Transient Reward Approximation for Continuous-Time Markov Chains

Ernst Moritz Hahn
State Key Laboratory of Computer Science,
Institute of Software, Chinese Academy of Sciences

Holger Hermanns
Saarland University
Germany

Ralf Wimmer
Albert-Ludwigs-Universität
Freiburg, Germany

Bernd Becker
Albert-Ludwigs-Universität
Freiburg, Germany

Abstract

We are interested in the analysis of very large continuous-time Markov chains (CTMCs) with many distinct rates. Such models arise naturally in the context of reliability analysis, e. g., of computer network performability analysis, of power grids, of computer virus vulnerability, and in the study of crowd dynamics. We use abstraction techniques together with novel algorithms for the computation of bounds on the expected final and accumulated rewards in continuous-time Markov decision processes (CTMDPs). These ingredients are combined in a partly symbolic and partly explicit (symblicit) analysis approach. In particular, we circumvent the use of multi-terminal decision diagrams, because the latter do not work well if facing a large number of different rates. We demonstrate the practical applicability and efficiency of the approach on two case studies.

Acknowledgements This work was partly supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Centre "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS), the NWO-DFG bilateral project ROCKS, the ERC Advanced Grant VERIWARE, the National Natural Science Foundation of China (NSFC) under grant No. 61350110518 and No. 61450110461, the CDZ project CAP (GZ 1023), the Chinese Academy of Sciences Fellowship for International Young Scientists (Grant No. 2013Y1GB0006), and has received funding from the European Union Seventh Framework Programme under grant agreement number 295261 as part of the MEALS project and under grant agreement number 318490 as part of the SENSATION project.

Part of this work was done while Ernst Moritz Hahn was with Saarland University and with the University of Oxford, United Kingdom

We thank Martin Neuhäußer and Lijun Zhang for fruitful discussions
This article appeared in IEEE Transactions on Reliability, Volume 64, Issue 4.

Abbreviations

ACTMC	abstract continuous-time Markov chain
BDD	binary decision diagram
CD	time-abstract, history-abstract, counting, deterministic scheduler
CR	time-abstract, history-abstract, counting, randomised scheduler
CSL	continuous stochastic logic
CTMC	continuous-time Markov chain
CTMDP	continuous-time Markov decision process
DTMC	discrete-time Markov chain
DTMDP	discrete-time Markov decision process
ECTMC	extended abstract continuous-time Markov chain
EVBDD	edge-valued decision diagram
HR	time-abstract, history-dependent, randomised scheduler
MDD	multiple-valued decision diagram
MTBDD	multi-terminal binary decision diagram
OBDD	reduced ordered binary decision diagram
PM	Prism model
symblicit	method combining symbolic and explicit aspects

ZDD	zero-suppressed decision diagram

Notation

μ	probability distribution
$Distr(A)$	set of probability distributions over the set A
$\mathcal{D}$	discrete-time Markov model
S	states of a Markov model
$\mathbf{P}$	probability matrix of a Markov model
$X^{\mathcal{D},s_0}$	stochastic process of a Markov model
s	state of a Markov model
(Ω, Σ)	measurable space
Pr	probability measure
$\mathbf{E}$	expectation
$\mathbf{R}$	rate matrix of a Markov model
$\mathbf{u}$	uniformisation rate
Act	action set of a Markov decision process
α	action of a Markov decision process
$\widehat{Act}$	action set of an extended abstract Markov chain
$\mathbf{I}^\ell, \mathbf{I}^u$	intervals of an extended abstract Markov chain
$\hat{\alpha}$	action of an extended abstract Markov chain
$[\![C]\!]$	continuous-time Markov decision process semantics of an extended abstract Markov chain
σ	scheduler of a Markov decision process
β	history of a Markov model
Σ_{HR}	set of time-abstract, history-dependent, randomised schedulers
Σ_{CR}	set of time-abstract, history-abstract, counting, randomised schedulers
Σ_{CD}	set of time-abstract, history-abstract, counting, deterministic schedulers
σ	scheduler
$\mathbf{r} = (\mathbf{r}_c, \mathbf{r}_f)$	reward structure with cumulative reward rate $\mathbf{r}_c$, and final reward value $\mathbf{r}_f$
$\mathbf{r}_f^{\max}$	maximal final reward value over model states
$\mathbf{r}_c^{\max}$	maximal cumulative reward rate over model states
$\mathbf{t}$	time bound
$\mathbf{V}$	reward value of a stochastic process or Markov model
m	Prism model
Var	variables of a Prism model
$init$	initial state of a Prism model
C	commands of a Prism model
$succ$	successors function of a Prism model
R_c	cumulative rewards of a Prism model
R_f	instantaneous rewards of a Prism model
$\overline{succ}$	qualitative successor function of a Prism model
S_m	reachable states of a Prism model
C_m	induced CTMC of a Prism model
$\mathbf{r}_m$	induced reward structure of a Prism model
$\mathfrak{P}$	partitioning of state space of a Prism model
$\mathfrak{z}$	abstract state
V	BDD variables
x	BDD variable
b	graph of BDD
N	set of nodes of a BDD
n_{root}	root node of a BDD
$\mathsf{v}(\mathsf{n})$	label of a BDD terminal node n
$\mathsf{h}(\mathsf{n})$	high successor of a BDD node n
$\mathsf{l}(\mathsf{n})$	low successor of a BDD node n
v	variable valuation of a BDD
Val	set of variable valuations of a given BDD
$[\![\mathsf{b}]\!]$	function represented by a BDD b
$\mathsf{bdd}_0, \mathsf{bdd}_1$	zero, and one BDDs
$\mathsf{b}_{\mathfrak{P}}$	BDD representation of a state space partitioning
$\mathfrak{V}$	set of BDD variables to encode the number of an abstract state
$\mathfrak{x}$	BDD variable to encode the number of an abstract state
ϕ_λ	Poisson distribution with rate λ
ψ_λ	cumulative Poisson distribution with rate λ

ε	precision to compute value bounds
Dom	domain of a partial function

1 Introduction

The analysis of large Markov chains is a recurring challenge in many important areas ranging from computer network dependability and performance [71, 36] to quantitative security [52]. To evaluate properties of such systems, a standard approach is to perform numerical analysis, nowadays often embedded in a stochastic model checker [55, 47, 20, 27]. At its core, the model checker has to operate with a very large matrix induced by the Markov chain. In this context, the use of symbolic representations, in particular variations of decision diagrams, such as multi-terminal decision diagrams (MTBDDs) [67, 41], multiple-valued decision diagrams (MDDs) [79], or zero-suppressed decision diagrams (ZDDs) [60], have made it possible to store and manipulate very large matrices in a symbolic manner (either of transition rates or just of adjacency). Many of the applications occurring in practice lead to very large continuous-time Markov chains (CTMCs) that nevertheless contain only a very small number of different transition rates. This is a primary reason why decision

diagrams, where distinct rates are stored as distinct values in the structure, are effective. Whenever there are many pairwise different rates occurring, the decision diagram degenerates to a decision tree, and thus its size explodes. Edge-valued binary decision diagrams (EVBDDs) [59] often can avoid this representation explosion, at the price of a more involved reconstruction of matrix entries. This trade-off makes them less suited for direct numerical computations, as needed for the model checking of CTMCs. Therefore, models with a large number of different rates are a notorious problem for symbolic representations, and hence for the stochastic model checkers available to date.

However, there is a growing spectrum of important applications that give rise to excessive numbers of distinct rates. Computer network performability analysis [37, 22, 2, 32, 76], power grid stability [72, 35], crowd dynamics [62, 63], as well as (computer) virus epidemiology [84, 86, 85] are important examples where Markov models are huge, and rates change from state to state. The study of these phenomena is of growing importance for the assurance of their reliability. Several of these examples can in some way be regarded as Markov population models [38, 43], where the rates change with population counts, similar to models appearing in systems biology [64], and also in classical performance and dependability engineering [37, 22].

This paper targets the analysis of transient properties of CTMCs with both a large number of states as well as a large number of distinct transition rates. It presents a combination of abstraction techniques, an explicit representation of a small abstract model, and symbolic techniques. The latter use reduced ordered binary decision diagrams (OBDDs), not MTBDDs or MDDs. As our method involves both symbolic and explicit state space representations, we call it *symblicit*. The abstraction method relies on visiting all concrete states of the abstract model to obtain bounds on the transition matrix, but without having to store the state space explicitly. We also present ideas how to speed up this admittedly time-consuming process. On the one hand, the approach can be seen as a continuation of our previous work on symblicit algorithms [81, 23]. On the other hand, we harvest work done on the abstraction of Markov chains to abstract Markov chains or Markov decision processes [50, 73, 46, 12, 26, 48, 42].

A number of related methods exist. Our solution method uses results for explicit-state model analysis using *uniformisation (randomisation)* [45, 34]. In this method, a continuous-time Markov model is transformed into a discrete-time Markov model and a Poisson process. Intuitively, the discrete-time model describes in which state the model resides after a state change, while the Poisson process describes the process of the state changes. Analyses using uniformisation then usually perform computations to obtain intermediate results on the discrete-time model, and later combine these results by weighting them using the Poisson process. As the number of possible state changes is unbounded, this process needs to be truncated, thereby only considering a finite number of potential state changes. However, the probability residing beyond that truncation point can be bounded [31], so that the precision of the results obtained via uniformisation can be precomputed. Uniformisation is well understood, numerically stable, and generally performs well in practice for non-stiff models. Because of this, a number of contributions have since been based on this method. Advanced methods for explicit-state Markov reward models have been pioneered by Trivedi et

al. [77]. Based on a variant of uniformisation using quasi-stationary detection, Carrasco has developed a method [18] to speed up the computation of transient reward properties for large stiff models where the state space can be partitioned into a transient set and an absorbing state set. Models with a similar structure are amenable to the efficient analysis of cumulative reward properties with a very general state-based reward notation [19].

We also build on many ideas for analyses using abstract Markov chains by Klink et al. [50, 46]. This paper extends their works by describing a widely applicable abstraction method, and also by handling more general transient properties.

MTBDD-based methods [67, 41] work well for some models, but have the disadvantages described above. The method of Wan et al. [79] uses a slightly different data structure to represent concrete models, and focuses on steady-state properties rather than transient ones. It does not rely on Markov decision models, and, though it works well in practice for certain model classes, it cannot guarantee safe bounds for properties of the concrete model.

There are techniques in which a symbolic representation of the transition matrix is used, but in which values assigned to the states of the model (such as probabilities) have to be stored explicitly and separately for each state of the model. Examples include the so-called hybrid method [67, 56], other variants of decision diagrams [60], and also methods using Kronecker representations [25, 14, 5, 30, 16]. These kinds of methods are more precise, and might be faster than the method we propose. They are however not applicable in case the state space is excessively large, too large to store one value per state.

Smith [73] developed means for the compositional abstraction of CTMCs given in a process calculus. This way, he obtains an abstract Markov chain, which is then analysed by a method of Baier et al. [4] to obtain bounds for the time-bounded reachability probability. Our approach uses a different abstraction method, and can handle a more general class of properties.

A paper by Buchholz [12] describes how bounds on long-run average (thus, non-transient) properties can be obtained from abstract Markov chains. It is based on a combination of policy and value iteration [68], and discusses the applicability of several variants of these methods on typical examples from queueing theory and performance evaluation.

The magnifying-lens abstraction [26] by de Alfaro et al. is similar to our approach in that it also builds on (repeated) visits of concrete model states without storing the whole concrete state space. It discusses a different model, discrete-time Markov decision processes (DTMDPs), and a different property, time-unbounded reachability probabilities.

D'Argenio et al. [24] discuss how DTMDPs given as MTBDDs can be abstracted to obtain a smaller abstract model, which is also a DTMDP, but small enough to be represented explicitly. In addition, a heuristic abstraction refinement method is presented. The target there was to obtain bounds for unbounded reachability probabilities. Works by Hermanns et al. [42] and by Kattenbelt et al. [48] later developed methods to use probabilistic games to provide tighter value bounds and predicate abstraction to handle larger or even infinitely large models, as well as refinement methods based on these frameworks. In contrast to the state of the art for discrete-time models, the discussed refinement method we consider is more preliminary.

Other methods work with a finite subset of concrete states of the model under consideration, rather than subsuming concrete states in abstract ones. There exists a wide range of methods based on this principle for the analysis of CTMCs [33, 78, 65, 39]. Recently, this approach was extended to infinite-state Markov decision processes [13]. Here, two finite submodels are constructed which guarantee to bound the values over all policies from below and above. They can also be used to obtain a policy which is ε-optimal in the original model.

Such methods are applicable if during a transient analysis the probability mass stays concentrated on a small subset of states at each point of time. If, however, the probability spreads evenly among too many states of the model, then such methods are not appropriate. The reason is that, in this case, either too many states need to be stored, or a too large amount of the probability mass is lost as too many states have to be disregarded.

In Section 2, we provide basic notations, and describe the symbolic data structures used for the later abstraction. We also describe the formal models we use, as well as the properties we are interested in. Section 3 describes algorithms to efficiently obtain an abstract model from a description of a concrete model, and discusses how they can be used to bound properties of the concrete model.

In Section 4, we apply this method on two case studies from the area of computer network performability analysis, thus to show its practical applicability. Finally, Section 5 concludes the paper.

2 Preliminaries

This section introduces basic notations, and formally defines the models and data structures that are used in the later parts of the paper.

In Subsection 2.1, we discuss the stochastic models that build the theoretical foundation of the method described in this paper. We start by describing Markov chains, the mechanism in which the models to analyse are formulated originally (Definitions 1 and 2). Next, we define Markov decision processes, which extend Markov chains by nondeterministic decisions (Definitions 3 through 5). We then state a model type which allows a compact representation of Markov decision processes with an infinite number of nondeterministic choices per state, and which we will later use to obtain abstractions of Markov chains (Definition 6). Afterwards, we discuss how the nondeterminism of Markov decision processes can be resolved using an entity called scheduler (Definitions 7 through 10). This process is necessary to obtain a stochastic process, which is needed to reason about their properties. Finally, we assign rewards to the states of the discussed stochastic models (Definition 11). These reward structures allow us to define time-dependent reward values (Definition 12), which allow us to express a wide variety of interesting properties.

Subsection 2.2 discusses a high-level specification language (Definition 13), the semantics of which is again a stochastic model (Definition 14). Such a language allows us to express stochastic models in a compact way, and is thus both more memory-efficient and easier to read by humans than an explicit state-wise representation. However, in contrast to an explicit-state representation (or certain symbolic representations), it is not amenable to a direct analysis of its properties. In this paper, we target to avoid constructing the semantics of such high-level models explicitly. Instead, we define abstract state spaces (Definition 15) in which we subsume sets of concrete states to abstract states. We emphasise that, with the method of this paper, we do not have to explicitly store all such concrete states at the same time to generate abstractions.

In Subsection 2.3, we describe the concrete data structure we are going to use to store such a partitioning in a compact way (Definition 19). In addition, we use this data structure to store whether there is a non-zero rate between two states of the semantics of a high-level model (Definition 18). For our method, we do not need to store concrete values of non-zero rates in this representation, however. Thus, we can use simple OBDDs (Definitions 16 and 17), because this data structure already fulfils these requirements.

2.1 Stochastic Models

A *distribution* over a finite or countable set A is a function $\mu\colon A \to [0,1]$ such that $\sum_{a\in A}\mu(a) = 1$. By $\mathit{Distr}(A)$, we denote the set of all distributions over A.

The simplest stochastic model we consider is as follows.

Definition 1 *A* discrete-time Markov chain (DTMC) *is a tuple* $\mathcal{D} = (S, \mathbf{P})$ *where*

- S *is a finite set of* states*, and*
- $\mathbf{P}\colon (S \times S) \to [0,1]$ *is the* probability matrix *such that* $\sum_{s'\in S}\mathbf{P}(s,s') = 1$ *for all* $s \in S$.

By $X^{\mathcal{D},s_0}\colon (\Omega_{\mathcal{D}}\times\mathbb{N}) \to S$ *with* $s_0 \in S$ *we denote the unique stochastic process [74] of* $\mathcal{D}$ *with initial state* s_0*, where* $\Omega_{\mathcal{D}}$ *is the sample space to be used.*

The time in a DTMC proceeds in discrete steps, and in each step a transition with non-zero probability is taken. At step 0 the model starts in a given initial state $s_0 \in S$. The model moves to the next state, and will be in s_1 with probability $\mathbf{P}(s_0, s_1)$ for all $s_1 \in S$. From there, again the next state is chosen according to $\mathbf{P}$, and so on.

By Pr, we denote the *probability measure* on the measurable spaces $(\Omega_{\mathcal{D}}, \Sigma_{\mathcal{D}})$ of the DTMC $\mathcal{D}$ under consideration, with sample space $\Omega_{\mathcal{D}}$, and set $\Sigma_{\mathcal{D}}$ of events, which is defined by the standard cylinder set construction over finite paths [49]. For instance, $\Pr(X_n^{\mathcal{D},s_0} = s_1 \vee X_{n+1}^{\mathcal{D},s_0} = s_2)$ describes the probability that, having started in state s_0, in step n we are in s_1, or in step $n+1$ we are in s_2. For a measurable function $X\colon \Omega_{\mathcal{D}} \to \mathbb{R}$, we thus also have an *expectation* $\mathbf{E}(X) \stackrel{\text{def}}{=} \int_{\Omega_{\mathcal{D}}} X(\omega)\Pr(\mathrm{d}\omega)$. For instance, consider $X \stackrel{\text{def}}{=} \sum_{i=0}^{n-1}(f \circ X_i^{\mathcal{D},s_0})$ such that $f(s_1) \stackrel{\text{def}}{=} 1$, and

$f(s) \stackrel{\text{def}}{=} 0$ for $s \neq s_1$. Then $\mathbf{E}(X)$ denotes the average number of steps within the first n steps in which the DTMC is in s_1, under the condition that we started in s_0.

We now discuss our basic stochastic model described informally in the introduction.

Definition 2 *A* (uniform) continuous-time Markov chain (CTMC) *is a tuple* $C = (S, \mathbf{R})$ *where*

- S *is a finite set of* states*, and*
- $\mathbf{R}\colon (S \times S) \to \mathbb{R}_{\geq 0}$ *is the* rate matrix *such that there is a* uniformisation rate $\mathbf{u}(C) > 0$ *with* $\sum_{s' \in S} \mathbf{R}(s, s') = \mathbf{u}(C)$ *for all* $s \in S$.

If C is clear from context, we write $\mathbf{u}$ instead of $\mathbf{u}(C)$. In a *non-uniform CTMC*, the requirement $\sum_{s' \in S} \mathbf{R}(s, s') = \mathbf{u}(C)$ does not hold for all states. Every finite non-uniform CTMC can be transformed into an equivalent uniform CTMC with the same stochastic behaviour by increasing $\mathbf{R}(s, s)$ such that the total sum is the same for all states [74]. We require uniformity only for ease of presentation; it does not restrict the applicability of the methods developed here to general CTMCs.

The behaviour of a CTMC $C = (S, \mathbf{R})$ is similar to a DTMC. However, the durations until state changes are now real numbers. They are chosen according to statistically independent negative exponential distributions with parameter $\mathbf{u}$. Thus, the probability that a state change takes place within time t is $1 - \mathrm{e}^{-\mathbf{u} \cdot t}$. The successor state is then selected according to the distribution $\mu\colon S \to [0, 1]$ with $\mu(s') = \mathbf{R}(s_0, s')/\mathbf{u}$ for all $s' \in S$. We assume that the process runs until a certain point of time $\mathbf{t}$ is reached. By $X^{C, s_0}\colon (\Omega_C \times \mathbb{R}_{\geq 0}) \to S$ with $s_0 \in S$, we denote the uniquely defined stochastic process [74] of C with initial state s_0, where Ω_C is the sample space to be used. As for DTMCs, we assume that we have probability measures and expectations on the sample spaces.

Example 1 *In Fig. 1(a), we give an example for a CTMC. We represent its states as circles, and non-zero rates between states are given as arrows labelled with the rates. The uniformisation rate is* 6*.*

We need to specify another discrete- and a continuous-time model [44, 6], which will later be used to abstract large CTMCs. In addition to stochastic behaviour, these models also feature a *nondeterministic* choice over the successor distributions. Nondeterministic choices are choices which cannot be assigned a probability a priori. Instead, different stochastic behaviours result according to the resolution of the nondeterminism.

Definition 3 *A* discrete-time Markov decision process (DTMDP) *is a tuple* $\mathcal{D} = (S, Act, \mathbf{P})$ *where* S *is as in Definition 1,*

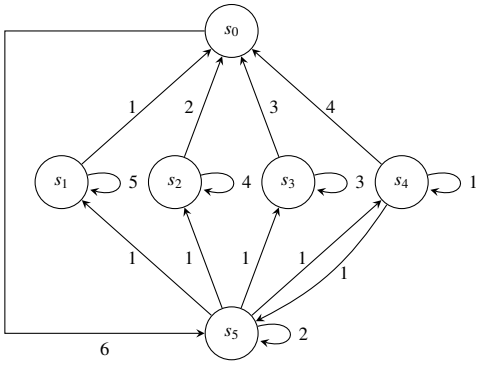
- Act *is a set of* actions*, and*
- $\mathbf{P}\colon (S \times Act \times S) \to [0, 1]$ *is the* probability matrix *such that* $\sum_{s' \in S} \mathbf{P}(s, \alpha, s') \in \{0, 1\}$ *for all* $s \in S$ *and* $\alpha \in Act$.

For $s \in S$*, we denote the set of* enabled *actions with* $Act(s) \stackrel{\text{def}}{=} \{\alpha \in Act \mid \sum_{s' \in S} \mathbf{P}(s, \alpha, s') = 1\}$*. We require that either* $|Act| < \infty$*, or that for all* $s \in S$ *and all* $p\colon S \to \mathbb{R}_{\geq 0}$ *the set* $\{\sum_{s' \in S} \mathbf{P}(s, \alpha, s') \cdot p(s') \mid \alpha \in Act(s)\}$ *is compact.*

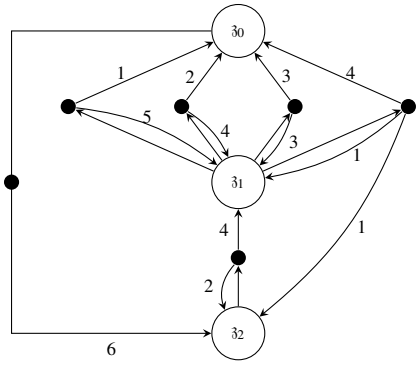
The behaviour of a DTMDP is such that, upon entering a state $s \in S$, an action $\alpha \in Act(s)$, or possibly a distribution over actions, is chosen. This choice determines the probabilities of the state the model moves to in the next time step. Notice that we do indeed allow uncountably many actions, with the given restriction.

Definition 4 *A* (uniform) continuous-time Markov decision process (CTMDP) *is a tuple* $C = (S, Act, \mathbf{R})$*. Here,* S *and* Act *are as in Definition 3. By* $\mathbf{R}\colon (S \times Act \times S) \to \mathbb{R}_{\geq 0}$*, we denote the* rate matrix *such that there is a fixed value* $\mathbf{u}(C)$ *with* $\sum_{s' \in S} \mathbf{R}(s, \alpha, s') \in \{0, \mathbf{u}(C)\}$ *for all* $s \in S$ *and* $\alpha \in Act$*. If* C *is clear from the context, we write* $\mathbf{u}$ *instead of* $\mathbf{u}(C)$*. For* $s \in S$*, we denote the set of* enabled *actions with* $Act(s) \stackrel{\text{def}}{=} \{\alpha \in Act \mid \sum_{s' \in S} \mathbf{R}(s, \alpha, s') = \mathbf{u}\}$*. We require that either* $|Act| < \infty$*, or that for all* $s \in S$ *and all* $p\colon S \to \mathbb{R}_{\geq 0}$ *the set* $\{\sum_{s' \in S} \mathbf{R}(s, \alpha, s')/\mathbf{u} \cdot p(s') \mid \alpha \in Act(s)\}$ *is compact.*

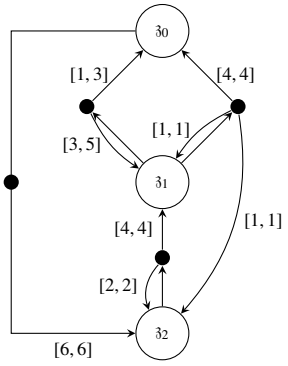
As in a DTMDP, upon entering a state s, an action $\alpha \in Act(s)$ (or a distribution over this set) is chosen to determine the distribution over the successor states. As for CTMCs, the model moves to this successor state after a time given according to the negative exponential distribution with parameter $\mathbf{u}$.



(a) CTMC



(b) CTMDP

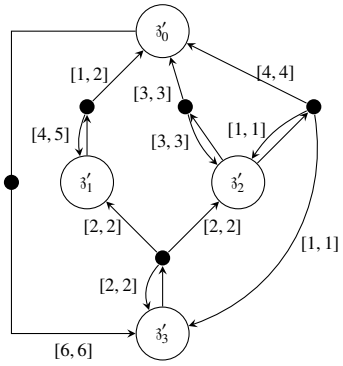


(c) ECTMC

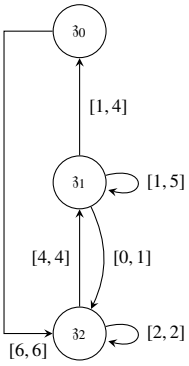
```
ctmc
module example
  n : [0..2] init 0;
  m : [1..4] init 1;
  [] n=0 -> 6 : (n'=2);
  [] n=1 -> m : (n'=0) & (m'=1);
  [] n=1 & (m=4) -> (n'=2) & (m'=1);
  [] n=2 -> 1 : (n'=1) & (m'=1);
  [] n=2 -> 1 : (n'=1) & (m'=2);
  [] n=2 -> 1 : (n'=1) & (m'=3);
  [] n=2 -> 1 : (n'=1) & (m'=4);
endmodule

rewards
  n=0 : 0;
  n=1 : 0.25*m;
  n=2 : 1;
endrewards
```

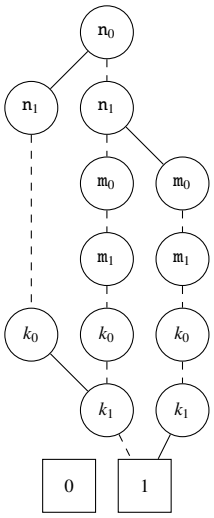
(d) PRISM



(e) ECTMC 2



(f) ACTMC



(g) OBDD abstraction

Figure 1: Example models.

Example 2 *In Fig. 1(b), we give an example for a CTMDP. The nondeterministic choices, available in each state, are represented by the arrows leading to a filled circle. From such a circle, a distribution leads to the successor states. As for the CTMC in Example 1, its uniformisation rate is* 6*.*

We need the following transformation from continuous-time to discrete-time models.

Definition 5 *Given a CTMDP* $C = (S, Act, \mathbf{R})$*, the* embedded DTMDP *is defined as* $\mathrm{emb}(C) \stackrel{\text{def}}{=} (S, Act, \mathbf{P})$ *with* $\mathbf{P}(s, \alpha, s') \stackrel{\text{def}}{=} \mathbf{R}(s, \alpha, s')/\mathbf{u}$ *for all* $s, s \in S$*, and* $\alpha \in Act(s)$*.*

We introduce a formalism to specify CTMDPs, extending the *abstract Markov chains* by Klink et al. [50, 46, 53], which is also a specific form of a *constraint Markov chain* [17]. The purpose of this model is to efficiently represent CTMDPs with a large number of actions. Instead of explicitly enumerating all possible choices over successor distributions, it allows us to specify lower and upper bounds on the rates between states.

Definition 6 *An* extended abstract continuous-time Markov chain (ECTMC) *is a tuple* $\mathcal{C} = (S, \widehat{Act}, \mathbf{I}^\ell, \mathbf{I}^u)$ *where* S *is as in Definition 3, and* $\widehat{Act}$ *is a finite set of actions. We consider the uniformisation rate* $\mathbf{u}(C)$ *of the model. The* intervals *are partial functions of the form* $\mathbf{I}^\ell, \mathbf{I}^u \colon (S \times \widehat{Act}) \rightharpoonup (S \to [0, \mathbf{u}])$*. We require* $\mathbf{I}^\ell$ *and* $\mathbf{I}^u$ *to have the same domain. In addition, for each* $s \in S$*, there must be at least one* $\hat{\alpha} \in \widehat{Act}$ *such that* $\mathbf{I}^\ell(s, \hat{\alpha})$ *is defined.*

The CTMDP semantics *of an ECTMC is defined as* $[\![C]\!] \stackrel{\text{def}}{=} (S, Act, \mathbf{R})$*. We have* $(\hat{\alpha}, \nu) \in Act$ *if* $\hat{\alpha} \in \widehat{Act}$*, and if* ν *is of the form* $\nu \colon S \to [0, \mathbf{u}]$ *with* $\sum_{s \in S} \nu(s) = \mathbf{u}$*. It is* $(\hat{\alpha}, \nu) \in Act(s)$ *if* $\mathbf{I}^\ell(s, \hat{\alpha})$ *and* $\mathbf{I}^u(s, \hat{\alpha})$ *are defined, and* $\nu(s') \in [\mathbf{I}^\ell(s, \hat{\alpha})(s'), \mathbf{I}^u(s, \hat{\alpha})(s')]$ *for all* $s' \in S$*. We let* $\mathbf{R}(s, (\hat{\alpha}, \nu), s') \stackrel{\text{def}}{=} \nu(s')$*.*

An ECTMC thus represents a CTMDP in which, for each state s, one chooses a possible action $\hat{\alpha}$ of $\widehat{Act}$. In addition, one has to choose an assignment of successor rates which fulfill the requirement on the intervals. This way, the action set is uncountably large, but satisfies the requirements of Definition 4. The difference to the model of Klink et al. is the choice of $\hat{\alpha} \in \widehat{Act}$ before the choice of the successor rates. This difference allows us to obtain more precise abstractions than we could obtain if we were using (non-extended) abstract Markov chains, while it still allows us to implement efficient analysis methods, as seen later in Sections 3 and 4.

Example 3 *We give an example for an ECTMC in Fig. 1(c) with uniformisation rate* 6*. Compared to the CTMDP in Fig. 1(b), the ECTMC allows rate intervals instead of rates.*

To obtain a stochastic process from nondeterministic models, the nondeterminism must be resolved. *Schedulers* (or *policies*) formalise the mechanism to do so. Below, we define the most powerful class of schedulers we consider in this paper, and the stochastic processes they induce. A scheduler of this class can resolve the nondeterminism according to the states and actions (and their sequence) that were visited before the model moved to the current state. It may also decide not to pick one specific action, but rather involve a probabilistic choice over the enabled actions of a state. It is however neither aware of the exact time at which former events happened nor of the current time.

Definition 7 *A* time-abstract, history-dependent, randomised scheduler (HR) *for a DTMDP* $\mathcal{D} = (S, Act, \mathbf{P})$ *or a CTMDP* $C = (S, Act, \mathbf{R})$ *is a function* $\sigma \colon ((S \times Act)^* \times S) \to Distr(Act)$ *such that, for all* $\beta \in (S \times Act)^*$*, and* $s \in S$*, we have that if* $\sigma(\beta, s)(\alpha) > 0$ *then* $\alpha \in Act(s)$*. With* Σ_{HR}*, we denote the set of all HRs.*

Definition 8 *Assume we are given a CTMDP* $C = (S, Act, \mathbf{R})$*, and a HR* $\sigma \colon ((S \times Act)^* \times S) \to Distr(Act)$*. We define the* induced CTMC *as* $C_\sigma \stackrel{\text{def}}{=} (S', \mathbf{R}')$ *with*

- $S' \stackrel{\text{def}}{=} (S \times Act)^* \times S$*,*
- $\mathbf{R}'((\beta, s), (\beta, s, \alpha, s')) \stackrel{\text{def}}{=} \sigma(\beta, s)(\alpha) \cdot \mathbf{R}(s, \alpha, s')$ *for* $\beta \in (S \times Act)^*$*,* $s, s' \in S$*,* $\alpha \in Act$*, and* $\mathbf{R}'(\cdot, \cdot) \stackrel{\text{def}}{=} 0$ *otherwise.*

Let $X^{C_\sigma, s_0} \colon (\Omega_{C_\sigma} \times \mathbb{R}_{\geq 0}) \to ((S \times Act)^* \times S)$ *be the stochastic process of the CTMC* C_σ *with initial state* $s_0 \in S$*, and let* $f \colon ((S \times Act)^* \times S) \to S$ *with* $f(\beta, s) \stackrel{\text{def}}{=} s$*. The* induced stochastic process $X^{C, \sigma, s_0} \colon (\Omega_{C_\sigma} \times \mathbb{R}_{\geq 0}) \to S$ *of* C *and* σ *starting in* s_0 *is then defined as* $X_t^{C, \sigma, s_0} \stackrel{\text{def}}{=} f \circ X_t^{C_\sigma, s_0}$ *for* $t \in \mathbb{R}_{\geq 0}$*. Definitions for DTMDPs are likewise using* $\mathbf{P}$ *instead of* $\mathbf{R}$*.*

We specify a simpler subclass of the schedulers of Definition 7. Schedulers of this class are only aware of the number of state changes that have happened so far, and may only choose a specific successor distribution rather than a distribution over them.

Definition 9 *A* time-abstract, history-abstract, counting, deterministic scheduler (CD) *for a DTMDP* $\mathcal{D} = (S, Act, \mathbf{P})$ *or a CTMDP* $C = (S, Act, \mathbf{R})$ *is a function* $\sigma\colon (S \times \mathbb{N}) \to Act$ *such that for all* $s \in S$ *and* $n \in \mathbb{N}$ *if* $\sigma(s,n) = \alpha$ *then* $\alpha \in Act(s)$*. With* Σ_{CD} *we denote the set of all CDs.*

Definition 10 *Assume we are given a CTMDP* $C = (S, Act, \mathbf{R})$*, and a CD* $\sigma\colon (S \times \mathbb{N}) \to Act$*. We define the* induced CTMC *as* $C_\sigma \stackrel{\text{def}}{=} (S', \mathbf{R}')$ *with*

- $S' \stackrel{\text{def}}{=} S \times \mathbb{N}$*,*
- $\mathbf{R}'((s,n),(s',n+1)) \stackrel{\text{def}}{=} \mathbf{R}(s, \sigma(s,n), s')$ *for* $s, s' \in S$ *and* $n \in \mathbb{N}$*, and* $\mathbf{R}'(\cdot,\cdot) \stackrel{\text{def}}{=} 0$ *otherwise.*

Let $X^{C_\sigma, s_0}\colon (\Omega_{C_\sigma} \times \mathbb{R}_{\geq 0}) \to (S \times \mathbb{N})$ *be the stochastic process of the CTMC* C_σ*, and let* $f\colon (S \times \mathbb{N}) \to S$ *with* $f(s,n) \stackrel{\text{def}}{=} s$*. The* induced stochastic process $X^{C,\sigma,s_0}\colon (\Omega_{C_\sigma} \times \mathbb{R}_{\geq 0}) \to S$ *of* C *and* σ *starting in* s_0 *is then defined as* $X^{C,\sigma,s_0}_t \stackrel{\text{def}}{=} f \circ X^{C_\sigma,(s_0,0)}_t$ *for* $t \in \mathbb{R}_{\geq 0}$*. Definitions for DTMDPs are likewise using* $\mathbf{P}$ *instead of* $\mathbf{R}$*.*

We equip our models with *reward structures*, assigning values to states.

Definition 11 *A* reward structure *for a stochastic process* $X\colon (\Omega \times \mathbb{R}_{\geq 0}) \to S$ *or a CTMC or CTMDP with state set* S *is a tuple* $(\mathbf{r}_c, \mathbf{r}_f)$ *with* $\mathbf{r}_c\colon S \to \mathbb{R}_{\geq 0}$ *and* $\mathbf{r}_f\colon S \to \mathbb{R}_{\geq 0}$*. We call* $\mathbf{r}_c$ *the* cumulative reward rate*, and* $\mathbf{r}_f$ *the* final reward value*. We let* $\mathbf{r}_f^{\max} \stackrel{\text{def}}{=} \max_{s \in S} \mathbf{r}_f(s)$*, and* $\mathbf{r}_c^{\max} \stackrel{\text{def}}{=} \max_{s \in S} \mathbf{r}_c(s)$*.*

For CTMCs, the cumulative reward rate $\mathbf{r}_c(s)$ is the reward obtained per time unit for staying in state s, until s is left or a given time bound $\mathbf{t}$ is reached. The final reward value $\mathbf{r}_f(s)$ specifies the reward one obtains for being in state s at this time bound $\mathbf{t}$. We are interested in the expected values of these numbers, as formalised in Definition 12. For CTMDPs, we strive for the maximal (and analogously the minimal) value under all possible schedulers in the class we considered.

Definition 12 *Given a time bound* $\mathbf{t} \in \mathbb{R}_{\geq 0}$*, the* value *of a stochastic process* $X\colon (\Omega \times \mathbb{R}_{\geq 0}) \to S$ *with a reward structure* $\mathbf{r} = (\mathbf{r}_c, \mathbf{r}_f)$ *is defined as* $\mathbf{V}(X, \mathbf{r}, \mathbf{t}) \stackrel{\text{def}}{=} \mathbf{E}[\int_0^{\mathbf{t}} \mathbf{r}_c(X_u)\,\mathrm{d}u + \mathbf{r}_f(X_{\mathbf{t}})]$*. For a CTMC* $C = (S, \mathbf{R})$ *and* $s_0 \in S$*, we let* $\mathbf{V}(C, s_0, \mathbf{r}, \mathbf{t}) \stackrel{\text{def}}{=} \mathbf{V}(X^{C,s_0}, \mathbf{r}, \mathbf{t})$*. For a CTMDP* $C = (S, Act, \mathbf{R})$*, the* maximal value *(*minimal value*) for* $s_0 \in S$ *is defined as* $\mathbf{V}^{\max}(C, s_0, \mathbf{r}, \mathbf{t}) \stackrel{\text{def}}{=} \max_{\sigma \in \Sigma_{HR}} \mathbf{V}(X^{C,\sigma,s_0}, \mathbf{r}, \mathbf{t})$ *(*$\mathbf{V}^{\min}(C, s_0, \mathbf{r}, \mathbf{t}) \stackrel{\text{def}}{=} \min_{\sigma \in \Sigma_{HR}} \mathbf{V}(X^{C,\sigma,s_0}, \mathbf{r}, \mathbf{t})$*).*

The interpretation of rewards and values depends on the model under consideration. For instance, in a CTMC representing a chemical reaction, we might assign a final reward value of n to state s if s contains n molecules of a given species. This way, the value of the CTMC represents the expected number of this species at a given point of time.

We do not explicitly consider impulse (instantaneous) rewards $\mathbf{r}_i\colon (S \times S) \to \mathbb{R}_{\geq 0}$ for CTMCs here, that is rewards obtained for moving from one state to another. However, given cumulative reward rates $\mathbf{r}_c$ and impulse rewards $\mathbf{r}_i$, we can define cumulative reward rates $\mathbf{r}'_c$ as $\mathbf{r}'_c(s) \stackrel{\text{def}}{=} \mathbf{r}_c(s) + \sum_{s' \in S} \mathbf{R}(s,s') \cdot \mathbf{r}_i(s,s')$. For the properties under consideration, this new reward structure is equivalent to the one which uses impulse rewards (follows from [54, (6)]). Definition 12 resembles the approach considered in a recent paper [15], where we maximised (minimised) the value over a more general class of schedulers than the one of Definition 7.

An important specific value of a stochastic process is the *time-bounded reachability probability*. The computation of this value is, for instance, necessary to model check the time-bounded until property of the probabilistic logic CSL (continuous stochastic logic) [3].

Given a set of *target states* $\mathbf{B}$, we can express the probability to be in $\mathbf{B}$ *at* time t by using a reward structure with $\mathbf{r}_c(s) = 0$ for all $s \in S$, and $\mathbf{r}_f(s) = 1$ if $s \in \mathbf{B}$ and $\mathbf{r}_f(s) = 0$ otherwise. For the probability to reach $\mathbf{B}$ *within* time t, we additionally modify the rate matrix $\mathbf{R}$ such that $\mathbf{R}(s,s) = \mathbf{u}$, and $\mathbf{R}(s,s') = 0$ for $s' \neq s$ if $s \in \mathbf{B}$. For CTMCs, the case to reach $\mathbf{B}$ within an interval $[a,b]$ with $0 < a < b < \infty$ can be handled by two successive analyses [3, Theorem 3]. Unbounded intervals $[0,\infty)$ and $[a,\infty)$ can be handled similarly [57, Section 4.4].

The fact that Definition 12 involves both final and cumulative rewards for CTMCs allows us to express the values

at different points of time in the following way. Assume we want to consider the cumulative reward rates $v_1, v_2, \ldots$ at consecutive time points $\mathbf{t}_1 = \delta_1, \mathbf{t}_2 = \mathbf{t}_1 + \delta_2, \ldots$ A short calculation then shows that $v_1 = \mathbf{V}(C, s, (\mathbf{r}_c, 0), \delta_1)$, $v_2 = \mathbf{V}(C, s, (\mathbf{r}_c, v_1), \delta_2), \ldots$ The formulation for the final reward at different points of time is likewise.

Example 4 *Reconsider the CTMC from Example 1, the CTMDP from Example 2, and the ECTMC from Example 3 all sketched in Fig. 1(a) through (c). Assume that for the CTMC we use a reward structure* $\mathbf{r} \stackrel{\text{def}}{=} (\mathbf{r}_c, \mathbf{r}_f)$ *with* $\mathbf{r}_c(s_0) \stackrel{\text{def}}{=} 0.0$, $\mathbf{r}_c(s_1) \stackrel{\text{def}}{=} 0.25$, $\mathbf{r}_c(s_2) \stackrel{\text{def}}{=} 0.5$, $\mathbf{r}_c(s_3) \stackrel{\text{def}}{=} 0.75$, $\mathbf{r}_c(s_4) \stackrel{\text{def}}{=} 1$, $\mathbf{r}_c(s_5) \stackrel{\text{def}}{=} 1$, *and* $\mathbf{r}_f(\cdot) \stackrel{\text{def}}{=} 0$. *Then we have that the reward value for* s_0 *for a time bound of* $\mathbf{t} \stackrel{\text{def}}{=} 5$ *is* $\mathbf{V}(C, s_0, \mathbf{r}, \mathbf{t}) \approx 2.70116$.

Assume that, for the CTMDP and the ECTMC, we have $\mathbf{r}_c(\mathfrak{z}_0) \stackrel{\text{def}}{=} 0$, $\mathbf{r}_c(\mathfrak{z}_1) \stackrel{\text{def}}{=} 1$, $\mathbf{r}_c(\mathfrak{z}_2) \stackrel{\text{def}}{=} 1$, *and* $\mathbf{r}_f(\cdot) \stackrel{\text{def}}{=} 0$*; and that we have* $\mathbf{t} \stackrel{\text{def}}{=} 5$. *Then we have* $\mathbf{V}^{\max}(C, \mathfrak{z}_0, \mathbf{r}, \mathbf{t}) \approx 4.30277$ *for both models. Now assume that the reward values are* $\mathbf{r}_c(\mathfrak{z}_0) \stackrel{\text{def}}{=} 0$, $\mathbf{r}_c(\mathfrak{z}_1) \stackrel{\text{def}}{=} 0.25$, $\mathbf{r}_c(\mathfrak{z}_2) \stackrel{\text{def}}{=} 1$, *and* $\mathbf{r}_f(\cdot) \stackrel{\text{def}}{=} 0$ *instead. Then we have* $\mathbf{V}^{\min}(C, \mathfrak{z}_0, \mathbf{r}, \mathbf{t}) \approx 1.82699$.

2.2 PRISM's guarded command language

Prism [55] is a widely used tool, which features a guarded command language to model CTMCs (among other classes). For our purposes, it suffices to take a rather abstract view on the high-level modelling language used by this tool.

Definition 13 *A* Prism model (PM) *is a tuple* $m = (Var, init, C, succ, R_c, R_f)$. *Here, Var is a set of Boolean* variables. *With* $init\colon Var \to \{0,1\}$, *we denote the* initial state, *and C is a finite set of* commands. *Let* $S_{Var} \stackrel{\text{def}}{=} \{s\colon Var \to \{0,1\}\}$ *be the set of variable assignments for Var. The* cumulative reward rate *is a function* $R_c\colon S_{Var} \to \mathbb{R}_{\geq 0}$ *as is the* final reward value $R_f\colon S_{Var} \to \mathbb{R}_{\geq 0}$. *The* successor function *is a partial function of the form* $succ\colon (S_{Var} \times C) \rightharpoonup (S_{Var} \times \mathbb{R}_{>0})$. *We define* $\overline{succ}(s,c) \stackrel{\text{def}}{=} s'$ *if* $succ(s,c) = (s', \lambda)$ *for some* $\lambda \in \mathbb{R}_{>0}$. *We also let*

$$succ(s) \stackrel{\text{def}}{=} \left\{ (s', \lambda) \,\middle|\, \exists c.\, \overline{succ}(s,c) = s' \wedge \lambda = \sum_{\lambda'; \exists c'.\,(s',\lambda') = succ(s,c')} \lambda' \right\}.$$

Further, $\overline{succ}(s) \stackrel{\text{def}}{=} \{s' \mid \exists \lambda.\, (s', \lambda) \in succ(s)\}$. *Let* $\overline{succ}^0 \stackrel{\text{def}}{=} init$, *and* $\overline{succ}^{i+1} \stackrel{\text{def}}{=} \{\overline{succ}(s) \mid s \in \overline{succ}^i\}$. *The set of* reachable states (state space) *is* $S_m \stackrel{\text{def}}{=} \bigcup_{i=0}^{\infty} \overline{succ}^i$. *We require that there is* $\mathbf{u}(m) > 0$ *such that* $\mathbf{u}(m) = \sum\{\lambda \mid \exists s'.\, (s', \lambda) \in succ(s)\}$ *for all* $s \in S_m$ *(where the latter is a multiset).*

The complete Prism syntax also defines models consisting of several *modules*, that is, sets of guarded commands, which may synchronise or interleave. However, as the semantics of a model with several modules is defined as one with a single module, a single set of commands suffices. Prism also allows us to specify commands with several pairs of successors $(s'_1, \lambda_1), \ldots, (s'_n, \lambda_n)$. For PMs describing CTMCs, such a command is equivalent to a set of n commands c_i in the above form. Each of them must be activated (defined) in the same states as the original command, and we then have $succ(c_i) = (s'_i, \lambda_i)$. The bounded integers Prism supports can be represented by a binary encoding. Impulse rewards can be transformed to cumulative rewards, as discussed for CTMCs. For models in which there is no $\mathbf{u}(m)$ with the required property, we can add an extra command to increase the self-loop rate where necessary.

The formal semantics of a PM is as follows.

Definition 14 *Consider a PM* $m = (Var, init, C, succ, R_c, R_f)$. *The* induced CTMC *is* $C_m \stackrel{\text{def}}{=} (S_m, \mathbf{R})$ *such that for all* $s, s' \in S_m$ *we have* $\mathbf{R}(s,s') \stackrel{\text{def}}{=} \lambda$ *if* $(s', \lambda) \in succ(s)$, *and* $\mathbf{R}(s,s') \stackrel{\text{def}}{=} 0$ *if no such tuple exists. The* induced reward structure *is* $\mathbf{r}_m \stackrel{\text{def}}{=} (R_c, R_f)$.

In Section 3, we will abstract CTMCs into ECTMCs. To do so, we will subsume several concrete states of a CTMC to abstract states of an ECTMC.

Definition 15 *Given a PM m, a* partitioning *of the state space* S_m *is a finite ordered set* $\mathfrak{P} = \langle \mathfrak{z}_0, \ldots, \mathfrak{z}_{n-1} \rangle$ *of nonempty, pairwise disjoint subsets of* S_m *such that* $S_m = \bigcup_{i=0}^{n-1} \mathfrak{z}_i$.

Example 5 *In Fig. 1(d) we give an example of a* Prism *model. The description is given in the textual form which is used by the tool itself. The first line states that the model is a CTMC. Then, a single module with the name* `example` *is declared. In this module, there are two variables* `n`*, and* `m` *with a specified variable range, and initial value. Afterwards, the successor function is given in terms of*

guarded commands of the form `<guard> -> <rate> : <successor>`.

The induced CTMC is the model in Fig. 1(a) (for readability, we have left out the commands that lead to the self-loops in this model). Here, s_0 *corresponds to* $\mathsf{n} = 0$, $\mathtt{m} = 1$, s_5 *corresponds to* $\mathsf{n} = 2$, $\mathtt{m} = 1$, *and the other* s_i *correspond to* $\mathsf{n} = 1$, $\mathtt{m} = i$.

The reward structure is given below the module definition. PRISM *only supports either final or cumulative rewards, but not both at the same time. Thus, either the final or the cumulative reward part is zero. Whether the reward specified this way shall denote the final or cumulative reward is decided by using a formula specification.*

As an example partitioning, we can consider $\mathfrak{P} = \langle \mathfrak{Z}_0, \mathfrak{Z}_1, \mathfrak{Z}_2 \rangle$ *with* $\mathfrak{Z}_0 = \{s_0\}$, $\mathfrak{Z}_1 = \{s_1, s_2, s_3, s_4\}$, *and* $\mathfrak{Z}_2 = \{s_5\}$.

2.3 Binary decision diagrams

Binary decision diagrams [11] are an efficient tool to symbolically represent structures which are too large to be represented in an explicit form.

Definition 16 *We fix a finite ordered set* $\mathsf{V} \stackrel{\text{def}}{=} \langle \mathsf{x}_1, \ldots, \mathsf{x}_m \rangle$ *of* Boolean variables. *A* binary decision diagram *(BDD) is a rooted acyclic directed graph* b *with node set* N, *and root node* n_{root}. *There are two types of nodes in* N*: terminal nodes, and non-terminal nodes.* Terminal *nodes* n *do not have out-going edges, and are labelled with a value* $\mathsf{v}(bddNode) \in \{0, 1\}$. *The remaining nodes are* non-terminal *nodes* $\mathsf{n} \in \mathsf{N}$, *which have exactly two successor nodes, denoted by* $\mathsf{h}(\mathsf{n})$ *(*high *successor), and* $\mathsf{l}(\mathsf{n})$ *(*low *successor). Non-terminal nodes* n *are labelled with a variable* $\mathsf{v}(\mathsf{n}) \in \mathsf{V}$.

A *variable valuation* is a function $v \colon \mathsf{V} \to \{0, 1\}$. We denote the set of all variable valuations by Val. Each valuation v induces a unique path in the BDD from the root node to a terminal node. At a non-terminal node n, we follow the edge to $\mathsf{h}(\mathsf{n})$ if $v(\mathsf{v}(\mathsf{n})) = 1$, and the edge to $\mathsf{l}(n)$ if $v(\mathsf{v}(\mathsf{n})) = 0$. The function $[\![\mathsf{b}]\!] : \mathsf{Val} \to \{0, 1\}$ represented by a BDD b returns for a variable valuation v the value of the terminal node reached by following the path induced by v.

Definition 17 *A BDD is* ordered *if for all non-terminal nodes n the following condition holds. Either* $\mathsf{h}(\mathsf{n})$ *is a terminal node, or* $\mathsf{v}(\mathsf{n}) < \mathsf{v}(\mathsf{h}(\mathsf{n}))$, *and the same for* $\mathsf{l}(\mathsf{n})$. *A BDD is* reduced *if all sub-BDDs rooted at the different nodes of the BDD represent distinct functions. Reduced and ordered BDDs are called* OBDDs.

The OBDD for the constant 0 (or 1) function, which consists of a single terminal node labelled with a 0 (or a 1), is denoted in the sequel by bdd_0 (or bdd_1, respectively).

OBDDs are a canonical representation (up to isomorphism) of arbitrary functions $f \colon \mathsf{Val} \to \{0, 1\}$ [11]. In the following, we will only use OBDDs. For more details on (O)BDDs, we refer the reader to [11, 80].

OBDDs support a wide number of operations like the Boolean operations $\wedge$, $\vee$, and $\neg$. Given two ordered sets $\mathsf{V}_1 = \langle \mathsf{x}_{i_1}, \ldots, \mathsf{x}_{i_n} \rangle$ and $\mathsf{V}_2 = \langle \mathsf{x}_{j_1}, \ldots, \mathsf{x}_{j_n} \rangle$ of Boolean variables, by $\mathsf{b}' = \mathsf{b}[\mathsf{V}_1/\mathsf{V}_2]$ we denote the OBDD which results from renaming the variables in V_1 to the corresponding variables in x_2. For $\mathsf{V}' \subseteq \mathsf{V}$ and OBDD b, we let $[\![\exists \mathsf{V}'.\mathsf{b}]\!](v) = \bigvee \{ [\![\mathsf{b}]\!](v') \,\big|\, \forall \mathsf{x} \notin \mathsf{V}'.v'(x) = v(x) \}$ be the existential quantification of the variables in V'.

We can use OBDDs to represent PMs in a symbolic form, if we leave out the stochastic aspects.

Definition 18 *Consider a PM* $m = (Var, init, C, succ, R_c, R_f)$. *The* OBDD representation *of m is a tuple* $\mathsf{b}_m \stackrel{\text{def}}{=} (Var, Var', \mathsf{init}, \{\mathsf{succ}_c\}_{c \in C})$. *There, Var and Var′ are sets of Boolean variables with* $Var \cap Var' = \emptyset$ *such that there is a one-to-one mapping between variables* $x \in Var$ *and* $x' \in Var'$. *Further,* init, *and* succ_c *are OBDDs over the variables Var, and* $Var \cup Var'$, *respectively. We require that* $[\![\mathsf{init}]\!](v) = 1$ *iff* $init(x) = v(x)$ *holds for all* $x \in Var$. *For all* succ_c, *we require* $[\![\mathsf{succ}_c]\!](v) = 1$ *iff for all* $x \in Var$ *it is true that* $v(x) = s(x)$, $v(x') = s'(x)$, *and* $\overline{succ}(s, c) = s'$. *By* succ, *we denote the OBDD such that* $[\![\mathsf{succ}]\!] = \bigvee_{c \in C} [\![\mathsf{succ}_c]\!]$.

OBDDs can also be used to symbolically represent a partitioning of the state space of a PM. Let $\mathfrak{P} = \langle \mathfrak{Z}_0, \ldots, \mathfrak{Z}_{n-1} \rangle$ be a partitioning of the PM $m = (Var, init, C, succ, R_c, R_f)$. The idea is to assign to each block $\mathfrak{Z}_i$ of $\mathfrak{P}$ a unique block number i, and to use a binary representation of i, which is encoded using $k = \lceil \log_2 n \rceil$ novel BDD variables $\mathfrak{V} = \langle \mathfrak{x}_0, \ldots, \mathfrak{x}_{k-1} \rangle$.

Definition 19 *The* OBDD representation *of* $\mathfrak{P} = \langle \mathfrak{Z}_0, \ldots, \mathfrak{Z}_{n-1} \rangle$ *is the OBDD* $\mathsf{b}_{\mathfrak{P}}$ *over the variables* $Var \uplus \mathfrak{V}$, *where* $\mathfrak{V} = \langle \mathfrak{x}_0, \ldots, \mathfrak{x}_{k-1} \rangle$ *with* $k = \lceil \log_2 n \rceil$. *We require that* $[\![\mathsf{b}_{\mathfrak{P}}]\!](v) = 1$ *iff there is* $s \in S_m$ *such that, for all*

$x \in Var$, we have $v(x) = s(x)$; and there is $\mathfrak{z} \in \mathfrak{P}$ such that $s \in \mathfrak{z}$, and for all $\mathfrak{x} \in \mathfrak{V}$ we have $v(\mathfrak{x}) = \mathfrak{z}(\mathfrak{x})$. With $\mathfrak{z}_i(\mathfrak{x}_j) \stackrel{\text{def}}{=} (i \operatorname{div} 2^j) \mod 2$, we denote the value of variable $\mathfrak{x}_j \in \mathfrak{V}$ in the binary encoding of the block number i of $\mathfrak{z}_i \in \mathfrak{P}$. With $\mathsf{b}_\mathfrak{z}$, we denote the OBDD such that $[\![\mathsf{b}_\mathfrak{z}]\!](v) = 1$ iff v represents a state of $\mathfrak{z}$.

There are several alternative OBDD-based partition representations available. One possibility is to use an MTBDD with the block numbers in the leaves. However, the algorithms of the tool Sigref [83], which we use for refining partitions, require us to represent sets of block numbers within the BDD, which is easily possible using our encoding. To do so, the authors of [29] encode such sets as products of prime numbers. However, in our application this would yield numbers which are too large to fit into the standard 32- or 64-bit data types. Therefore this approach is not directly possible, and requires technical tricks which might affect the efficiency of the method. Another possibility is to use one OBDD for each abstract state. In this representation, finding the abstract state containing a given concrete state is more expensive (in $O(|Var| \cdot |\mathfrak{P}|)$ instead of $O(|Var| + |\mathfrak{P}|)$). The logarithmic partition encoding proposed by Derisavi [28] has certain advantages regarding memory consumption, but partition refinement is much more expensive regarding computation time than the representation we use [82]. Finally, Bouali and de Simone [9] represent the corresponding equivalence relation for bisimulation computation. In previous work, [83], we have observed that this representation is often larger than the one used by Sigref, slowing down the computations.

Regarding the variable order of $Var \uplus \mathfrak{V}$, we assume in the following that all variables in *Var* precede all variables in $\mathfrak{V}$. This ordering leads to more efficient algorithms for accessing the block number of a given state.

Example 6 *Reconsider the* Prism *model from Example 5. We can encode each of the variables* n *and* m *by two binary variables n_0, n_1 and m_0, m_1 by using the binary representations $n = 0 \leftrightarrow (n_1, n_0) = (0,0)$, $n = 1 \leftrightarrow (n_1, n_0) = (0,1)$, $n = 2 \leftrightarrow (n_1, n_0) = (1,0)$, $m = 1 \leftrightarrow (m_1, m_0) = (0,0)$, $m = 2 \leftrightarrow (m_1, m_0) = (0,1)$, $m = 3 \leftrightarrow (m_1, m_0) = (1,0)$, $m = 4 \leftrightarrow (m_1, m_0) = (1,1)$. To encode the partitioning of the previous example ($\mathfrak{z}_0 = \{s_0\}$, $\mathfrak{z}_1 = \{s_1, s_2, s_3, s_4\}$, and $\mathfrak{z}_2 = \{s_5\}$), we enumerate $\mathfrak{z}_i$ by introducing new binary variables k_0, k_1. We sketch the encoding in Fig. 1(g). The OBDD terminal nodes are given as squares at the bottom of the diagram. Non-terminal nodes are given as circles, labelled with their variables. The* h *successors are connected by solid lines, whereas* l *successors are connected by dashed ones. For readability, we leave out the connections to the* 0 *terminal node.*

3 Algorithms

In this section, we first describe an algorithm to approximate minimal and maximal values of CTMDPs. Afterwards, we describe how to obtain an ECTMC from a PM, such that its induced CTMDP is a valid abstraction (cf. Proposition 2) of the CTMC semantics of the PM. We provide an algorithm which computes an ECTMC overapproximation of a PM given in an OBDD representation. Using the first algorithm, we can obtain intervals from this abstraction which are guaranteed to bound the actual value (cf. Definition 12) of the CTMC from above and below.

3.1 Computing Reward Values for CTMDPs

Let $\phi_\lambda(i) \stackrel{\text{def}}{=} \lambda^i e^{-\lambda}/(i!)$ denote the probabilities of a Poisson distribution with parameter λ, and let $\psi_\lambda(i) \stackrel{\text{def}}{=} \sum_{j=i+1}^{\infty} \phi_\lambda(j) = 1 - \sum_{j=0}^{i} \phi_\lambda(j)$.

The algorithm to compute the maximal values of CTMDPs is given in Algorithm 1. The input is a CTMDP C with reward structure $\mathbf{r} = (\mathbf{r}_c, \mathbf{r}_f)$, and the precision $\varepsilon > 0$ up to which the values are to be computed. The algorithm for the minimum is likewise, replacing max by min in Line 7.

The choice of k in line 1 of Algorithm 1 is based on the following lemma, which is proven in the Appendix.

Lemma 1 *Given a CTMDP $C = (S, Act, \mathbf{P})$ with a reward structure $\mathbf{r} = (\mathbf{r}_c, \mathbf{r}_f)$, a precision $\varepsilon > 0$, and k such that*

$$\sum_{n=0}^{k} \psi_{\mathbf{ut}}(n) > \mathbf{ut} - \frac{\varepsilon \mathbf{u}}{2\mathbf{r}_c^{\max}} \text{ and } \psi_{\mathbf{ut}}(k) \cdot \mathbf{r}_f^{\max} < \frac{\varepsilon}{2},$$

Algorithm 1: Compute maximal values for $C = (S, Act, \mathbf{R})$, $\mathbf{r} = (\mathbf{r}_c, \mathbf{r}_f)$ up to ε.

1 let k s. t. $\sum_{i=0}^{k} \psi_{\mathbf{ut}}(i) > \mathbf{ut} - \varepsilon\mathbf{u}/(2\mathbf{r}_c^{\max}) \wedge \psi_{\mathbf{ut}}(k) \cdot \mathbf{r}_f^{\max} < \varepsilon/2$
2 $C' = (S, Act, \mathbf{P}) := \mathrm{emb}(C)$
3 **forall** $s \in S$ **do** $q_{k+1}(s) := 0$
4
5 **forall** $i = k, k-1, \ldots, 0$ **do**
6 **forall** $s \in S$ **do**
7 $m := \max_{\alpha \in Act(s)} \sum_{s' \in S} \mathbf{P}(s, \alpha, s') q_{i+1}(s')$
8 $q_i(s) := m + \phi_{\mathbf{ut}}(i) \cdot \mathbf{r}_f(s) + \psi_{\mathbf{ut}}(i) \cdot \mathbf{r}_c(s)/\mathbf{u}$
9 **return** q_0

then for all schedulers $\sigma \in \Sigma_{CR}$, and all $s_0 \in S$, we have

$$\sum_{i=k+1}^{\infty} \left(\phi_{\mathbf{ut}}(i) \sum_{s \in S} \pi^{\mathrm{emb}(C),\sigma}(s_0, i, s) \mathbf{r}_f(s) + \psi_{\mathbf{ut}}(i) \sum_{s \in S} \pi^{\mathrm{emb}(C),\sigma}(s_0, i, s) \frac{\mathbf{r}_c(s)}{\mathbf{u}} \right) < \varepsilon.$$

Here, $\pi^{\mathrm{emb}(C),\sigma}(s_0, i, s)$ is the probability of being in state s of emb(C) *in step i if having started in s_0 when using scheduler σ. By Σ_{CR}, we denote the set of schedulers which extend Σ_{CD} by randomised choice over the actions.*

The requirement on the actions in Definition 4 assures that the maximum in Algorithm 1 (line 7) exists. We can also directly apply this algorithm on ECTMCs without constructing the uncountably large induced CTMDPs. The crucial part here is the optimisation over the uncountable actions, which can be done using a slight adaptation of methods from [50, Chapter 4.1]. There, optimising the assignment of successor rates with restrictions given by lower and upper bounds is already described. For each $s \in S$, and for each $\hat{\alpha}$ such that $\mathbf{I}^{\ell}(s, \hat{\alpha})$ (and thus $\mathbf{I}^{u}$) is defined, we can apply the method described in [50, Chapter 4.1], thus to find the optimal $v_{\hat{\alpha}} \colon S \to [0, \mathbf{u}]$ for this $\hat{\alpha}$. Afterwards, we choose the optimal $v_{\hat{\alpha}} \colon S \to [0, \mathbf{u}]$ among all $\hat{\alpha}$, which is easy as there are only a finite number.

Proposition 1 *Let $C = (S, Act, \mathbf{R})$ be a CTMDP with reward structure $\mathbf{r} = (\mathbf{r}_c, \mathbf{r}_f)$. Then, there exists $\sigma \in \Sigma_{CD}$ such that $\mathbf{V}^{\max}(C, s_0, \mathbf{r}, \mathbf{t}) = \mathbf{V}(X^{C,\sigma,s_0}, \mathbf{r}, \mathbf{t})$ for all $s_0 \in S$. Further, the return value q_0 of Algorithm 1 is such that $|\mathbf{V}^{\max}(C, s_0, \mathbf{r}, \mathbf{t}) - q_0(s_0)| < \varepsilon$.*

Proof sketch: At first, we show that we can simulate each history-dependent randomised scheduler by a randomised counting scheduler (CR). In contrast to CD, these schedulers may be randomised. However, as for CDs, decisions of these schedules only depend on the number of steps which have passed rather than on the full history. The fact that CR can simulate HR schedulers allows us to use a result about discrete-time Markov chains. Next, we show that Algorithm 1 cannot yield values which are larger than the maximal value resulting from such a CR. Then, we show that the algorithm does not return values which are larger than the value obtained by any CR plus the specified precision.

Looking at the decisions the algorithm takes at Line 7, we can reconstruct a prefix of the decisions of a CD. By letting the precision approach 0, we can show that there is indeed a complete CD yielding the same value. □ The full proof can be found in the Appendix.

Algorithm 1 generalises an approach from a previous paper about time-bounded reachability [4] using results by Kwiatkowska et al. [54]. Its correctness also proves that deterministic counting schedulers suffice to obtain optimal values, because the algorithm implicitly computes such a scheduler.

It is also related to an earlier work in queueing theory [61], which is however different in a number of ways. The target there was to obtain approximations for a more general class of schedulers of CTMDPs than we need here, and thus does not consider maxima over HRs explicitly. It assumes a fixed maximal number of steps to happen in the uniformised DTMDP, rather than deriving the necessary number, as we do in our algorithm. [61] is also more involved with models featuring a particular structure rather than computing conservative bounds on properties of CTMCs.

The next proposition states how CTMDPs can be used to over-approximate CTMCs.

Proposition 2 *Let $C = (S, \mathbf{R})$ be a CTMC with reward structure $(\mathbf{r}_c, \mathbf{r}_f)$, and let $\mathfrak{P} = \langle \mathfrak{z}_0, \ldots, \mathfrak{z}_{n-1} \rangle$ be a partitioning of S. Consider the CTMDP $C' \stackrel{\text{def}}{=} (\mathfrak{P}, Act, \mathbf{R}')$ where for each $\mathfrak{z} \in \mathfrak{P}$ and $s \in \mathfrak{z}$ we find $\alpha_s \in Act$ such that for all $\mathfrak{z}' \in \mathfrak{P}$ we have $\mathbf{R}'(\mathfrak{z}, \alpha_s, \mathfrak{z}') \stackrel{\text{def}}{=} \sum_{s' \in \mathfrak{z}'} \mathbf{R}(s, s')$.*

Further, consider a reward structure $(\mathbf{r}'_c, \mathbf{r}'_f)$ *such that for all* $\mathfrak{z} \in \mathfrak{P}$ *it is true that* $\mathbf{r}'_c(\mathfrak{z}) \geq \max_{s \in \mathfrak{z}} \mathbf{r}_c(s)$*, and* $\mathbf{r}'_f(\mathfrak{z}) \geq \max_{s \in \mathfrak{z}} \mathbf{r}_f(s)$*. Then, for all* $\mathfrak{z}_0 \in \mathfrak{P}$ *and* $s_0 \in \mathfrak{z}_0$*, we have* $\mathbf{V}(C, s_0, \mathbf{r}, \mathbf{t}) \leq \mathbf{V}^{\max}(C', \mathfrak{z}_0, \mathbf{r}', \mathbf{t})$*.*

Proof sketch: We can construct a scheduler σ such that the embedded DTMDP of C' mimics the behaviour of the embedded DTMC of C, so that in each step the probability to be in a given abstract state $\mathfrak{z}$ is the sum of the probabilities of being in a state s of C with $s \in \mathfrak{z}$. By the definition of reward structures, the value obtained in C' using σ is as least as high as the value in C. As the maximal value in C' is at least as high as the one using σ, the result follows. □ The full proof can be found in the Appendix.

We remark that Algorithm 1 can only compute maximal bounds up to ε. Thus, when applying it to compute upper bounds on the values of CTMCs, one has to add ε to get a number which is guaranteed to bound the value of the original model from above. Alternatively, one could apply bounding semantics [88].

As mentioned before, CSL bounded-until properties can be expressed using rewards. Their probabilities can thus also be bounded using Algorithm 1.

We remark that, for the case of intervals $[a, b]$ or $[a, \infty)$, the successive application of the algorithm only bounds the value in the CTMC that has been abstracted. Assume that one divides the interval into several parts, and successively applies the algorithm for each of these parts by using the result of the previous application as the instantaneous reward of the next application. Doing so allows us to obtain values at different points of time. However, the last minimal (or maximal, respectively) value is not guaranteed to be identical to the value which one would have obtained by a single analysis of the whole interval.

Example 7 *Consider the CTMDP* C *in Fig. 2, which is adapted from previous publications [87, 15]. The only difference is that non-uniform CTMDPs were used previously, i. e., CTMDPs in which the sum of leaving rates may be different for each state. Here, we have increased the self-loop rates thus to obtain a uniformisation rate of* $\mathbf{u} = 10$*. States of the model are given as circles, in which the state name is written. If there is more than one activated action in a state, we draw all of them as small black circles connected to the corresponding states. Non-zero transition rates are then drawn starting in the corresponding black circle. In case there is just one possible action in a state, we directly draw the transitions from this state. Rates with value* 0 *are left out.*

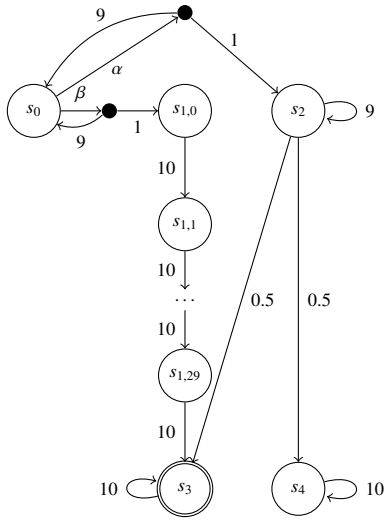


Figure 2: Example demonstrating that successive applications of Algorithm 1 to sub-intervals are not equivalent to a single application on the whole interval.

We assign a cumulative reward rate of 0 *to each state. The final reward is* 0 *for all states except for* s_3*, where it is* 1*. Intuitively, if we want to maximise the value, the optimal choice of the action depends on the amount of time left. If much time is left, it is best to choose* β *in* s_0*, because this action leads to a sequence of states which, given an infinite amount of time, always reaches* s_3*. If little time is left, it is better to choose* α*, because then* s_3 *can be reached quickly, although there is a significant chance that this state will not be reached at all.*

With this model and reward structure, we can compute the probabilities of the CSL formula $\mathcal{P}\left(true\ \mathcal{U}^{[0,4]}\ s_3\right)$ *by computing the value for* $\mathbf{t} = 4$*. Because the state* s_3 *is absorbing, this value is also the probability of the interval-bounded until property* $\mathcal{P}\left(true\ \mathcal{U}^{[1,4]}\ s_3\right)$*.*

In this special case, we can thus compute this probability using the method developed in this paper, or by the previous algorithm by Baier et al. [4]. For s_0*, this value is* $\mathbf{V}^{\max}(C, s_0, (0, \mathbf{r}_f), 4) \approx 0.659593$*.*

Now we apply two consecutive analyses to bound the

interval-bounded reachability probability. For this procedure, we first compute $v(\cdot) \stackrel{\text{def}}{=} \mathbf{V}^{\max}(C, s_0, (0, \mathbf{r}_f), \cdot, 3)$*, and afterwards we consider* $v'(\cdot) \stackrel{\text{def}}{=} \mathbf{V}^{\max}(C, s_0, (0, v), \cdot, 1)$*. We now have* $v'(s_0) \approx 0.671162$*, which shows that this value is an upper bound for the reachability probability in the original model. It is indeed between the value obtained using HRs as discussed previously and the one obtained using time-dependent schedulers [87, 15]. This value is larger than the one considered in the last paragraph, which shows that a consecutive analysis does not yield the maximum interval-bounded reachability probability over all HRs in a given CTMDP. The reason that this outcome happens is that, by dividing the analysis into two parts, the schedulers of the two analyses can change their decisions more often, and obtain more information than they are supposed to have. Instead of only having the information that the objective is to optimise the reward until* $\mathbf{t} = 4$*, it is now also known whether* $\mathbf{t} \leq 1$ *or not.*

From the discussion of the values of consecutive time points $\mathbf{t}_1 = \delta_1, \mathbf{t}_2 = \mathbf{t}_1 + \delta_2, \ldots$, it follows that we can also use the algorithm to compute bounds for them in an efficient way. Instead of doing analyses with time bounds $\mathbf{t}_1, \mathbf{t}_2, \ldots$, we only have to do analyses with values $\delta_1, \delta_2, \ldots$, which might be much smaller than $\mathbf{t}_1, \mathbf{t}_2, \ldots$. Thus, the fact that the algorithm allows us to handle final and cumulative rewards at the same time has the potential to speed up such a series of analyses.

3.2 Abstracting Prism Models

To take advantage of Proposition 2, we want to avoid actually constructing the CTMC to be abstracted. Doing so allows us to handle models which are too large to be handled in an explicit-state form. For this approach, we can use non-probabilistic model checkers which feature a guarded-command language, like NuSMV [21]. Such a tool can work with an OBDD-based representation of PMs as in Definition 18, and compute the set of reachable states. We can then specify some OBDDs representing *predicates*, i. e., sets of concrete states. These sets can be used to split the state space by subsuming all concrete states that are contained in the same subset of predicates, thus to obtain an OBDD partitioning as in Definition 19.

Next, we consider the ECTMC abstraction of a given Prism model.

Definition 20 *Consider a PM* $m = (Var, init, C, succ, R_c, R_f)$ *with induced CTMC* $C = (S, \mathbf{R})$*, and a partitioning* $\mathfrak{P} = \langle \mathfrak{z}_0, \ldots, \mathfrak{z}_{n-1} \rangle$ *of its state space. The* ECTMC abstraction *of* m *is defined as* $C \stackrel{\text{def}}{=} (\mathfrak{P}, \widehat{Act}, \mathbf{I}^\ell, \mathbf{I}^u)$ *with* $\widehat{Act} \stackrel{\text{def}}{=} \{\hat{\alpha} \colon C \rightharpoonup \mathfrak{P}\}$*. We let* $A(\mathfrak{z}, \hat{\alpha})$ *denote the set of all* $s \in \mathfrak{z}$ *such that* $Dom(succ(s, \cdot)) = Dom(\hat{\alpha})$ *(that is, the domains of the two partial functions agree), and for all applicable* $c \in C$ *we have* $\overline{succ}(s, c) \in \hat{\alpha}(c)$*. We then choose the domain of* $\mathbf{I}^\ell$ *and* $\mathbf{I}^u$ *such that, for all* $\mathfrak{z} \in \mathfrak{P}$*, we have*

$$Dom(\mathbf{I}^\ell(\mathfrak{z}, \cdot)) \stackrel{\text{def}}{=} Dom(\mathbf{I}^u(\mathfrak{z}, \cdot)) \stackrel{\text{def}}{=} \{\hat{\alpha} \in \widehat{Act} \mid A(\mathfrak{z}, \hat{\alpha}) \neq \emptyset\}.$$

Then, for $\mathfrak{z}, \mathfrak{z}' \in \mathfrak{P}$*, and* $\hat{\alpha} \in Dom(\mathbf{I}^u(\mathfrak{z}, \cdot))$ *we define*

$$\mathbf{I}^\ell(\mathfrak{z}, \hat{\alpha})(\mathfrak{z}') \stackrel{\text{def}}{=} \min_{s \in A(\mathfrak{z}, \hat{\alpha})} \sum_{\substack{(s', \lambda) \in succ(s), \\ s' \in \mathfrak{z}'}} \lambda,$$

and accordingly $\mathbf{I}^u$ *using* max*. The abstract reward structure* $\mathbf{r} \stackrel{\text{def}}{=} (\mathbf{r}_c, \mathbf{r}_f)$ *is defined as* $\mathbf{r}_c(\mathfrak{z}) \stackrel{\text{def}}{=} \max_{s \in \mathfrak{z}} R_c(s)$*, and* $\mathbf{r}_f(\mathfrak{z}) \stackrel{\text{def}}{=} \max_{s \in \mathfrak{z}} R_f(s)$*.*

By construction, the CTMDP semantics of the ECTMC fulfils the requirements of Proposition 2 for a correct abstraction of the CTMC semantics of the Prism model. It is also monotone in the sense that, by using a refined partitioning, we cannot obtain worse bounds than with a coarser partitioning.

Proposition 3 *Consider a PM* $m = (Var, init, C, succ, R_c, R_f)$ *with a partitioning* $\mathfrak{P} = \langle \mathfrak{z}_0, \ldots, \mathfrak{z}_{n-1} \rangle$ *of the state space of its induced CTMC, and a further partitioning* $\mathfrak{P}' = \langle \mathfrak{z}'_0, \ldots, \mathfrak{z}'_{m-1} \rangle$ *such that for each* $\mathfrak{z}_t \in \mathfrak{P}$ *we find* $\mathfrak{z}'_{t,1}, \ldots, \mathfrak{z}'_{t,u} \in \mathfrak{P}'$ *such that* $\mathfrak{z}_t = \bigcup_{j=1}^{u} \mathfrak{z}'_{t,j}$*.*

Then, for two ECTMC abstractions $C \stackrel{\text{def}}{=} (\mathfrak{P}, \widehat{Act}, \mathbf{I}^\ell, \mathbf{I}^u)$ *and* $C' \stackrel{\text{def}}{=} (\mathfrak{P}', \widehat{Act'}, \mathbf{I}^{\ell\prime}, \mathbf{I}^{u\prime})$ *with corresponding reward structures* $\mathbf{r}$ *and* $\mathbf{r}'$*, we have* $\mathbf{V}^{\max}(C, \mathfrak{z}_t, \mathbf{r}, \mathbf{t}) \geq \mathbf{V}^{\max}(C', \mathfrak{z}'_{t,j}, \mathbf{r}', \mathbf{t})$*.*

Proof sketch: We can show that, for an arbitrary $\varepsilon > 0$, we have that $\mathbf{V}^{\max}$ of a state $\mathfrak{z}$ of the original partition plus ε is at least equal to the value of a state $\mathfrak{z}'$ of the refined partition for which we have $\mathfrak{z}' \subseteq \mathfrak{z}$. This result implies that the same holds for $\varepsilon = 0$, which means that the value of a state of the refined partition cannot be higher than the one of the original abstraction.

We use the fact that we can apply Algorithm 1 to compute values up to any precision $\varepsilon > 0$. Consider the runs of the algorithm on the original, and refined partitioning. Before the execution of the main loop at Line 5, we have that $q_{k+1}(\mathfrak{z}) = q_{k+1}(\mathfrak{z}') = 0$. For each execution of the main loop, we have a maximising decision in $\mathfrak{z}'$, leading to $v_i(\mathfrak{z}')$ to be added to $q_{i+1}(\mathfrak{z}')$ to obtain $q_i(\mathfrak{z}')$. We can construct a decision for $\mathfrak{z}$ such that $v_i(\mathfrak{z}') \leq v_i(\mathfrak{z})$. This result means that, in each iteration of the main loop, the q_i of $\mathfrak{z}'$ can never become larger than the one of $\mathfrak{z}$. Thus, after the termination of the main loop and the algorithm, the value obtained for the coarser abstraction is at least as high as the one in the refined abstraction. □ The full proof can be found in the Appendix.

Example 8 *Consider the* Prism *model Fig. 1(d). As seen in Example 5, its induced CTMC is the model in Fig. 1(a). With the partitioning of Example 5 ($\mathfrak{z}_0 = \{s_0\}$, $\mathfrak{z}_1 = \{s_1, s_2, s_3, s_4\}$, and $\mathfrak{z}_2 = \{s_5\}$), the model in Fig. 1(b) is an abstraction of the CTMC. Assume that the reward specification given denotes cumulative rewards. As we have already seen in Example 4, the actual value for this model is* ≈ 2.70116 *while with the ECTMC abstraction we can bound the value to* $\approx [1.82699, 4.30277]$. *If we split $\mathfrak{z}_1$, yielding the partitioning $\mathfrak{z}'_0 = \{s_0\}$, $\mathfrak{z}'_1 = \{s_1, s_2\}$, $\mathfrak{z}'_2 = \{s_3, s_4\}$, and $\mathfrak{z}'_3 = \{s_5\}$, we obtain the refined ECTMC abstraction of Fig. 1(e). By doing so, the value bounds improve to* $\approx [2.32375, 3.21667]$.

For comparison, we also provide the abstraction when using CTMDPs, cf. Fig. 1(b). When using a CTMDP, we have more transitions, because in this example we basically have to represent the behaviour of each state by a distinct nondeterministic choice, such that the abstraction is not much smaller than the original model. Indeed, as already discussed in the introduction, this problem is likely to occur for models in which there is a large number of different rate values. In the example here, as seen from Example 4, we obtain the same values when using CTMDP or ECTMC abstractions.

On the other hand, if we use abstract Markov chains (ACTMCs) [46] such as in Fig. 1(f), the abstraction would be even smaller. However, it would also be less precise, as we would have to subsume the transitions with different domains of the successor transition $Dom(succ(s, \cdot))$. In this particular example, because of this constraint, we have a transition from $\mathfrak{z}_1$ to $\mathfrak{z}_2$ which states that the transition probability is between 0 *and* 1*, while for the ECTMC abstraction we do not have intervals with a lower bound of* 0*. The bounds we can obtain are* $\approx [1.82699, 4.38]$. *Thus, the upper bound is larger than if using ECTMC abstractions.*

With a given partitioning, we can apply Algorithm 2 to obtain an abstraction of the model. The algorithm computes an ECTMC to provide an upper bound of the model value; a corresponding algorithm for the lower bound can be defined likewise by minimising over the rewards of a given abstract state rather than maximising over them. Indeed, we can use the same partitioning to obtain ECTMCs for the computation of both lower and upper bounds, and thus compute abstractions for both directions at the same time.

The algorithm does some initializations, and afterwards, in line 6, calls Algorithm 3. This algorithm descends into the OBDD partitioning (lines 5 to 16), visiting each state of the model explicitly. When a specific state s contained in an abstract state $\mathfrak{z}$ is reached (lines 18 to 31), we extend the abstracting model to take into account the behaviour of this state. In lines 19 and 20, we extend the upper bounds for the reward rates of $\mathfrak{z}$ such that they are at least as high as those of s. Notice that to compute the reward rates of this state we use the original high-level PM, not the OBDD representation. Then, in lines 22 to 31, we handle the transition rates, thus to include the rates of the concrete state. We again use the high-level model, this time to compute the set of commands that are enabled in the current state (line 22), the corresponding action $\hat{\alpha}$ (line 23, cf. Definition 20), and the concrete successor states with their corresponding rates (line 24). For each of them, we use the function sAbs to obtain the abstract state it belongs to. For all successor states, we add up the rates to the same abstract state (line 28). Then, starting from line 29, we apply the actual widening of the rates.

Function sAbs works as follows. Because we use a variable order in which the variables encoding states are placed above the variables for the abstract states, each state $s \in S$ induces a path in the OBDD P which ends at the OBDD node that represents the abstract state of s. We follow the unique path, given by the encoding of s, to the terminal node labelled with 1. This following yields the encoding of the abstract state of s. The running time of sAbs is therefore linear in the number of OBDD variables.

Let $n = |S|$ be the number of concrete states of the model under consideration, let k be the total number of positive transitions, and let c be the number of OBDD variables. Algorithm 3 visits each state and transition once. Because the block number variables are placed at the bottom of the variable order, accessing the block number of a state in function sAbs has a running time of $O(c)$. From this result, we have that the overall complexity of Algorithm 2 is $O((n + k) \cdot c)$.

Algorithm 2: Compute ECTMC and reward structure from a given partitioning $\mathfrak{P}$ with OBDD $\mathsf{b}_{\mathfrak{P}} = (\mathsf{N}, \mathsf{n}_{\mathfrak{P}}, \mathsf{h}, \mathsf{l}, \mathsf{v})$ of PM $m = (Var, init, C, succ, R_c, R_f)$.

1 **global** $\mathbf{I}^\ell, \mathbf{I}^u, \mathbf{r}_c, \mathbf{r}_f$ (cf. Algorithm 3)
2 $\mathbf{I}^\ell(\cdot,\cdot)(\cdot) :=$ undefined
3 $\mathbf{I}^u(\cdot,\cdot)(\cdot) :=$ undefined
4 $\mathbf{r}_c(\cdot) := \mathbf{r}_f(\cdot) := -\infty$
5 $\mathfrak{A} := \{0, 1, \ldots, |\mathfrak{P}| - 1\}$
6 approx($\mathsf{n}_{\mathfrak{P}}$, 0)
7 **return** $\left((\mathfrak{A}, \mathbf{I}^\ell, \mathbf{I}^u), (\mathbf{r}_c, \mathbf{r}_f)\right)$

In the discussion so far, we assumed that it is already clear how the set of concrete states shall be divided into abstract states. We might however come across models where this is not clear, or where the results obtained from the abstraction are unsatisfactory. In these cases, we have to apply *refinement*, that is, split existing abstract states into new ones. For other model types, such refinement procedures already exist [48, 42]. In the analysis types considered before, schedulers sufficed which fix a decision per state, and take neither the past history nor number of steps before the state was entered into account. Then, depending on the decisions of the scheduler per state, new predicates are introduced to split the state space. In our case, such simple schedulers are not sufficient to obtain extremal values, as has already been shown for the simpler case of time-bounded reachability [4]. Thus, it is not clear how to introduce predicates to split the state space.

As a first heuristic, we do the following. We treat an OBDD representation of a PM as a labelled transition system, in which the commands play the role of the labels. We then use an existing algorithm to symbolically compute (non-probabilistic) strong bisimulations [83], but stop the algorithm after a number of steps. This way, we obtain a

Algorithm 3: Procedure approx(n, level).

1 **global** $\mathbf{I}^\ell, \mathbf{I}^u, \mathbf{r}_c, \mathbf{r}_f$ (cf. Algorithm 2)
2 **if** n = bdd_0 **then return**
3
4 **else if** level < leafLevel **then**
5 *// We are still at a variable level.*
6 x = varAtLevel(level)
7 **if** n ≠ bdd_1 **and** x = v(n) **then**
8 n_l := l(n), n_h := h(n)
9 **else**
10 n_l := n, n_h := n
11 **if** x ∈ $\mathfrak{V}$ **then**
12 $\mathfrak{z}$(x) := 0, approx(n_l, level + 1)
13 $\mathfrak{z}$(x) := 1, approx(n_h, level + 1)
14 **else**
15 s(x) := 0, approx(n_l, level + 1)
16 s(x) := 1, approx(n_h, level + 1)
17 **else**
18 *// We have traversed all variable levels.*
19 $\mathbf{r}_c(\mathfrak{z}) := \max(\mathbf{r}_c(\mathfrak{z}), R_c(s))$
20 $\mathbf{r}_f(\mathfrak{z}) := \max(\mathbf{r}_f(\mathfrak{z}), R_f(s))$
21
22 $C := Dom(succ(s, \cdot))$ *// commands enabled in s*
23 $\hat{\alpha} = \{(c, \text{sAbs}(\mathsf{n}_{\mathfrak{P}}, s')) \,\big|\, c \in C \wedge s' = \overline{succ}(s, c)\}$
24 $A := succ(s)$
25 $\Lambda(\cdot) := 0$
26 **forall** $(s', \lambda) \in A$ **do**
27 $\mathfrak{z}' := \text{sAbs}(\mathsf{n}_{\mathfrak{P}}, s')$
28 $\Lambda(\mathfrak{z}') := \Lambda(\mathfrak{z}') + \lambda$
29 **forall** $\mathfrak{z}' \in \mathfrak{A}$ **do**
30 $\mathbf{I}^\ell(\mathfrak{z}, \hat{\alpha})(\mathfrak{z}') := \min(\mathbf{I}^\ell(\mathfrak{z}, \hat{\alpha})(\mathfrak{z}'), \Lambda(\mathfrak{z}'))$
31 $\mathbf{I}^u(\mathfrak{z}, \hat{\alpha})(\mathfrak{z}') := \max(\mathbf{I}^u(\mathfrak{z}, \hat{\alpha})(\mathfrak{z}'), \Lambda(\mathfrak{z}'))$

partitioning in the form of Definition 19. As we will see later in Section 4, although the method is not guaranteed to yield a good abstraction, it can work well in practice.

The method discussed works for a very general class of PMs and arbitrary state partitionings. However, because it is based on explicitly visiting each concrete state at least once, it may take much time to perform for large models. To tackle this problem, a parallel implementation of the technique is possible. Given a computing system with a number of processors, one can symbolically divide the states of the model, such that each processor works on a different part of the OBDD representing the state space. Each processor can then process the model part it is assigned to. The only point of interaction is the widening of the rates and reward rates of the abstract model. On a shared memory architecture, one could use different semaphores for the reward rates and successor transitions of each abstract state to avoid delays. Without shared memory, the processors can compute partial abstractions separately, which are merged after the computations are finished. This technique is faster, but has the disadvantage of having to store several (partial) copies of the abstraction. If the state space is divided such that all states of an abstract state are assigned to a single processor, no locking is needed, and the overhead is reduced.

As an alternative to parallelisation, it should also be possible to use optimisation methods over variants of BDDs [58, 51, 66, 79] to compute the rate and reward intervals symbolically rather than rely on explicit enumeration of all possible variable assignments.

4 Case Studies

To show the practicality of the method, we applied it on two case studies from classical performance and dependability engineering [37, 22]. We implemented the techniques of Algorithm 1, and Algorithm 2. To represent the ECTMCs, we used a sparse-matrix-like data structure.

Where possible, we compared the results to Prism. Prism always starts by building an MTBDD representation of the model under consideration. The subsequent analysis is then performed using *value iteration* in the CTMC semantics similarly to Algorithm 1. The data structure used here is either an MTBDD, a sparse matrix, or a *hybrid* structure [56]. In the latter, values for the model states are stored explicitly, but parts of the transition structure are stored implicitly.

Table II: Number of repairs in the workstation cluster until time $\mathbf{t} = 500$

	ECTMC Results			
N	$\lvert\mathfrak{P}\rvert$	Time	Memory	Interval
32	19 420	107.15	90.03	[64.176, 64.199]
64	19 420	109.43	86.28	[127.980, 128.490]
128	19 420	115.93	89.54	[255.455, 259.797]
256	19 420	132.89	94.58	[509.000, 580.485]
512	19 420	181.99	91.87	[869.749, 900.052]
1 024	19 420	412.43	107.22	[905.018, 905.200]
2 048	19 420	1 335.54	103.31	[905.766, 905.767]
4 096	19 420	5 298.29	104.89	[905.955, 905.955]
8 192	19 420	28 361.36	132.48	[906.040, 906.040]
16 384	19 420	147 691.30	139.56	[906.084, 906.084]

For all experiments, we used a Quad-Core AMD Opteron™ Processor 8356 (of which we only used one core) with 2300 MHz, and 64 GB of main memory.

We consider a fault-tolerant workstation cluster [37]. It consists of two sub-clusters, which, in turn, contain N workstations connected via a central switch. The two switches are connected via a backbone. Each component of the system can break down, and is then fixed by a single repair unit responsible for the entire system.

We are interested in the expected number of repairs until a time bound of $\mathbf{t} = 500$. This property can be expressed using cumulative rewards. For N up to 512, the model has been successfully analysed before using Prism[1]. While the existing analysis methods worked well for model instantiations up to this N, and somewhat above, the techniques do not work well anymore for a very large number of workstations. Constructing the model using MTBDDs seems not to be problematic, but the subsequent analyses cannot be performed successfully. The sparse-matrix and the hybrid method fail at some point, because they rely on an explicit representation of the state space, and thus run out of memory. Also, the MTBDD-based value iteration fails at some point, and works rather slowly. The reason for this failure is probably that, during the value iteration, a large number of different non-terminal nodes

[1] `http://www.prismmodelchecker.org/casestudies/cluster.php#mc`, Property `R{"num_repairs"}=?[ C<=T ]`.

Table I: PRISM results for the number of repairs in the workstation cluster until $\mathbf{t} = 500$

			sparse engine		hybrid engine		symbolic engine		
N	$\lvert S\rvert$	$\lvert R\rvert$	Time	Memory	Time	Memory	Time	Memory	Result
32	$3.87 \cdot 10^4$	$1.86 \cdot 10^5$	9.29	36.67	14.90	37.48	13 791.20	184.49	64.17635
64	$1.51 \cdot 10^5$	$7.33 \cdot 10^5$	62.11	42.92	88.93	41.69	– *Time out* –		127.98101
128	$5.97 \cdot 10^5$	$2.91 \cdot 10^6$	380.51	60.18	585.55	54.48	– *Time out* –		255.48297
256	$2.37 \cdot 10^6$	$1.16 \cdot 10^7$	3 182.73	141.71	4 737.73	98.27	– *Time out* –		509.58417
512	$9.47 \cdot 10^6$	$4.62 \cdot 10^7$	10 540.54	817.39	14 965.74	284.66	– *Time out* –		896.80612
1 024	$3.78 \cdot 10^7$	$1.85 \cdot 10^8$	13 242.08	3 154.91	25 513.31	1 014.79	– *Time out* –		905.19921
2 048	$1.51 \cdot 10^8$	$7.39 \cdot 10^8$	– *Time out* –		– *Time out* –		– *Time out* –		??
4 096	$6.04 \cdot 10^8$	$2.95 \cdot 10^9$	– *Memory out* –		– *Time out* –		– *Time out* –		??
8 192	$2.42 \cdot 10^9$	$1.18 \cdot 10^{10}$	– *Memory out* –		– *Memory out* –		– *Time out* –		??
16 384	$9.66 \cdot 10^9$	$4.72 \cdot 10^{10}$	– *Memory out* –		– *Memory out* –		– *Time out* –		??

appear, which make the MTBDD complex, and thus large and slow to operate on. Detailed information about the performance of PRISM on this case study is given in Table I.

By $|S|$, and $|R|$, we give the approximate number of states, and transitions, resp., of the original CTMC model. For each of the three PRISM engines, we give the running time (columns labelled with Time), and memory consumption (columns labelled with Memory) for computing the expected rewards. An entry of – *Time out* – means that PRISM did not terminate within 160,000 seconds, while an entry of – *Memory out* – indicates that more than 60 GB of memory are required to complete the analysis.

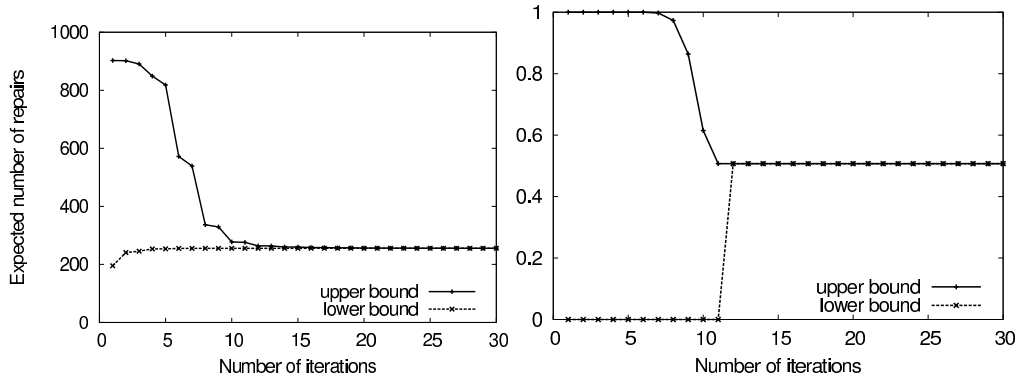
In Table II, we apply the method developed in this paper on several instantiations of the number of workstations N. The results we obtained by our method are given in ECTMC Results. Besides the running time and memory consumption, we give in the column titled $|\mathfrak{P}|$ the number of abstract states we used for the corresponding analysis. The column labelled Interval gives the lower and upper bounds of the actual value of the expected reward.

As we see from the time and memory usage, for smaller models, it is advantageous to use an explicit-state method as implemented in PRISM, because of the additional overhead our method introduces. As instantiations become larger, using the method of this paper becomes worthwhile. While we do not always get precise bounds for all analyses performed with this number of abstract states, we always were able to compute the order of magnitude. Interestingly, the value bounds get tighter with an increasing number of model states.

As discussed in Section 3, we apply a heuristic refinement algorithm based on bisimulations for labelled transition systems. We use the symbolic algorithm [83] for computing (non-stochastic) strong bisimulations to obtain a suitable state partitioning. We abort its fix-point iteration prematurely after a user-specified number n of iterations. In Fig. 3, we show how the quality of the approximation evolves depending on n. The behaviour of the cluster case study is shown on the left. One can observe that the width of the computed interval converges quickly to the actual value when increasing the number of iterations.

Notably, if we use the same number of refinement steps, for all model instantiations considered, $|\mathfrak{P}|$ stayed constant, although the number of model states $|S|$ was different for each instantiation (cf. Table II).

Table III contains more detailed running times for the cluster benchmark with $N = 2048$ workstations using the ECTMC abstraction for different numbers of bisimulation iterations, which are given in the first column. The second column contains the number of abstract states. The running times in seconds are given separately for the different main operations. The running times include the call to NUSMV to generate the OBDDs for the underlying transition system (col. 3), the given number of bisimulation iterations (col. 4), the construction of the ECTMC from the partitioning (col. 5), the value iteration to compute the reward interval (col. 6), and finally the total computation time (col. 7). The last two columns contain the memory consumption in Megabytes, and the computed reward interval.



(a) Workstation cluster with $N = 128$.

(b) Google file system with $M = 256$.

Figure 3: Quality of the ECTMC approximation for different numbers of bisimulation iterations.

Table III: Detailed experimental results for the workstation cluster ($N = 2048$, $\mathbf{t} = 500$) using the ECTMC abstraction

		Running Times						
Iterations	$\lvert\mathfrak{P}\rvert$	NuSMV	Refinement	ECTMC	Value Iter.	Total	Memory	Interval
5	2440	64.75	0.93	991.16	13.45	1070.34	56.13	[902.092, 905.771]
10	9216	68.46	6.96	1135.57	41.68	1252.79	70.11	[905.739, 905.767]
15	19420	65.25	18.26	1029.23	116.89	1229.78	103.31	[905.766, 905.767]
20	34596	73.93	43.71	1150.70	213.72	1482.29	134.62	[905.767, 905.767]
25	52600	69.82	78.88	1070.13	321.43	1540.49	180.85	[905.767, 905.767]
30	76176	66.43	126.85	1052.87	466.31	1712.76	227.80	[905.767, 905.767]

Table V: Google file system with property 12 and $\mathbf{t} = 60$, $N = 100\,000$, $C = 5000$

	ECTMC Results			
M	$\lvert\mathfrak{P}\rvert$	Time	Mem.	Interval
32	6 138	30.28	72.28	[0.0000, 0.0000]
64	18 042	251.86	86.53	[0.5071, 0.5071]
128	29 515	822.08	142.23	[0.5071, 0.5071]
256	29 515	1 531.46	147.66	[0.5071, 0.5071]
512	29 515	2 951.37	129.72	[0.5071, 0.5071]
1 024	29 515	6 776.42	128.40	[0.5071, 0.5071]
2 048	29 515	14 957.94	137.47	[0.5071, 0.5071]

We additionally consider a replicated file system as used as part of the Google search engine [22, 2]. Originally, the model was given as a generalised stochastic Petri net, but was transformed to a PM for the analysis.

Files are divided into *chunks* of equal size. Several copies of each chunk reside at several *chunk servers*. There is a single *master* server which knows the location of the chunk copies. If a user of the file system wants to access a certain chunk of a file, it asks the master for the location. Data transfer then takes place directly between a chunk server and the user. The model describes the life cycle of a single chunk, but accounts for the load caused by the other chunks.

The model features three parameters: M is the number of chunk servers, with C we denote the number of chunks a chunk server may store, and the total number of chunks is N.

We consider the minimal probability over all states in which severe hardware problems have occurred (the master server is down, and more than three quarters of the chunk servers are down), that within time $\mathbf{t}$ a state will be reached in which a guaranteed quality-of-service level (all three chunk copies are present, and the master server is available) holds. This is a bounded-reachability property, and thus

Table IV: PRISM results for the Google file system

			sparse engine		hybrid engine		symbolic engine		
N	$\lvert S\rvert$	$\lvert R\rvert$	Time	Memory	Time	Memory	Time	Memory	Result
32	$6.15 \cdot 10^3$	$4.03 \cdot 10^4$	1.60	607.42	1.68	607.78	67.95	624.01	0.000000
64	$2.46 \cdot 10^4$	$1.66 \cdot 10^5$	34.72	614.24	71.04	616.84	66319.89	791.23	0.507119
128	$9.83 \cdot 10^4$	$6.77 \cdot 10^5$	464.25	624.81	1 890.41	627.23	– *Time out* –		0.507119
256	$3.93 \cdot 10^5$	$2.74 \cdot 10^6$	3 083.24	683.27	26 132.52	701.11	– *Time out* –		0.507119
512	$1.57 \cdot 10^6$	$1.11 \cdot 10^7$	41 901.82	938.10	– *Time out* –		– *Time out* –		0.507119
1 024	$6.29 \cdot 10^6$	$4.44 \cdot 10^7$	– *Time out* –		– *Time out* –		– *Time out* –		??
2 048	$2.52 \cdot 10^7$	$1.78 \cdot 10^8$	– *Time out* –		– *Time out* –		– *Time out* –		??

based on final rewards.

We fix $C = 5000$, $N = 100\,000$, and $\mathbf{t} = 60$; and we provide results for several M in Table V. In the analyses with PRISM (see Table IV), we used an improved OBDD variable order, such that the performance results are better than in [2]. In contrast to the previous case study, PRISM's sparse matrix engine was faster, and did not use more memory compared to the hybrid engine. The symbolic engine was again the slowest. The MTBDD representation of the model requires more memory per concrete state compared to the previous case study. We assume that this requirement is because the number of different rates occurring is much higher, and because some of the rates are obtained by multiplying state variables, thus leading to a more complex MTBDD structure. Notice that in this model, from a certain value of M onward, the probability discussed is almost the same.

We give detailed information on the instance with $M = 128$ of the Google file system in Table VI and Fig. 3 (right-hand side) for different numbers n of bisimulation iterations. We can again observe that the computed interval for the bounded-reachability property quickly converges to the actual probability with increasing n.

From Table II, we see that the obtained reward converges to a fixed value with increasing model parameter N. Therefore, if one is interested in the value obtained if the model parameters go towards infinity, methods concerned with the limiting behaviour [7, 8, 40, 1] might be more appropriate. As discussed in the introduction, our method targets at finding conservative bounds for the model under consideration. The asymptotic methods we are aware of do not provide such guaranteed bounds in the general case. Also, the models which we used are specified in a very general guarded commands language, and cannot be adequately modelled in a restricted input language, as usually required as the input format of asymptotic analyses. Our method provides correct results for all choices of the parameters, whereas for smaller or medium model sizes the results obtained by asymptotic analyses would be very far off, even if it would be possible to perform such an analysis for increasing parameter sizes. Many scalable models (e.g. many of the ones from the homepage of the probabilistic model checker PRISM) behave in a similar way.

5 Conclusion

We developed an efficient method to compute extremal values of CTMDPs over HR schedulers. It can be used to safely bound quantities of interest of CTMCs, by abstracting them into a special class of CTMDPs, and then applying this method. Experimental results have shown that the approach works well in practice.

There are a number of possible future works. The current refinement technique surely does not yield an optimal partitioning of the state space in all cases. We thus want to see how the scheduler we implicitly obtain by Algorithm 1 can be used to refine the model. The abstraction technique could also be extended to other property classes and models. For instance, models already involving nondeterminism could be abstracted and approximated using Markov games [10, 69, 70]. It would also be interesting to see how a parallelised or symbolic abstraction method sketched at the end of Section 3 performs against the one currently implemented. Using a three-valued logic [50], the technique could also be integrated into an existing

Table VI: Detailed experimental results for the Google file system ($M = 128$, $\mathbf{t} = 60$) using the ECTMC abstraction

		Running Times						
Iterations	$\lvert\mathfrak{P}\rvert$	NuSMV	Refinement	ECTMC	Value Iter.	Total	Memory	Interval
5	3204	0.86	3.19	1.80	70.42	76.98	65.00	[0.000000, 0.999999]
10	15564	0.81	15.44	2.01	426.32	445.42	92.08	[0.000000, 0.614823]
15	29515	0.83	41.24	2.20	716.67	761.89	142.23	[0.507119, 0.507119]
20	42725	0.83	82.54	2.41	1040.74	1127.53	177.59	[0.507119, 0.507119]
25	57472	0.79	137.50	2.57	1349.73	1491.65	249.21	[0.507119, 0.507119]
30	69932	0.82	191.83	2.62	1652.66	1849.04	279.13	[0.507119, 0.507119]

probabilistic (CSL) model checker.

Appendix

Proof of Proposition 1

Definition 21 *Let $\mathcal{D} = (S, \mathbf{P})$ be a DTMC. For $s_0, s_k \in S$, and $k \in \mathbb{N}$, we define $\pi^{\mathcal{D}}(s_0, k, s_k) = \Pr(X_k^{\mathcal{D},s_0} = s_k)$.*

Definition 21 specifies the transient probability to be in state s_k at step k when having started in state s_0.

Corollary 1 *Notice that, for $\mathcal{D} = (S, \mathbf{P})$, $k \in \mathbb{N}$, and $s_0, s_k \in S$, it holds that*

$$\begin{aligned}
&\pi^{\mathcal{D}}(s_0, k, s_k) \\
&= \mathbf{P}^k(s_0, s) \\
&\stackrel{\text{def}}{=} \sum_{s_1 \in S} \mathbf{P}(s_0, s_1) \cdot \sum_{s_2 \in S} \mathbf{P}(s_1, s_2) \cdot \sum_{s_3 \in S} \mathbf{P}(s_2, s_3) \\
&\qquad \cdot \dots \cdot \sum_{s_{k-1} \in S} \mathbf{P}(s_{k-2}, s_{k-1}) \cdot \mathbf{P}(s_{k-1}, s_k),
\end{aligned}$$

i. e., the transient probability in a DTMC can be expressed using matrix multiplications.

We extend the definition of values to discrete-time models, which will be used in the further parts of the proof.

Definition 22 *Let $\mathcal{D} = (S, \mathbf{P})$ be a DTMC, let $\mathbf{r} = (\mathbf{r}_c, \mathbf{r}_f)$ be a reward structure, and let $\mathbf{u}, \mathbf{t} \geq 0$. We define*

$$\begin{aligned}
&\mathbf{V}(\mathcal{D}, s_0, \mathbf{r}, \mathbf{t}, \mathbf{u}) \\
&\stackrel{\text{def}}{=} \sum_{i=0}^{\infty} \left(\phi_{\mathbf{ut}}(i) \sum_{s_i \in S} \pi^{\mathcal{D}}(s_0, i, s_i) \mathbf{r}_f(s_i) \right. \\
&\qquad \left. + \psi_{\mathbf{ut}}(i) \sum_{s_i \in S} \pi^{\mathcal{D}}(s_0, i, s_i) \frac{\mathbf{r}_c(s_i)}{\mathbf{u}} \right).
\end{aligned}$$

We define a type of schedulers which are simpler than the HRs of Definition 7, and at the same time generalise the CD of Definition 9.

Definition 23 *A* time-abstract, history-abstract, counting, randomised scheduler (CR) *for a DTMDP $\mathcal{D} = (S, Act, \mathbf{P})$ or a CTMDP $\mathcal{C} = (S, Act, \mathbf{R})$ is a function $\sigma \colon (S \times \mathbb{N}) \to Distr(Act)$ such that, for all $s \in S$, and $n \in \mathbb{N}$, if $\sigma(s, n)(\alpha) > 0$, then $\alpha \in Act(s)$. With Σ_{CR}, we denote the set of all CRs.*

Definition 24 *Assume we are given a CTMDP $\mathcal{C} = (S, Act, \mathbf{R})$, and a CR $\sigma \colon (S \times \mathbb{N}) \to Distr(Act)$. We define the* induced CTMC *as $\mathcal{C}_\sigma \stackrel{\text{def}}{=} (S', \mathbf{R}')$ with*

- $S' \stackrel{\text{def}}{=} S \times \mathbb{N}$,
- $\mathbf{R}'((s, n), (s', n+1)) \stackrel{\text{def}}{=} \sum_{\alpha \in Act} \sigma(s, n)(\alpha) \cdot \mathbf{R}(s, \alpha, s')$ *for $s, s' \in S$ and $n \in \mathbb{N}$, and $\mathbf{R}'(\cdot) \stackrel{\text{def}}{=} 0$ else.*

Let $X^{\mathcal{C}_\sigma, s_0} \colon (\Omega_{\mathcal{C}_\sigma} \times \mathbb{R}_{\geq 0}) \to (S \times \mathbb{N})$ be the stochastic process of the CTMC $\mathcal{C}_\sigma$, and let $f \colon (S \times \mathbb{N}) \to S$ with $f(s, n) = s$. Induced DTMCs of DTMDPs are defined accordingly. The induced stochastic process *$X^{\mathcal{C},\sigma,s_0} \colon (\Omega_{\mathcal{C}_\sigma} \times \mathbb{R}_{\geq 0}) \to S$ of $\mathcal{C}$ and σ starting in $s_0 \in S$ is then defined as $X_t^{\mathcal{C},\sigma,s_0} = f \circ X_t^{\mathcal{C}_\sigma,(s_0,0)}$ for $t \in \mathbb{R}_{\geq 0}$. Definitions for DTMDPs are likewise using $\mathbf{P}$ instead of $\mathbf{R}$.*

We extend the notation of transient probabilities to scheduled nondeterministic models. It is known that, for DTMDPs, CR schedulers are as powerful as HR schedulers.

Definition 25 *For a DTMDP $\mathcal{D} = (S, Act, \mathbf{P})$, and a scheduler $\sigma \in \Sigma_{HR} \cup \Sigma_{CD} \cup \Sigma_{CR}$, we define $\pi^{\mathcal{D},\sigma}(s_0, k, s_k) = \Pr(X_k^{\mathcal{D},\sigma,s_0} = s_k)$ for all $k \in \mathbb{N}$ and $s_0, s_k \in S$.*

Lemma 2 *Consider a DTMDP $\mathcal{D} = (S, Act, \mathbf{P})$, and a HR σ_{hr}. Then there is a CR σ_{cr} such that, for all $s_0, s_n \in S$, and $n \in \mathbb{N}$, we have $\pi^{\mathcal{D},\sigma_{hr}}(s_0, n, s_n) = \pi^{\mathcal{D},\sigma_{cr}}(s_0, n, s_n)$.*

Proof: The proof is given in [75] and [68, Theorem 5.5.1], where CR are denoted as MR (Markov randomised) policies. □

Definition 26 *For a CTMC $\mathcal{C} = (S, \mathbf{R})$, we let* emb$(\mathcal{C}) \stackrel{\text{def}}{=} (S, \mathbf{P})$ *denote the DTMC such that, for all $s, s' \in S$, we have $\mathbf{P}(s, s') \stackrel{\text{def}}{=} \mathbf{R}(s, s')/\mathbf{u}(\mathcal{C})$.*

The following lemma states how values of CTMCs can be computed using the embedded discrete-time model.

Lemma 3 *Let $\mathcal{C} = (S, \mathbf{R})$ be a CTMC with a reward structure $\mathbf{r} = (\mathbf{r}_c, \mathbf{r}_f)$. Let $\mathbf{u} = \mathbf{u}(\mathcal{C})$. Then, for $\mathbf{t} \geq 0$, and all $s_0 \in S$, the following holds.*

$$\mathbf{V}(\mathcal{C}, s_0, \mathbf{r}, \mathbf{t}) = \mathbf{V}(\text{emb}(\mathcal{C}), s_0, \mathbf{r}, \mathbf{t}, \mathbf{u})$$

Proof: By Definition 12, for $s_0 \in S$, it holds that

$$\mathbf{V}(C, s_0, \mathbf{r}, \mathbf{t}) = \underbrace{\mathbf{E}\left[\int_0^{\mathbf{t}} \mathbf{r}_c(X_u^{C,s_0})\,\mathrm{d}u\right]}_{\text{accumulated}} + \underbrace{\mathbf{E}\left[\mathbf{r}_f(X_{\mathbf{t}}^{C,s_0})\right]}_{\text{final}}.$$

Thus, we can divide $\mathbf{V}(C, s_0, \mathbf{r}, \mathbf{t})$ into a sum of accumulated and final. We have

$$\begin{aligned}
\text{final} &= \mathbf{E}\left[\mathbf{r}_f(X_{\mathbf{t}}^{C,s_0})\right] \\
&= \sum_{s\in S} \Pr(X_{\mathbf{t}}^{C,s_0} = s)\mathbf{r}_f(s) \\
&= \sum_{s\in S}\left(\sum_{i=0}^{\infty} \pi^{\mathrm{emb}(C)}(s_0, i, s)\phi_{\mathbf{ut}}(i)\right)\mathbf{r}_f(s) \\
&= \sum_{i=0}^{\infty} \phi_{\mathbf{ut}}(i) \sum_{s\in S} \pi^{\mathrm{emb}(C)}(s_0, i, s)\mathbf{r}_f(s).
\end{aligned}$$

Further, Kwiatkowska et al. [54, Theorem 1] have shown that

$$\text{accumulated} = \sum_{i=0}^{\infty} \psi_{\mathbf{ut}}(i) \sum_{s\in S} \pi^{\mathrm{emb}(C)}(s_0, i, s)\frac{\mathbf{r}_c(s)}{\mathbf{u}}.$$

Thus,

$$\begin{aligned}
\mathbf{V}(C, s_0, \mathbf{r}, \mathbf{t}) &= \text{accumulated} + \text{final} \\
&= \sum_{i=0}^{\infty}\left(\phi_{\mathbf{ut}}(i) \sum_{s\in S} \pi^{\mathrm{emb}(C)}(s_0, i, s)\mathbf{r}_f(s)\right. \\
&\quad \left. + \psi_{\mathbf{ut}}(i) \sum_{s\in S} \pi^{\mathrm{emb}(C)}(s_0, i, s)\frac{\mathbf{r}_c(s)}{\mathbf{u}}\right) \\
&= \mathbf{V}(\mathrm{emb}(C), s_0, \mathbf{r}, \mathbf{t}, \mathbf{u}).
\end{aligned}$$

□

We can now show that the restricted class CR suffices to obtain optimal values.

Lemma 4 *Given a CTMDP $C = (S, Act, \mathbf{P})$, and $\sigma_{hr} \in \Sigma_{HR}$, there is $\sigma_{cr} \in \Sigma_{CR}$ such that $\mathbf{V}(X^{C,\sigma_{hr},s_0}, \mathbf{r}, \mathbf{t}) = \mathbf{V}(X^{C,\sigma_{cr},s_0}, \mathbf{r}, \mathbf{t})$. Further, for all $\sigma'_{cr} \in \Sigma_{CR}$, we can find $\sigma'_{hr} \in \Sigma_{HR}$ such that $\mathbf{V}(X^{C,\sigma'_{cr},s_0}, \mathbf{r}, \mathbf{t}) = \mathbf{V}(X^{C,\sigma'_{hr},s_0}, \mathbf{r}, \mathbf{t})$.*

Proof: Consider a CTMDP $C = (S, Act, \mathbf{P})$, and $\sigma_{hr} \in \Sigma_{HR}$. By Lemma 2, we can find a scheduler $\sigma_{cr} \in \Sigma_{CR}$ such that

$$\pi^{\mathrm{emb}(C),\sigma_{hr}} = \pi^{\mathrm{emb}(C),\sigma_{cr}}. \tag{1}$$

Define

- $\mathbf{r}^{hr} \stackrel{\text{def}}{=} (\mathbf{r}_c^{hr}, \mathbf{r}_f^{hr})$ with
- $\mathbf{r}_c^{hr}(\beta, s) \stackrel{\text{def}}{=} \mathbf{r}_c(s)$, and
- $\mathbf{r}_f^{cr}(\beta, s) \stackrel{\text{def}}{=} \mathbf{r}_f(s)$;

and let

- $\mathbf{r}^{cr} \stackrel{\text{def}}{=} (\mathbf{r}_c^{cr}, \mathbf{r}_f^{cr})$ with
- $\mathbf{r}_c^{cr}(s, n) \stackrel{\text{def}}{=} \mathbf{r}_c(s)$, and
- $\mathbf{r}_f^{cr}(s, n) \stackrel{\text{def}}{=} \mathbf{r}_f(s)$

for $\beta \in (S \times Act)^*$, $n \in \mathbb{N}$, and $s \in S$. Then for all $s_0 \in S$, we have

$$\begin{aligned}
&\mathbf{E}[\mathbf{r}_f(X_{\mathbf{t}}^{C,\sigma_{hr},s_0})] \\
&= \mathbf{E}[\mathbf{r}_f(f(X_{\mathbf{t}}^{C_{\sigma_{hr}},s_0}))] \\
&= \sum_{s\in S} \Pr(f(X_{\mathbf{t}}^{C_{\sigma_{hr}},s_0}) = s)\mathbf{r}_f(s) \\
&= \sum_{s\in S} \Pr(\exists \beta \in (S \times Act)^*.\, X_{\mathbf{t}}^{C_{\sigma_{hr}},s_0} = (\beta, s))\mathbf{r}_f(s) \\
&= \sum_{\substack{s\in S,\\ \beta\in(S\times Act)^*}} \Pr(X_{\mathbf{t}}^{C_{\sigma_{hr}},s_0} = (\beta, s))\mathbf{r}_f(s) \\
&= \sum_{\substack{s\in S,\\ \beta\in(S\times Act)^*}} \Pr(X_{\mathbf{t}}^{C_{\sigma_{hr}},s_0} = (\beta, s))\mathbf{r}_f^{hr}(\beta, s) \\
&= \mathbf{E}[\mathbf{r}_f^{hr}(X_{\mathbf{t}}^{C_{\sigma_{hr}},s_0})],
\end{aligned}$$

similarly

$$\begin{aligned}
&\mathbf{E}\left[\int_0^{\mathbf{t}} \mathbf{r}_c(X_u^{C,\sigma_{hr},s_0})\,\mathrm{d}u\right] \\
&= \int_0^{\mathbf{t}} \mathbf{E}\left[\mathbf{r}_c(X_u^{C,\sigma_{hr},s_0})\right]\mathrm{d}u \\
&= \int_0^{\mathbf{t}} \mathbf{E}\left[\mathbf{r}_c^{hr}(X_u^{C_{\sigma_{hr}},s_0})\right]\mathrm{d}u \\
&= \mathbf{E}\left[\int_0^{\mathbf{t}} \mathbf{r}_c^{hr}(X_u^{C_{\sigma_{hr}},s_0})\,\mathrm{d}u\right],
\end{aligned}$$

and in turn

$$\mathbf{V}(X^{C,\sigma_{hr},s_0},\mathbf{r},\mathbf{t}) = \mathbf{V}(X^{C_{\sigma_{hr}},s_0},\mathbf{r}^{hr},\mathbf{t}). \tag{2}$$

In the same way, using $n \in \mathbb{N}$ instead of $\beta \in (S \times Act)^*$, one can show

$$\mathbf{V}(X^{C,\sigma_{cr},s_0},\mathbf{r},\mathbf{t}) = \mathbf{V}(X^{C_{\sigma_{cr}},s_0},\mathbf{r}^{cr},\mathbf{t}). \tag{3}$$

Notice that, for $\sigma \in \Sigma_{CR} \cup \Sigma_{HR}$, it is

$$\mathrm{emb}(C_\sigma) = \mathrm{emb}(C)_\sigma. \tag{4}$$

In addition,

$$\begin{aligned}
&\mathbf{V}(\mathrm{emb}(C)_{\sigma_{hr}}, s_0, \mathbf{r}^{hr}, \mathbf{t}, \mathbf{u}) \\
&= \sum_{i=0}^{\infty}\left(\phi_{\mathbf{ut}}(i)\sum_{s\in(S\times Act)^*\times S}\pi^{\mathrm{emb}(C)_{\sigma_{hr}}}(s_0,i,s)\cdot\mathbf{r}_f^{hr}(s)\right. \\
&\quad\left. + \psi_{\mathbf{ut}}(i)\sum_{s\in(S\times Act)^*\times S}\pi^{\mathrm{emb}(C)_{\sigma_{hr}}}(s_0,i,s)\frac{\mathbf{r}_c^{hr}(s)}{\mathbf{u}}\right) \\
&= \sum_{i=0}^{\infty}\left(\phi_{\mathbf{ut}}(i)\sum_{s\in S}\pi^{\mathrm{emb}(C),\sigma_{hr}}(s_0,i,s)\cdot\mathbf{r}_f(s)\right. \\
&\quad\left. + \psi_{\mathbf{ut}}(i)\sum_{s\in S}\pi^{\mathrm{emb}(C),\sigma_{hr}}(s_0,i,s)\frac{\mathbf{r}_c(s)}{\mathbf{u}}\right) \\
&\overset{\text{Eqn. 1}}{=} \sum_{i=0}^{\infty}\left(\phi_{\mathbf{ut}}(i)\sum_{s\in S}\pi^{\mathrm{emb}(C),\sigma_{cr}}(s_0,i,s)\cdot\mathbf{r}_f(s)\right. \\
&\quad\left. + \psi_{\mathbf{ut}}(i)\sum_{s\in S}\pi^{\mathrm{emb}(C),\sigma_{cr}}(s_0,i,s)\frac{\mathbf{r}_c(s)}{\mathbf{u}}\right) \\
&= \sum_{i=0}^{\infty}\left(\phi_{\mathbf{ut}}(i)\sum_{s\in S\times\mathbb{N}}\pi^{\mathrm{emb}(C)_{\sigma_{cr}}}((s_0,0),i,s)\cdot\mathbf{r}_f^{cr}(s)\right. \\
&\quad\left. + \psi_{\mathbf{ut}}(i)\sum_{s\in S\times\mathbb{N}}\pi^{\mathrm{emb}(C)_{\sigma_{cr}}}((s_0,0),i,s)\frac{\mathbf{r}_c^{cr}(s)}{\mathbf{u}}\right) \\
&= \mathbf{V}(\mathrm{emb}(C)_{\sigma_{cr}}, s_0, \mathbf{r}^{cr}, \mathbf{t}, \mathbf{u}).
\end{aligned} \tag{5}$$

From these facts, we have

$$\begin{aligned}
&\mathbf{V}(X^{C,\sigma_{hr},s_0},\mathbf{r},\mathbf{t}) \\
&\overset{\text{Eqn. 2}}{=} \mathbf{V}(X^{C_{\sigma_{hr}},s_0},\mathbf{r}^{hr},\mathbf{t}) \\
&\overset{\text{Lemma 3}}{=} \mathbf{V}(\mathrm{emb}(C_{\sigma_{hr}}),s_0,\mathbf{r}^{hr},\mathbf{t},\mathbf{u}) \\
&\overset{\text{Eqn. 4}}{=} \mathbf{V}(\mathrm{emb}(C)_{\sigma_{hr}},s_0,\mathbf{r}^{hr},\mathbf{t},\mathbf{u}) \\
&\overset{\text{Eqn. 5}}{=} \mathbf{V}(\mathrm{emb}(C)_{\sigma_{cr}},s_0,\mathbf{r}^{cr},\mathbf{t},\mathbf{u}) \\
&\overset{\text{Eqn. 4}}{=} \mathbf{V}(\mathrm{emb}(C_{\sigma_{cr}}),s_0,\mathbf{r}^{cr},\mathbf{t},\mathbf{u}) \\
&\overset{\text{Lemma 3}}{=} \mathbf{V}(X^{C_{\sigma_{cr}},s_0},\mathbf{r}^{cr},\mathbf{t}) \\
&\overset{\text{Eqn. 3}}{=} \mathbf{V}(X^{C,\sigma_{cr},s_0},\mathbf{r},\mathbf{t}).
\end{aligned} \tag{6}$$

If we start with a CR σ'_{cr}, we can define the HR σ'_{hr} such that, for $\beta \in (S \times Act)^n$ with $n \in \mathbb{N}$, we have $\sigma'_{hr}(\beta, s) \overset{\text{def}}{=} \sigma'_{cr}(s, n)$, and then the result can be shown in the same way. □

From Lemma 4, we can conclude that the maximum is obtained by a scheduler in Σ_{CR}.

Corollary 2 *Given a CTMDP C, and reward structure* $\mathbf{r}$*, for all* $s_0 \in S$*, we have*

$$\mathbf{V}^{\max}(C, s_0, \mathbf{r}, \mathbf{t}) = \max_{\sigma\in\Sigma_{CR}} \mathbf{V}(X^{C,\sigma,s_0}, \mathbf{r}, \mathbf{t}).$$

Because of Corollary 2, we only have to show that Algorithm 1 computes the maximum over all $\sigma \in \Sigma_{CR}$ up to the specified precision.

We first show that, to compute the value of a given CTMDP for a given scheduler up to a required precision, it suffices to consider a limited number of steps in the embedded model.

Lemma 5 *Given a CTMDP* $C = (S, Act, \mathbf{P})$ *with a reward structure* $\mathbf{r} = (\mathbf{r}_c, \mathbf{r}_f)$*, a precision* $\varepsilon > 0$*, and k such that*

$$\sum_{n=0}^{k}\psi_{\mathbf{ut}}(n) > \mathbf{ut} - \frac{\varepsilon\mathbf{u}}{2\mathbf{r}_c^{\max}} \text{ and } \psi_{\mathbf{ut}}(k)\cdot\mathbf{r}_f^{\max} < \frac{\varepsilon}{2},$$

then for all schedulers $\sigma \in \Sigma_{CR}$*, and all* $s_0 \in S$*, we have*

$$\sum_{i=k+1}^{\infty}\left(\phi_{\mathbf{ut}}(i)\sum_{s\in S}\pi^{\mathrm{emb}(C),\sigma}(s_0,i,s)\mathbf{r}_f(s) + \psi_{\mathbf{ut}}(i)\sum_{s\in S}\pi^{\mathrm{emb}(C),\sigma}(s_0,i,s)\frac{\mathbf{r}_c(s)}{\mathbf{u}}\right) < \varepsilon.$$

Proof: Consider a CTMC $C' = (S', \mathbf{P}')$ with reward structure $\mathbf{r}' = (\mathbf{r}'_c, \mathbf{r}'_f)$, and

$$\mathbf{V}(C', s_0, \mathbf{r}', \mathbf{t}) = \underbrace{\mathbf{E}\left[\int_0^{\mathbf{t}} \mathbf{r}'_c(X_u^{C', s_0})\,\mathrm{d}u\right]}_{\text{accumulated}} + \underbrace{\mathbf{E}\left[\mathbf{r}'_f(X_{\mathbf{t}}^{C', s_0})\right]}_{\text{final}}$$

for any $s_0 \in S$. It is known [54, remark below Theorem 2] that, if

$$\sum_{n=0}^{k} \psi_{\mathbf{ut}}(n) > \mathbf{ut} - \frac{\varepsilon \mathbf{u}}{2\mathbf{r}'^{\max}_c},$$

then

$$\begin{aligned}
&\text{accumulated} \\
&\overset{\text{def}}{=} \sum_{n=k+1}^{\infty} \psi_{\mathbf{ut}}(i) \sum_{s \in S'} \pi^{\mathrm{emb}(C')}(s_0, i, s) \frac{\mathbf{r}'_c(s)}{\mathbf{u}} \\
&< \frac{\varepsilon}{2};
\end{aligned}$$

and if

$$\psi_{\mathbf{ut}}(k) \cdot \mathbf{r}'^{\max}_f < \frac{\varepsilon}{2},$$

then

$$\begin{aligned}
&\text{final} \\
&\overset{\text{def}}{=} \sum_{i=k+1}^{\infty} \phi_{\mathbf{ut}}(i) \sum_{s \in S'} \pi^{\mathrm{emb}(C')}(s_0, i, s) \cdot \mathbf{r}'_f(s) \\
&\leq \sum_{i=k+1}^{\infty} \phi_{\mathbf{ut}}(i) \sum_{s \in S'} \pi^{\mathrm{emb}(C')}(s_0, i, s) \cdot \mathbf{r}'^{\max}_f \\
&= \sum_{i=k+1}^{\infty} \phi_{\mathbf{ut}}(i) \mathbf{r}'^{\max}_f \\
&= \psi_{\mathbf{ut}}(k) \cdot \mathbf{r}'^{\max}_f \\
&< \frac{\varepsilon}{2}.
\end{aligned}$$

Thus,

$$\begin{aligned}
&\sum_{i=k+1}^{\infty} \left(\phi_{\mathbf{ut}}(i) \sum_{s \in S} \pi^{\mathrm{emb}(C')}(s_0, i, s) \mathbf{r}'_f(s) \right. \\
&\quad \left. + \psi_{\mathbf{ut}}(i) \sum_{s \in S} \pi^{\mathrm{emb}(C')}(s_0, i, s) \frac{\mathbf{r}'_c(s)}{\mathbf{u}} \right) \qquad (7) \\
&= \text{accumulated} + \text{final} \\
&< \varepsilon.
\end{aligned}$$

Now, with $\mathbf{r}' \overset{\text{def}}{=} (\mathbf{r}'_c, \mathbf{r}'_f)$ where $\mathbf{r}'_c(s, n) \overset{\text{def}}{=} \mathbf{r}_c(s)$, and $\mathbf{r}'_f(s, n) \overset{\text{def}}{=} \mathbf{r}_f(s)$, because $\mathbf{r}^{\max}_c = \mathbf{r}'^{\max}_c$ and $\mathbf{r}^{\max}_f = \mathbf{r}'^{\max}_f$ the following holds.

$$\begin{aligned}
&\sum_{i=k+1}^{\infty} \left(\phi_{\mathbf{ut}}(i) \sum_{s \in S} \pi^{\mathrm{emb}(C), \sigma}(s_0, i, s) \cdot \mathbf{r}_f(s) \right. \\
&\quad \left. + \psi_{\mathbf{ut}}(i) \sum_{s \in S} \pi^{\mathrm{emb}(C), \sigma}(s_0, i, s) \frac{\mathbf{r}_c(s)}{\mathbf{u}} \right) \\
&= \sum_{i=k+1}^{\infty} \left(\phi_{\mathbf{ut}}(i) \sum_{s \in S \times \mathbb{N}} \pi^{\mathrm{emb}(C)_\sigma}(s_0, i, s) \cdot \mathbf{r}'_f(s) \right. \\
&\quad \left. + \psi_{\mathbf{ut}}(i) \sum_{s \in S \times \mathbb{N}} \pi^{\mathrm{emb}(C)_\sigma}(s_0, i, s) \frac{\mathbf{r}'_c(s)}{\mathbf{u}} \right) \\
&< \varepsilon.
\end{aligned}$$

□

Algorithm 4: Compute value of $C = (S, Act, \mathbf{R})$, reward structure $(\mathbf{r}_c, \mathbf{r}_f)$ and CR σ up to precision ε.

let k s.t. $\sum_{n=0}^{k} \psi_{\mathbf{ut}}(n) > \mathbf{ut} - \varepsilon\mathbf{u}/(2\mathbf{r}^{\max}_c) \wedge \psi_{\mathbf{ut}}(k) \cdot \mathbf{r}^{\max}_f < \varepsilon/2$
$C' = (S, Act, \mathbf{P}) := \mathrm{emb}(C)$
forall $s \in S$ **do** $q_{k+1}(s) := 0$

forall $i = k, k-1, \ldots, 0$ **do**
  forall $s \in S$ **do**
    $m := \sum_{\alpha \in Act} \sigma(s, k)(\alpha) \sum_{s' \in S} \mathbf{P}(s, \alpha, s') q_{i+1}(s')$
    $q_i(s) := m + \phi_{\mathbf{ut}}(i) \cdot \mathbf{r}_f(s) + \psi_{\mathbf{ut}}(i) \cdot \mathbf{r}_c(s)/\mathbf{u}$
return q_0

We can show that Algorithm 4 computes the values of a CTMDP given a certain scheduler up to a required precision.

Lemma 6 *Consider a CTMDP $C = (S, Act, \mathbf{R})$, a time bound $\mathbf{t}$, a scheduler $\sigma \in \Sigma_{CR}$, and let q be the return value of Algorithm 4. Then $|q(s_0) - \mathbf{V}(X^{C, \sigma, s_0}, \mathbf{r}, \mathbf{t})| < \varepsilon$ for all $s_0 \in S$.*

Proof: Consider the values $q = q_0$ returned by Algorithm 4. It holds that

$$
\begin{aligned}
& q(s_0) \\
&= \phi_{\mathbf{ut}}(0) \cdot \mathbf{r}_f(s_0) + \psi_{\mathbf{ut}}(0)\frac{\mathbf{r}_c(s_0)}{\mathbf{u}} \\
&\quad + \sum_{\alpha_0 \in Act} \sigma(s_0, 0)(\alpha_0) \sum_{s_1 \in S} \mathbf{P}(s_0, \alpha_0, s_1) \\
&\qquad \cdot (\phi_{\mathbf{ut}}(1) \cdot \mathbf{r}_f(s_1) + \psi_{\mathbf{ut}}(1)\frac{\mathbf{r}_c(s_1)}{\mathbf{u}} \\
&\qquad\quad + \sum_{\alpha_1 \in Act} \sigma(s_1, 1)(\alpha_1) \sum_{s_2 \in S} \mathbf{P}(s_1, \alpha_1, s_2) \cdots \\
&= \phi_{\mathbf{ut}}(0) \cdot \mathbf{r}_f(s_0) + \psi_{\mathbf{ut}}(0)\frac{\mathbf{r}_c(s_0)}{\mathbf{u}} \\
&\quad + \sum_{\alpha_0 \in Act} \sigma(s_0, 0)(\alpha_0) \sum_{s_1 \in S} \mathbf{P}(s_0, \alpha_0, s_1)(\phi_{\mathbf{ut}}(1) \cdot \mathbf{r}_f(s_0) \\
&\qquad\qquad + \psi_{\mathbf{ut}}(1)\frac{\mathbf{r}_c(s_0)}{\mathbf{u}}) \\
&\quad + \sum_{\alpha_0 \in Act} \sigma(s_0, 0)(\alpha_0) \sum_{s_1 \in S} \mathbf{P}(s_0, \alpha_0, s_1) \sum_{\alpha_1 \in Act} \sigma(s_1, 1) \cdots \\
&\quad + \cdots \\
&\overset{\text{Cor. 1}}{=} \sum_{i=0}^{k} \left(\phi_{\mathbf{ut}}(i) \sum_{s \in S} \pi^{\mathrm{emb}(C),\sigma}(s_0, i, s) \mathbf{r}_f(s) \right. \\
&\quad \left. + \psi_{\mathbf{ut}}(i) \sum_{s \in S} \pi^{\mathrm{emb}(C),\sigma}(s_0, i, s) \frac{\mathbf{r}_c(s)}{\mathbf{u}} \right) \\
&= \mathbf{V}(\mathrm{emb}(C), s_0, \mathbf{r}, \mathbf{t}, \mathbf{u}) - \\
&\quad \sum_{i=k+1}^{\infty} \left(\phi_{\mathbf{ut}}(i) \sum_{s \in S} \pi^{\mathrm{emb}(C),\sigma}(s_0, i, s) \mathbf{r}_f(s) \right. \\
&\quad \left. + \psi_{\mathbf{ut}}(i) \sum_{s \in S} \pi^{\mathrm{emb}(C),\sigma}(s_0, i, s) \frac{\mathbf{r}_c(s)}{\mathbf{u}} \right) \\
&\overset{\text{Lemma 1}}{=} \mathbf{V}(\mathrm{emb}(C), s_0, \mathbf{r}, \mathbf{t}, \mathbf{u}) - \varepsilon'
\end{aligned}
\tag{8}
$$

for some ε' with $0 \leq \varepsilon' < \varepsilon$, and thus $|q(s_0) - \mathbf{V}(X^{C,\sigma,s_0}, \mathbf{r}, \mathbf{t})| < \varepsilon$. □

The value obtained from Algorithm 1 will be no smaller than the one obtained from applying Algorithm 4 on an arbitrary scheduler.

Lemma 7 *Consider a CTMDP $C = (S, Act, \mathbf{R})$, a time bound $\mathbf{t}$, and an arbitrary scheduler $\sigma \in \Sigma_{CR}$; let q be the return value of Algorithm 4, and let q' be the return value of Algorithm 1. Then $q(s_0) \leq q'(s_0)$ holds for all $s_0 \in S$.*

Proof: Let q_i be as given in Algorithm 4, and let q'_i be the corresponding vector of Algorithm 1. We show the lemma by backward induction on the program variable i.

Induction start: $i = k + 1$. Before the main loop at the lines 3, both algorithms assign $q_{k+1}(s) = q'_{k+1}(s) = 0$ for all $s \in S$.

Induction assumption: Assume it is $q_{i+1}(s) \leq q'_{i+1}(s)$ at the beginning of the main loops, that is before line 4.

Induction step: Consider m of Algorithm 4, and corresponding m' of Algorithm 1 after the assignment to this variable at line 6. We have

$$
\begin{aligned}
m &= \sum_{\alpha \in Act(s)} \sigma(s, k)(\alpha) \sum_{s' \in S} \mathbf{P}(s, \alpha, s') q_{i+1}(s') \\
&\leq \max_{\alpha \in Act(s)} \sum_{s' \in S} \mathbf{P}(s, \alpha, s') q_{i+1}(s') \\
&\overset{\text{Ass.}}{\leq} \max_{\alpha \in Act(s)} \sum_{s' \in S} \mathbf{P}(s, \alpha, s') q'_{i+1}(s') \\
&= m'.
\end{aligned}
$$

Because lines 7 are identical in both algorithms, also $q_i \leq q'_i$ at the end of the main loops. □

With these preparations, we can now prove the first part of Proposition 1.

Lemma 8 *Consider a CTMDP $C = (S, Act, \mathbf{R})$, a reward structure $\mathbf{r}$, a time bound $\mathbf{t}$, and let q' be the return value of Algorithm 1. Then, for all $s_0 \in S$, it is $|q'(s_0) - \mathbf{V}^{\max}(C, s_0, \mathbf{r}, \mathbf{t})| < \varepsilon$.*

Proof: Let $\sigma \in \Sigma_{CR}$ be such that

$$
\mathbf{V}(X^{C,\sigma,s_0}, \mathbf{r}, \mathbf{t}) = \mathbf{V}^{\max}(C, s_0, \mathbf{r}, \mathbf{t}). \tag{9}
$$

Then, because of Lemma 6, $|q(s_0) - \mathbf{V}(X^{C,\sigma,s_0}, \mathbf{r}, \mathbf{t})| < \varepsilon$ is true, where q is the return value of Algorithm 4. Because of Lemma 7, we know that $q(s_0) \leq q'(s_0)$. By adding the assignment

$$
\sigma'_{cd}(s, i) := \arg \max_{\alpha \in Act(s)} \sum_{s' \in S} \mathbf{P}(s, \alpha, s') q_{i+1}(s')
$$

into the inner loop of Algorithm 1 after Line 8, we can obtain a prefix of the scheduler $\sigma'_{cd} \in \Sigma_{CD}$. Consider

$\sigma'_{cr} \in \Sigma_{CR}$ such that $\sigma'_{cr}(s,i)(\alpha) = 1$ if $\sigma'_{cd}(s,i) = \alpha$, and $\sigma'_{cr}(s,i)(\alpha) = 0$ else. It can easily be shown that applying Algorithm 4 on σ'_{cr} also yields the value q'. Thus, again by Lemma 6, we have $|q'(s_0) - \mathbf{V}(X^{C,\sigma,s_0}, \mathbf{r}, \mathbf{t})| < \varepsilon$. □

We can now show that deterministic schedulers suffice, by using the fact that Algorithm 1 indeed maximises only over this class.

Lemma 9 *Let $C = (S, Act, \mathbf{R})$ be a CTMDP with reward structure $\mathbf{r} = (\mathbf{r}_c, \mathbf{r}_f)$. Then there exists $\sigma \in \Sigma_{CD}$ such that $\mathbf{V}^{\max}(C, s_0, \mathbf{r}, \mathbf{t}) = \mathbf{V}(X^{C,\sigma,s_0}, \mathbf{r}, \mathbf{t})$.*

Proof: Assume the lemma does not hold. Then, for each $\sigma \in \Sigma_{CD}$, there is a $\varepsilon > 0$ such that, for some state $s_0 \in S$, we have $|\mathbf{V}(X^{C,\sigma,s_0}, \mathbf{r}, \mathbf{t}) - \mathbf{V}^{\max}(C, s_0, \mathbf{r}, \mathbf{t})| \geq \varepsilon$. Now consider $\sigma'_{cd} \in \Sigma_{CD}$ obtained by Algorithm 1 as in the proof of Lemma 8 using the same required precision ε. By the correctness of Algorithm 1, we then have $|\mathbf{V}(X^{C,\sigma',s_0}, \mathbf{r}, \mathbf{t}) - \mathbf{V}^{\max}(C, s_0, \mathbf{r}, \mathbf{t})| < \varepsilon$. This result contradicts the assumption, because with the extension in the proof of Lemma 8, the algorithm also computes a CD which obtains this precision. □ The idea of using the fact that the optimising algorithm computes a scheduler of a more restricted class than the class considered was adapted from various proofs for similar problems in the discrete-time setting [68].

Proof of Proposition 2

Consider $(S, \mathbf{P}) = \mathcal{D} \stackrel{\text{def}}{=} \text{emb}(C)$, and $(S, \mathbf{P}') = \mathcal{D}' \stackrel{\text{def}}{=} \text{emb}(C')$. We define the HR $\sigma\colon ((\mathfrak{A} \times Act)^* \times \mathfrak{A}) \to Distr(Act)$ so that for $\beta = \mathfrak{z}_0 \alpha_{s_0} \mathfrak{z}_1 \alpha_{s_1} \ldots \alpha_{s_{n-1}} \mathfrak{z}_n$ we have

$$\sigma(\beta)(\alpha_s) \stackrel{\text{def}}{=} \Pr(X_n^{\mathcal{D},s_0} = s \mid X_n^{\mathcal{D},s_0} \wedge \bigwedge_{i=0}^{n-1} X_i^{\mathcal{D},s_0} = s_i).$$

By induction, for all $n \in \mathbb{N}$, and $\mathfrak{z} \in \mathfrak{A}$, we have

$$\Pr(X_n^{\mathcal{D},s_0} \in \mathfrak{z}) = \Pr(X_n^{\mathcal{D}',\sigma,\mathfrak{z}_0} = \mathfrak{z}).$$

Thus, using Definition 22, Definition 24, and the definition of the reward structures $\mathbf{r}, \mathbf{r}'$ it is

$$\mathbf{V}(\mathcal{D}, s_0, \mathbf{r}, \mathbf{t}, \mathbf{u}) \leq \mathbf{V}(\mathcal{D}'_\sigma, \mathfrak{z}_0, \mathbf{r}, \mathbf{t}, \mathbf{u});$$

and thus using Lemma 3 we obtain

$$\mathbf{V}(X^{C,s_0}, \mathbf{r}, \mathbf{t}) \leq \mathbf{V}(X^{C',\sigma,\mathfrak{z}_0}, \mathbf{r}', \mathbf{t}) \leq \mathbf{V}^{\max}(C', \mathfrak{z}_0, \mathbf{r}', \mathbf{t}).$$

Proof of Proposition 3

We only show that, by using a finer abstraction, the maximal bound cannot increase. The case for the minimal bound is likewise. We have to show

$$\mathbf{V}^{\max}(C, \mathfrak{z}_t, \mathbf{r}, \mathbf{t}) \geq \mathbf{V}^{\max}(C', \mathfrak{z}'_{t,j}, \mathbf{r}', \mathbf{t}). \tag{10}$$

We will first show that, for each $\varepsilon > 0$, it holds that

$$\mathbf{V}^{\max}(C, \mathfrak{z}_t, \mathbf{r}, \mathbf{t}) + \varepsilon \geq \mathbf{V}^{\max}(C', \mathfrak{z}'_{t,j}, \mathbf{r}', \mathbf{t}). \tag{11}$$

To do so, we show by a backward induction on Algorithm 1 that, for the value q_0 obtained for C, and the value q' obtained for C', we have $q \geq q'_0$, using a precision of ε. By the precision guarantee of the algorithm from Proposition 1, doing so in turn shows (11) for the given ε.

Induction start with $i = k + 1$. Before the main loop, at Line 3, both runs assign $q_{k+1}(\mathfrak{z}_t) = q'_{k+1}(\mathfrak{z}'_{t,j}) = 0$ for all $\mathfrak{z}_t \in \mathfrak{P}, \mathfrak{z}'_{t,j} \in \mathfrak{P}'$.

Induction assumption. Assume $q_{i+1}(\mathfrak{z}_t) \geq q'_{i+1}(\mathfrak{z}'_{t,j})$ holds for all $\mathfrak{z}_t \in \mathfrak{P}, \mathfrak{z}'_{t,j} \in \mathfrak{P}'$ with $\mathfrak{z}_t = \bigcup_{j=1}^{u} \mathfrak{z}'_{t,j}$ at the beginning of the main loops, before Line 5.

Induction step. Consider m for $\mathfrak{z}_t$ of C, and corresponding m' for $\mathfrak{z}'_{t,j}$ of C', after the assignment to this variable at Line 7. Let $(\hat{\alpha}', \alpha')$ be a maximising decision for some $\mathfrak{z}'_{t,j}$ at this line. By the definition of the abstraction, for $\mathfrak{z}_t$, we find a corresponding $(\hat{\alpha}, \alpha)$ such that

- $Dom(\hat{\alpha}') = Dom(\hat{\alpha})$, and
- for each $c \in Dom(\hat{\alpha}')$ we have $\hat{\alpha}'(c) = (\mathfrak{z}'_{v,l}, \mathbf{I})$ and $\hat{\alpha}(c) = (\mathfrak{z}_v, \mathbf{I})$ with $\mathfrak{z}'_{v,l} \subseteq \mathfrak{z}_v$.

Then we have

$$\begin{aligned}
m' &= \max_{(\hat{\alpha}',\alpha') \in Act(\mathfrak{z}'_{t,j})} \sum_{\mathfrak{z}'_{v,l} \in \mathfrak{P}'} \mathbf{P}(\mathfrak{z}_{t,j}, (\hat{\alpha}', \alpha'), \mathfrak{z}'_{v,l}) q_{i+1}(\mathfrak{z}'_{v,l}) \\
&= \sum_{\mathfrak{z}'_{v,l} \in \mathfrak{P}'} \mathbf{P}(\mathfrak{z}'_{t,j}, (\hat{\alpha}', \alpha'), \mathfrak{z}'_{v,l}) q_{i+1}(\mathfrak{z}'_{v,l}) \\
&\stackrel{\text{Ass.}}{\leq} \sum_{\mathfrak{z}_v \in \mathfrak{P}} \mathbf{P}(\mathfrak{z}_t, (\hat{\alpha}, \alpha), \mathfrak{z}_v) q_{i+1}(\mathfrak{z}_v) \\
&\leq \max_{(\hat{\alpha},\alpha) \in Act(\mathfrak{z}_t)} \sum_{\mathfrak{z}_v \in \mathfrak{P}} \mathbf{P}(\mathfrak{z}_t, (\hat{\alpha}, \alpha), \mathfrak{z}_v) q_{i+1}(\mathfrak{z}_v) \\
&= m.
\end{aligned}$$

The rewards for the refined abstraction cannot be larger than the ones for the coarser one. Thus, after Line 8, we still have $q_i \geq q'_i$ at the end of each iteration.

The validity of (11) then also shows that the inequality (10) holds for $\varepsilon = 0$: if $\mathbf{V}^{\max}(C, \mathfrak{d}_t, \mathbf{r}, \mathbf{t}) < \mathbf{V}^{\max}(C', \mathfrak{d}'_{t,j}, \mathbf{r}', \mathbf{t})$, then $\varepsilon' \stackrel{\text{def}}{=} \mathbf{V}^{\max}(C', \mathfrak{d}'_{t,j}, \mathbf{r}', \mathbf{t}) - \mathbf{V}^{\max}(C, \mathfrak{d}_t, \mathbf{r}, \mathbf{t})$ is positive. By subtracting $\mathbf{V}^{\max}(C, \mathfrak{d}_t, \mathbf{r}, \mathbf{t})$ from (11), we have

$$\varepsilon \geq \varepsilon'.$$

This equation must hold for all ε, for instance $\varepsilon'/2$. Thus, we would obtain a contradiction if it would not hold for $\varepsilon = 0$.

■

References

[1] Alessio Angius, András Horváth, and Verena Wolf. Approximate transient analysis of queuing networks by quasi product forms. In Alexander N. Dudin and Koen De Turck, editors, *Int'l Conf. on Analytical and Stochastic Modelling Techniques and Applications (ASMTA)*, volume 7984 of *Lecture Notes in Computer Science*, pages 22–36, Ghent, Belgium, July 2013. Springer-Verlag.

[2] Christel Baier, Ernst Moritz Hahn, Boudewijn R. Haverkort, Holger Hermanns, and Joost-Pieter Katoen. Model checking for performability. *Mathematical Structures in Computer Science*, 2012.

[3] Christel Baier, Boudewijn R. Haverkort, Holger Hermanns, and Joost-Pieter Katoen. Model-checking algorithms for continuous-time Markov chains. *IEEE Trans. on Software Engineering*, 29(6):524–541, 2003.

[4] Christel Baier, Holger Hermanns, Joost-Pieter Katoen, and Boudewijn R. Haverkort. Efficient computation of time-bounded reachability probabilities in uniform continuous-time Markov decision processes. *Theoretical Computer Science*, 345(1):2–26, 2005.

[5] Anne Benoit, Brigitte Plateau, and William J. Stewart. Memory-efficient Kronecker algorithms with applications to the modelling of parallel systems. *Future Generation Computer Systems*, 22(7):838–847, 2006.

[6] D. P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 2005.

[7] Luca Bortolussi and Jane Hillston. Fluid model checking. In *Int'l Conf. on Concurrency Theory (CONCUR)*, volume 7454 of *Lecture Notes in Computer Science*, pages 333–347. Springer-Verlag, 2012.

[8] Luca Bortolussi and Jane Hillston. Checking individual agent behaviours in Markov population models by fluid approximation. In *Int'l School on Formal Methods for the Design of Computer, Communication and Software Systems (SFM)*, pages 113–149, Bertinoro, Italy, June 2013.

[9] Amar Bouali and Robert de Simone. Symbolic bisimulation minimisation. In *Int'l Conf. on Computer Aided Verification (CAV)*, volume 663 of *Lecture Notes in Computer Science*, pages 96–108. Springer-Verlag, 1992.

[10] Tomás Brázdil, Vojtech Forejt, Jan Krcál, Jan Kretínský, and Antonín Kucera. Continuous-time stochastic games with time-bounded reachability. In *IARCS Annual Conf. on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 4 of *LIPIcs*, pages 61–72, 2009.

[11] Randal E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. on Computers*, 35(8):677–691, 1986.

[12] Peter Buchholz. Bounding reward measures of Markov models using the Markov decision processes. *Numerical Linear Algebra with Applications*, 18(6):919–930, 2011.

[13] Peter Buchholz. Finite horizon analysis of infinite CTMDPs. In *Int'l Conf. on Dependable Systems and Networks (DSN)*, pages 1–12. IEEE Computer Society Press, 2012.

[14] Peter Buchholz, Gianfranco Ciardo, Susanna Donatelli, and Peter Kemper. Complexity of memory-efficient Kronecker operations with applications to the solution of Markov models. *INFORMS Journal on Computing*, 12(3):203–222, 2000.

[15] Peter Buchholz, Ernst Moritz Hahn, Holger Hermanns, and Lijun Zhang. Model checking algorithms for CTMDPs. In *Int'l Conf. on Computer Aided Verification (CAV)*, volume 6806 of *Lecture Notes in Computer Science*, pages 225–242. Springer-Verlag, 2011.

[16] Peter Buchholz and Peter Kemper. Kronecker based matrix representations for large Markov models. In *Validation of Stochastic Systems*, volume 2925 of *Lecture Notes in Computer Science*, pages 256–295. Springer-Verlag, 2004.

[17] Benoît Caillaud, Benoît Delahaye, Kim G. Larsen, Axel Legay, Mikkel L. Pedersen, and Andrzej Wasowski. Compositional design methodology with constraint Markov chains. In *Int'l Conf. on Quantitative Evaluation of Systems (QEST)*, pages 123–132. IEEE Computer Society Press, 2010.

[18] Juan A. Carrasco. Transient analysis of some rewarded Markov models using randomization with quasistationarity detection. *IEEE Trans. on Computers*, 53(9):1106–1120, 2004.

[19] Juan A. Carrasco and Víctor Suñé. A numerical method for the evaluation of the distribution of cumulative reward till exit of a subset of transient states of a Markov reward model. *IEEE Trans. Dependable and Secure Computing*, 8(6):798–809, 2011.

[20] Gianfranco Ciardo, Andrew S. Miner, and Min Wan. Advanced features in SMART: the stochastic model checking analyzer for reliability and timing. *SIGMETRICS Performance Evaluation Review*, 36(4):58–63, 2009.

[21] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV version 2: An opensource tool for symbolic model checking. In *Int'l Conf. on Computer Aided Verification (CAV)*, volume 2404 of *Lecture Notes in Computer Science*, pages 359–364. Springer-Verlag, 2002.

[22] Lucia Cloth and Boudewijn R. Haverkort. Model checking for survivability. In *Int'l Conf. on Quantitative Evaluation of Systems (QEST)*, pages 145–154. IEEE Computer Society Press, 2005.

[23] Pepijn Crouzen, Ernst Moritz Hahn, Holger Hermanns, Abhishek Dhama, Oliver E. Theel, Ralf Wimmer, Bettina Braitling, and Bernd Becker. Bounded fairness for probabilistic distributed algorithms. In *Int'l Conf. on Application of Concurrency to System Design (ACSD)*, pages 89–97. IEEE Computer Society Press, 2011.

[24] Pedro R. D'Argenio, Bertrand Jeannet, Henrik Ejersbo Jensen, and Kim Guldstrand Larsen. Reachability analysis of probabilistic systems by successive refinements. In *Int'l Workshop on Process Algebra and Performance Modelling and Probabilistic Methods in Verification (PAPM-PROBMIV)*, volume 2165 of *Lecture Notes in Computer Science*, pages 39–56. Springer-Verlag, 2001.

[25] Tuğrul Dayar. *Analyzing Markov chains using Kronecker products. Theory and applications*. Springer-Verlag, 2012.

[26] Luca de Alfaro and Pritam Roy. Magnifying-lens abstraction for Markov decision processes. In *Int'l Conf. on Computer Aided Verification (CAV)*, volume 4590 of *Lecture Notes in Computer Science*, pages 325–338. Springer-Verlag, 2007.

[27] Daniel D. Deavours, Graham Clark, Tod Courtney, David Daly, Salem Derisavi, Jay M. Doyle, William H. Sanders, and Patrick G. Webster. The Möbius framework and its implementation. *IEEE Trans. on Software Engineering*, 28(10):956–969, 2002.

[28] Salem Derisavi. A symbolic algorithm for optimal Markov chain lumping. In *Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 4424 of *Lecture Notes in Computer Science*, pages 139–154. Springer-Verlag, 2007.

[29] Felipe Martins dos Santos, Leliane Nunes de Barros, and Mijail Gamarra Holguin. Stochastic bisimulation for MDPs using reachability analysis. In *Brazilian Conf. on Intelligent Systems (BRACIS)*, pages 213–218. IEEE, October 2013.

[30] Paulo Fernandes, Brigitte Plateau, and William J. Stewart. Efficient descriptor-vector multiplications in stochastic automata networks. *Journal of the ACM*, 45(3):381–414, 1998.

[31] Bennett L. Fox and Peter W. Glynn. Computing Poisson Probabilities. *Communications of the ACM*, 31(4):440–445, 1988.

[32] Swapna S. Gokhale, Michael R. Lyu, and Kishor S. Trivedi. Analysis of software fault removal policies using a non-homogeneous continuous time Markov chain. *Software Quality Journal*, 12(3):211–230, 2004.

[33] Winfried K. Grassmann. Finding Transient Solutions in Markovian Event Systems Through Randomization. In *Int'l Workshop on the Numerical Solution of Markov Chains (NSMC)*, pages 357–371, 1991.

[34] D. Gross and D. Miller. The randomization technique as a modeling tool and solution procedure for transient Markov processes. *Operations Research*, 32(2):926–944, 1984.

[35] Arnd Hartmanns, Pascal Berrang, and Holger Hermanns. A comparative analysis of decentralized power grid stabilization strategies. In *Winter Simulation Conference (WSC)*, 2012.

[36] Boudewijn R. Haverkort. *Performance of computer communication systems – a model-based approach*. Wiley, 1998.

[37] Boudewijn R. Haverkort, Holger Hermanns, and Joost-Pieter Katoen. On the use of model checking techniques for dependability evaluation. In *IEEE Symposium on Reliable Distributed Systems (SRDS)*, pages 228–237. IEEE Computer Society Press, 2000.

[38] Thomas A. Henzinger, Barbara Jobstmann, and Verena Wolf. Formalisms for specifying Markovian population models. *Int'l Journal of Foundations of Computer Science*, 22(4):823–841, 2011.

[39] Thomas A. Henzinger, Maria Mateescu, and Verena Wolf. Sliding window abstraction for infinite Markov chains. In *Int'l Conf. on Computer Aided Verification (CAV)*, pages 337–352, 2009.

[40] Thomas A. Henzinger, Linar Mikeev, Maria Mateescu, and Verena Wolf. Hybrid numerical solution of the chemical master equation. In Paola Quaglia, editor, *Int'l Conf. on Computational Methods in Systems Biology (CMSB)*, pages 55–65. ACM Press, 2010.

[41] Holger Hermanns, Marta Z. Kwiatkowska, Gethin Norman, David Parker, and Markus Siegle. On the use of MTBDDs for performability analysis and verification of stochastic systems. *Journal of Logic and Algebraic Programming*, 56(1–2):23–67, 2003.

[42] Holger Hermanns, Björn Wachter, and Lijun Zhang. Probabilistic CEGAR. In *Int'l Conf. on Computer Aided Verification (CAV)*, volume 5123 of *Lecture Notes in Computer Science*, pages 162–175. Springer-Verlag, 2008.

[43] Jane Hillston, Mirco Tribastone, and Stephen Gilmore. Stochastic process algebras: From individuals to populations. *Computer Journal*, 55(7):866–881, 2012.

[44] Ronald A. Howard. *Dynamic Programming and Markov Processes*. John Wiley and Sons, Inc., 1960.

[45] A. Jensen. Markov chains as an aid in the study of Markov processes. *Skandinavisk Aktuarietidsskrift*, 36:87–91, 1953.

[46] Joost-Pieter Katoen, Daniel Klink, Martin Leucker, and Verena Wolf. Three-valued abstraction for continuous-time Markov chains. In *Int'l Conf. on Computer Aided Verification (CAV)*, volume 4590 of *Lecture Notes in Computer Science*, pages 311–324. Springer-Verlag, 2007.

[47] Joost-Pieter Katoen, Ivan S. Zapreev, Ernst Moritz Hahn, Holger Hermanns, and David N. Jansen. The ins and outs of the probabilistic model checker MRMC. *Performance Evaluation*, 68(2):90–104, 2011.

[48] Mark Kattenbelt, Marta Z. Kwiatkowska, Gethin Norman, and David Parker. A game-based abstraction-refinement framework for Markov decision processes. *Formal Methods in System Design*, 36(3):246–280, 2010.

[49] J. Kemeny, J. Snell, and A. Knapp. *Denumerable Markov Chains*. D. Van Nostrand Company, 1966.

[50] Daniel Klink. *Three-Valued Abstraction for Stochastic Systems*. PhD thesis, RWTH Aachen, Germany, 2010.

[51] Donald E. Knuth. *The Art of Computer Programming, Volume 4, Fascicle 1: Bitwise Tricks & Techniques; Binary Decision Diagrams*. Addison-Wesley Professional, 12th edition, 2009.

[52] Boris Köpf and David A. Basin. Automatically deriving information-theoretic bounds for adaptive side-channel attacks. *Journal of Computer Security*, 19(1):1–31, 2011.

[53] Igor Kozine and Lev V. Utkin. Interval-valued finite Markov chains. *Reliable Computing*, 8(2):97–113, 2002.

[54] M. Kwiatkowska, G. Norman, and A. Pacheco. Model checking expected time and expected reward formulae with random time bounds. *Computers & Mathematics with Applications*, 51:305–316, 2006.

[55] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *Int'l Conf. on Computer Aided Verification (CAV)*, volume 6806 of *Lecture Notes in Computer Science*, pages 585–591. Springer-Verlag, 2011.

[56] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. Probabilistic symbolic model checking with PRISM: a hybrid approach. *Software Tools for Technology Transfer*, 6(2):128–142, 2004.

[57] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. Stochastic model checking. In *Int'l School on Formal Methods for the Design of Computer, Communication and Software Systems (SFM)*, volume 4486 of *Lecture Notes in Computer Science*, pages 220–270. Springer-Verlag, 2007.

[58] Yung-Te Lai, Massoud Pedram, and Sarma B. K. Vrudhula. EVBDD-based algorithms for integer linear programming, spectral transformation, and function decomposition. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 13(8):959–975, 1994.

[59] Yung-Te Lai, Massoud Pedram, and Sarma B. K. Vrudhula. Formal verification using edge-valued binary decision diagrams. *IEEE Trans. on Computers*, 45(2):247–255, 1996.

[60] Kai Lampka, Markus Siegle, Jörn Ossowski, and Christel Baier. Partially-shared zero-suppressed multi-terminal BDDs: concept, algorithms and applications. *Formal Methods in System Design*, 36(3):198–222, 2010.

[61] Steven A. Lippman. Countable-state, continuous-time dynamic programming with structure. *Operations Research*, 24(3):477–490, 1976.

[62] Mieke Massink, Diego Latella, Andrea Bracciali, Michael D. Harrison, and Jane Hillston. Scalable context-dependent analysis of emergency egress models. *Formal Aspects of Computing*, 24(2):267–302, 2012.

[63] Mieke Massink, Diego Latella, Andrea Bracciali, and Jane Hillston. Modelling non-linear crowd dynamics in Bio-PEPA. In *Int'l Conf. on Fundamental Approaches to Software Engineering (FASE)*, volume 6603 of *Lecture Notes in Computer Science*, pages 96–110. Springer-Verlag, 2011.

[64] M Mateescu, V Wolf, F Didier, and T A Henzinger. Fast adaptive uniformisation of the chemical master equation. *IET Systems Biology*, 4(6):441–452, 2010.

[65] Brian Munsky and Mustafa Khammash. The finite state projection algorithm for the solution of the chemical master equation. *Journal of Chemical Physics*, 124(044104), 2006.

[66] Jörn Ossowski and Christel Baier. Symbolic reasoning with weighted and normalized decision diagrams. *Electronic Notes in Theoretical Computer Science*, 151(1):39–56, 2006.

[67] D. Parker. *Implementation of Symbolic Model Checking for Probabilistic Systems*. PhD thesis, University of Birmingham, UK, 2002.

[68] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley and Sons, 1994.

[69] Markus Rabe and Sven Schewe. Optimal time-abstract schedulers for CTMDPs and Markov games. In *Int'l Workshop on Quantitative Aspects of Programming Languages (QAPL)*, volume 28 of *EPTCS*, pages 144–158, 2010.

[70] Markus N. Rabe and Sven Schewe. Finite optimal control for time-bounded reachability in CTMDPs and continuous-time Markov games. *Acta Informatica*, 48(5-6):291–315, 2011.

[71] W.H. Sanders and John F. Meyer. Performability evaluation of distributed systems using stochastic activity networks. In *Int'l Workshop on Petri Nets and Performance Models (PNPM)*, pages 111–120, Madison, WI, USA, 1987. IEEE Computer Society Press.

[72] William H. Sanders. Assuring the trustworthiness of the smarter electric grid. In *IEEE Int'l Symp. on Network Computing and Applications (NCA)*. IEEE Computer Society Press, 2012.

[73] Michael J. A. Smith. Compositional abstraction of PEPA models for transient analysis. In *European Workshop on Performance Engineering (EPEW)*, volume 6342 of *Lecture Notes in Computer Science*, pages 252–267. Springer-Verlag, 2010.

[74] William J. Stewart. *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press, 1994.

[75] R. Strauch. Negative dynamic programming. *Annals of Mathematical Statistics*, 37:871–890, 1966.

[76] Kishor S. Trivedi. *Probability and statistics with reliability, queuing, and computer science applications*. Prentice Hall, 1982.

[77] Kishor S. Trivedi, Andrew L. Reibman, and Roger Smith. Transient analysis of Markov and Markov reward models. In *Int'l Workshop on Computer Performance and Reliability (MCPR)*, pages 535–545, Rome, Italy, 1987.

[78] Aad P. A. van Moorsel and William H. Sanders. Adaptive Uniformization. *Communications in Statistics – Stochastic Models*, 10(3):619–647, 1994.

[79] Min Wan, Gianfranco Ciardo, and Andrew S. Miner. Approximate steady-state analysis of large Markov models based on the structure of their decision diagram encoding. *Performance Evaluation*, 68(5):463–486, 2011.

[80] Ingo Wegener. *Branching Programs and Binary Decision Diagrams – Theory and Applications*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, USA, 2000.

[81] Ralf Wimmer, Bettina Braitling, Bernd Becker, Ernst Moritz Hahn, Pepijn Crouzen, Holger Hermanns, Abhishek Dhama, and Oliver E. Theel. Symblicit calculation of long-run averages for concurrent probabilistic systems. In *Int'l Conf. on Quantitative Evaluation of Systems (QEST)*, pages 27–36. IEEE Computer Society Press, 2010.

[82] Ralf Wimmer, Salem Derisavi, and Holger Hermanns. Symbolic partition refinement with automatic balancing of time and space. *Performance Evaluation*, 67(9):815–835, 2010.

[83] Ralf Wimmer, Marc Herbstritt, Holger Hermanns, Kelley Strampp, and Bernd Becker. Sigref – A symbolic bisimulation tool box. In *Int'l Symposium on Automated Technology for Verification and Analysis (ATVA)*, volume 4218 of *Lecture Notes in Computer Science*, pages 477–492. Springer-Verlag, 2006.

[84] Shouhuai Xu, Wenlian Lu, and Zhenxin Zhan. A stochastic model of multivirus dynamics. *IEEE Trans. on Dependable and Secure Computing*, 9(1):30–45, 2012.

[85] Y. Yuan and J. Allen L.˙ Stochastic models for virus and immune system dynamics. *Mathematical Biosciences*, 234(2):84–94, December 2011.

[86] Gianguglielmo Zehender, Erika Ebranati, Flavia Bernini, Alessandra Lo Presti, Giovanni Rezza, Mauro Delogu, Massimo Galli, and Massimo Ciccozzi. Phylogeography and epidemiological history of West Nile virus genotype 1a in Europe and the Mediterranean basin. *Infection, Genetics and Evolution*, 11(3):646–653, April 2011.

[87] Lijun Zhang and Martin R. Neuhäußer. Model checking interactive Markov chains. In *Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 6015 of *Lecture Notes in Computer Science*, pages 53–68. Springer-Verlag, 2010.

[88] Yang Zhao and Gianfranco Ciardo. Tackling truncation errors in CSL model checking through bounding semantics. In *European Workshop on Performance Engineering (EPEW)*, volume 8168 of *Lecture Notes in Computer Science*, pages 58–73. Springer-Verlag, 2013.

Ernst Moritz Hahn received his Ph.D. from Saarland University in 2013. He wrote his doctoral thesis at the chair of Dependable Systems and Software, advised by Prof. Dr.-Ing. Holger Hermanns. After his dissertation, he became research assistant at the group of automated verification at the University of Oxford in the VERIWARE project lead by Marta Kwiatkowska. Ernst Moritz Hahn is currently associate professor in the Institute of Software Chinese Academy of Sciences. His research area is in probabilistic verification, such as in the analysis of very large Markov chains, probabilistic hybrid systems, and parametric Markov models.

Holger Hermanns is a full professor at the Department of Computer Science at Saarland University, Saarbrücken, Germany, holding the chair for Dependable Systems and Software. His research interests include compositional modeling and verification of concurrent systems, resource-aware embedded systems, and performance and dependability evaluation of critical infrastructure. In these areas, Holger Hermanns has authored or co-authored more than 150 peer-reviewed scientific papers (ha-index 92). He serves on the steering committees of ETAPS, TACAS, and QEST. He is Member of Academia Europaea.

Ralf Wimmer received his diploma with distinction in computer science from the Albert-Ludwigs-Universität Freiburg, Germany in 2004. Afterwards, he worked as a Ph.D. student at the Chair of Computer Architecture at the same university, advised by Prof. Dr. Bernd Becker. He obtained his Ph.D. degree with distinction in 2011 for his thesis on symbolic methods for probabilistic verification. Since then, he is continuing his work as a research assistant and leader of the verification group at the Chair of Computer Architecture. His research focus is on symbolic methods and solver technologies, and their application for the verification of digital and stochastic systems.

Bernd Becker is a Full Professor (Chair of Computer Architecture) at the Faculty of Engineering, University of Freiburg, Germany. Prior to joining the University of Freiburg in 1995, he was with J. W. Goethe-University Frankfurt as an associate professor for complexity theory and efficient algorithms. His research activities include design, test, and verification methods for embedded systems and nanoelectronic circuitry. He is a Co-Speaker of the DFG Transregional Collaborative Research Center "Automatic Analysis and Verification of Complex Systems (AVACS)," and a Director of the Centre for Security and Society, University of Freiburg. He is a fellow of IEEE, and Member of Academia Europaea.