *Proc. of the 16th IEEE Computer Security Foundations Workshop (CSFW'03)*, pages 3–13. IEEE, 2003.

[41] J. Spel, S. Junges, and J.-P. Katoen. Are parametric Markov chains monotonic? In *Proc. of the 17th Int. Symp. on Automated Technology for Verification and Analysis (ATVA'19)*, volume 11781 of *LNCS*, pages 479–496. Springer, 2019.

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

[43] S. Stucki, C. Sánchez, G. Schneider, and B. Bonakdarpour. Gray-box monitoring of hyperproperties. In *Proc. of the 3rd World Congress on Formal Methods (FM'19)*, volume 11800 of *LNCS*, pages 406–424. Springer, 2019.

[44] Y. Wang, S. Nalluri, B. Bonakdarpour, and M. Pajic. Statistical model checking for probabilistic hyperproperties. *CoRR*, abs/1902.04111, 2019.

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