Faster and Symbolic CTMC Model Checking*

Joost-Pieter Katoen[a], Marta Kwiatkowska[b],
Gethin Norman[b] and David Parker[b]

[a] *Formal Methods and Tools Group, Faculty of Computer Science*
University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands

[b] *School of Computer Science, University of Birmingham*
Edgbaston, Birmingham B15 2TT, United Kingdom

Abstract. This paper reports on the implementation and the experiments with symbolic model checking of continuous-time Markov chains using multi-terminal binary decision diagrams (MTBDDs). Properties are expressed in Continuous Stochastic Logic (CSL) [7] which includes the means to express both transient and steady-state performance measures. We show that all CSL operators can be treated using standard operations on MTBDDs, thus allowing a rather straightforward implementation of symbolic CSL model checking on existing MTBDD-based platforms such as the verifier PRISM. The main result of the paper is an improvement of $\mathcal{O}(N)$ in the time complexity of checking time-bounded until-formulas, where N is the number of states in the CTMC under consideration. This result yields a drastic speed-up in the verification time of model checking CTMCs, both in the symbolic and non-symbolic case.

1 Introduction

In model-based performance and dependability evaluation, techniques such as stochastic Petri nets, stochastic process algebras, stochastic activity networks, and queueing networks are used to specify the system behaviour at a high level of abstraction. Most of these techniques assume a continuous-time Markov chain (CTMC) as underlying stochastic process. While the analysis of CTMCs focuses mostly on transient-state and steady-state (i.e. long run) characteristics, the specification and analysis of path measures is a subject of growing interest [25].

The temporal logic CSL (Continuous Stochastic Logic) developed originally by Aziz *et al.* [2,3] and extended by Baier *et al.* [7] provides a powerful means to specify path-based as well as traditional state-based measures on CTMCs in a concise, flexible and unambiguous way. CSL is based on the well-known branching-time temporal logic CTL (Computation Tree Logic [11]) and PCTL (Probabilistic CTL [17]); a steady-state operator, a time-bounded until, and a probabilistic (path) operator constitute its main ingredients. It allows one to state, for example, that the probability of reaching a certain set of goal-states within a specified real-valued time bound, provided that all paths to these states obey certain properties, is at least/at most some probability value.

* Partly supported by EPSRC grants GR/M04617, GR/M13046 and GR/N31573.

L. de Alfaro and S. Gilmore (Eds.), 1st Joint Int. Workshop on Process Algebra and Performance Modelling and Probabilistic Methods in Verification, volume 2165 of LNCS, pages 23–38, 2001.

Verification of a given finite-state CTMC against a CSL formula is performed using model checking. The model checking problem for CSL is decidable for rational time bounds [2,3]. Approximate CSL model-checking algorithms have been studied in [7] where the satisfaction of time-bounded until formulas is shown to be based on solving a (recursive) Volterra equation system. More recently, Baier *et al.* [6] reduced verifying time-bounded until formulas to the problem of computing transient-state probabilities for CTMCs. This significant result employs a formula-dependent transformation of the CTMC and – more importantly – allows one to adopt efficient techniques like *uniformisation* [16,23] for verifying time-bounded until-formulas. This paper builds upon this earlier work, and considers two issues: improving the time and the space efficiency of model checking CTMCs against CSL formulas based on transient analysis.

Faster CSL model checking. Verifying time-bounded until formulas using transient analysis of CTMCs [6] suggests that uniformisation should be applied to each individual state separately. This results in a worst case time complexity of $\mathcal{O}(M{\cdot}N)$, where N is the number of states in the CTMC and M the number of transitions. The main result of this paper is an improvement of $\mathcal{O}(N)$ in the time complexity of checking time-bounded until formulas. Inspired by PCTL model checking, the basic idea underlying this efficiency improvement is to carry out the uniformisation for all states at once. Our experiments show that this result yields a drastic speed-up in the verification time of model checking CTMCs.

Symbolic CSL model checking. To combat the infamous state-space explosion problem we investigate representing the state space by multi-terminal binary decision diagrams (MTBDDs [12], also called algebraic decision diagrams [4]). MTBDDs are variants of BDDs that can efficiently deal with real matrices; they allow arbitrary real numbers in the terminal nodes instead of just 0 and 1. We show that CSL model checking can be treated using standard operations on MTBDDs, thus generalising the result for PCTL [5,18] to the continuous-time setting. This basically follows from the fact that CSL model checking amounts to the analysis of either the *embedded* discrete-time Markov chain (DTMC) – in the case of untimed until formulas and steady-state formulas – or the *uniformised* DTMC – in the case of time-bounded until – of the CTMC under consideration. This reduces to graph analysis and iterative matrix-vector multiplication which can be implemented with standard MTBDD operations. Variants of MTBDDs tailored to numerical integration [7] are not needed. This paper reports on the implementation of symbolic CSL model checking as part of PRISM[1] (PRobabilistIc Symbolic Model checker), a prototype tool for the symbolic verification of Markov decision processes (MDPs) with DTMCs as a subset thereof.

Organisation of the paper. Section 2 briefly recalls PCTL model checking. Section 3 introduces CTMCs and CSL. Handling time-bounded until is covered in Section 4. Section 5 discusses CSL model checking with MTBDDs. Section 6 presents our empirical results. Section 7 concludes the paper.

[1] `www.cs.bham.ac.uk/~dxp/prism`

2 The discrete-time setting

DTMCs. Let AP be a fixed, finite set of atomic propositions. A (labelled) DTMC $\mathcal{D}$ is a tuple $(S, \mathbf{P}, L)$ where S is a finite set of *states*, $\mathbf{P} : S \times S \rightarrow [0,1]$ is a *probability matrix* such that $\sum_{s' \in S} \mathbf{P}(s,s') = 1$ for all $s \in S$, and $L : S \rightarrow 2^{AP}$ is a *labelling* function which assigns to each state $s \in S$ the set $L(s)$ of atomic propositions that are valid in s. A path through a DTMC is a sequence[2] of states $\sigma = s_0\, s_1\, s_2 \ldots$ with $\mathbf{P}(s_i, s_{i+1}) > 0$ for all i. Let $Path^{\mathcal{D}}$ denote the set of all paths in $\mathcal{D}$. $\sigma[i]$ denotes the $(i{+}1)$th state of σ, i.e. $\sigma[i] = s_{i+1}$. Let Pr_s denote the unique probability measure on sets of paths that start in state s [22].

PCTL. Let $a \in AP$, $p \in [0,1]$, k be a natural (or ∞) and $\bowtie \in \{\leqslant, \geqslant\}$. The syntax of PCTL is:

$$\Phi ::= \mathrm{tt} \;\Big|\; a \;\Big|\; \Phi \wedge \Phi \;\Big|\; \neg \Phi \;\Big|\; \mathcal{P}_{\bowtie p}(\Phi\, \mathcal{U}^{\leqslant k}\, \Phi)$$

The other boolean connectives are derived in the usual way. For the sake of simplicity, we do not consider the next state operator in this paper. The standard (i.e. unbounded) until formula is obtained by taking k equal to ∞, i.e. $\Phi\, \mathcal{U}\, \Psi = \Phi\, \mathcal{U}^{\leqslant \infty}\, \Psi$. The semantics of PCTL is defined by [17]:

$$\begin{array}{ll} s \models \mathrm{tt} & \text{for all } s \in S \\ s \models a & \text{iff } a \in L(s) \\ s \models \neg\Phi & \text{iff } s \not\models \Phi \end{array} \qquad \begin{array}{ll} s \models \Phi \wedge \Psi & \text{iff } s \models \Phi \wedge s \models \Psi \\ s \models \mathcal{P}_{\bowtie p}(\Phi\, \mathcal{U}^{\leqslant k}\, \Psi) & \text{iff } \mathit{Prob}^{\mathcal{D}}(s, \Phi\, \mathcal{U}^{\leqslant k}\, \Psi) \bowtie p \end{array}$$

$\mathcal{P}_{\bowtie p}(\Phi\, \mathcal{U}^{\leqslant k}\, \Psi)$ asserts that the probability measure of the paths that start in s and that satisfy $\Phi\, \mathcal{U}^{\leqslant k}\, \Psi$ meets the bound $\bowtie p$. Here,

$$\mathit{Prob}^{\mathcal{D}}(s, \Phi\, \mathcal{U}^{\leqslant k}\, \Psi) = \mathrm{Pr}_s\{\, \sigma \in \mathit{Path}^{\mathcal{D}} \mid \sigma \models \Phi\, \mathcal{U}^{\leqslant k}\, \Psi \,\}$$

Formula $\Phi\, \mathcal{U}^{\leqslant k}\, \Psi$ asserts that Ψ will be satisfied within k steps and that all preceding states satisfy Φ, i.e.:

$$\sigma \models \Phi\, \mathcal{U}^{\leqslant k}\, \Psi \;\text{ iff }\; \exists j \leqslant k.\; (\sigma[j] \models \Psi \wedge \forall i < j.\, \sigma[i] \models \Phi)$$

Model checking PCTL. PCTL model checking [17] is carried out in the same way as verifying CTL [11] by recursively computing the set $\mathit{Sat}(\Phi) = \{\, s \in S \mid s \models \Phi \,\}$. Checking bounded until formulas amounts to computing the least solution of the following set of equations: $\mathit{Prob}^{\mathcal{D}}(s, \Phi\, \mathcal{U}^{\leqslant k}\, \Psi)$ equals 1 if $s \in \mathit{Sat}(\Psi)$,

$$\mathit{Prob}^{\mathcal{D}}(s, \Phi\, \mathcal{U}^{\leqslant k}\, \Psi) = \sum_{s' \in S} \mathbf{P}(s,s') \cdot \mathit{Prob}^{\mathcal{D}}(s', \Phi\, \mathcal{U}^{\leqslant k-1}\, \Psi) \qquad (1)$$

if $s \in \mathit{Sat}(\Phi \wedge \neg\Psi)$ and $k > 0$, and equals 0 otherwise. For DTMC $\mathcal{D} = (S, \mathbf{P}, L)$ and PCTL formula Φ, let DTMC $\mathcal{D}[\Phi] = (S, \mathbf{P}', L)$ where if $s \not\models \Phi$, then $\mathbf{P}'(s,s') = \mathbf{P}(s,s')$ for all $s' \in S$, and if $s \models \Phi$, then $\mathbf{P}'(s,s) = 1$ and $\mathbf{P}'(s,s') = 0$

[2] In this paper, we do not dwell upon distinguishing finite and infinite paths.

for all $s' \neq s$. We have $\mathcal{D}[\Phi][\Psi] = \mathcal{D}[\Phi \vee \Psi]$. Let $\pi^{\mathcal{D}}(s,k)(s')$ denote the probability of being in state s' after k steps in DTMC $\mathcal{D}$ when starting in s, i.e. $\pi^{\mathcal{D}}(s,k)(s') = \Pr_s\{\, \sigma \in \mathit{Path}^{\mathcal{D}} \mid \sigma[k] = s' \,\}$.

Proposition 1. *For DTMC $\mathcal{D}$: $\mathit{Prob}^{\mathcal{D}}(s, \Phi\,\mathcal{U}^{\leqslant k}\,\Psi) = \displaystyle\sum_{s' \models \Psi} \pi^{\mathcal{D}[\neg\Phi\vee\Psi]}(s,k)(s')$.*

Note that $\mathcal{D}[\neg\Phi\vee\Psi] = \mathcal{D}[\neg(\Phi\wedge\Psi)][\Psi]$, i.e. all $\neg(\Phi\wedge\Psi)$-states and all Ψ-states in $\mathcal{D}$ are made absorbing[3]. The former is correct since $\Phi\,\mathcal{U}^{\leqslant k}\,\Psi$ is violated as soon as some state is visited that neither satisfies Φ nor Ψ. The latter is correct since, once a Ψ-state in $\mathcal{D}$ has been reached (along a Φ-path) in at most k steps, then $\Phi\,\mathcal{U}^{\leqslant k}\,\Psi$ holds, regardless of which states will be visited later on.

Model checking $\mathcal{U}^{\leqslant k}$ for all states thus amounts to computing $(\mathbf{P}^{\mathcal{D}[\neg\Phi\vee\Psi]})^k \cdot \underline{\iota}_\Psi$, where $\underline{\iota}_\Psi$ characterises $\mathit{Sat}(\Psi)$, i.e. $\iota_\Psi(s) = 1$ if $s \models \Psi$, and 0 otherwise. As iterative squaring is not attractive for stochastic matrices due to fill in [28], the product is typically computed in an iterative fashion: $\mathbf{P}\cdot(\ldots(\mathbf{P}\cdot\underline{\iota}_\Psi)))$.

3 The continuous-time setting

CTMCs. A (labelled) CTMC $\mathcal{C}$ is a tuple $(S, \mathbf{R}, L)$ where S and L are as for DTMCs, and $\mathbf{R} : S \times S \to \mathbb{R}_{\geqslant 0}$ is the *rate matrix.* (We adopt the same conventions as in [6,7], i.e. we do allow self-loops.) The exit rate $E(s) = \sum_{s' \in S} \mathbf{R}(s,s')$ denotes that the probability of taking a transition from s within t time units equals $1 - e^{-E(s)\cdot t}$. If $\mathbf{R}(s,s') > 0$ for more than one state s', a *race* between the outgoing transitions from s exists. That is, the probability $\mathbf{P}(s,s')$ of moving from s to s' in a single step equals the probability that the delay of going from s to s' "finishes before" the delays of any other outgoing transition from s.

Definition 1. *For CTMC $\mathcal{C} = (S, \mathbf{R}, L)$, the* embedded *DTMC is given by $\mathit{emb}(\mathcal{C}) = (S, \mathbf{P}, L)$, where $\mathbf{P}(s,s') = \mathbf{R}(s,s')/E(s)$ if $E(s) > 0$, and $\mathbf{P}(s,s) = 1$ and $\mathbf{P}(s,s') = 0$ for $s \neq s'$ if $E(s) = 0$.*

A path through a CTMC is an alternating sequence $\sigma = s_0\, t_0\, s_1\, t_1\, s_2 \ldots$ with $\mathbf{R}(s_i, s_{i+1}) > 0$ and $t_i \in \mathbb{R}_{>0}$ for all i. The time stamps t_i denote the amount of time spent in state s_i. Let $\mathit{Path}^{\mathcal{C}}$ denote the set of paths through $\mathcal{C}$. $\sigma@t$ denotes the state of σ occupied at time t, i.e. $\sigma@t = \sigma[i]$ with i the smallest index such that $t \leqslant \sum_{j=0}^{i} t_j$. Let $\Pr_s$ denote the unique probability measure on sets of paths that start in s [7].

CSL. Let a, p and $\bowtie$ be as before and $t \in \mathbb{R}_{\geqslant 0}$ (or ∞). The syntax of CSL is:

$$\Phi ::= \mathrm{tt} \;\Big|\; a \;\Big|\; \Phi \wedge \Phi \;\Big|\; \neg\Phi \;\Big|\; \mathcal{S}_{\bowtie p}(\Phi) \;\Big|\; \mathcal{P}_{\bowtie p}(\Phi\,\mathcal{U}^{\leqslant t}\,\Phi)$$

$\mathcal{S}_{\bowtie p}(\Phi)$ asserts that the steady-state probability for a Φ-state meets the bound $\bowtie p$. The semantics of CSL for the boolean operators is identical to that for

[3] That is, the only transitions available in these states are self-loops.

PCTL. For the remaining state formulas [7]:

$$\begin{array}{ll} s \models \mathcal{S}_{\bowtie p}(\Phi) & \text{iff } \lim_{t \to \infty} \Pr_s\{\, \sigma \in \mathit{Path}^{\mathcal{C}} \mid \sigma@t \models \Phi \,\} \bowtie p \\ s \models \mathcal{P}_{\bowtie p}(\Phi \, \mathcal{U}^{\leqslant t} \, \Phi) & \text{iff } \mathit{Prob}^{\mathcal{C}}(s, \Phi \, \mathcal{U}^{\leqslant t} \, \Phi) \bowtie p \end{array}$$

The limit in the first equation always exists as $\mathcal{C}$ contains finitely many states [28]. $\mathit{Prob}^{\mathcal{C}}(\cdot)$ is defined in a similar way as for PCTL:

$$\mathit{Prob}^{\mathcal{C}}(s, \Phi \, \mathcal{U}^{\leqslant t} \, \Phi) = \Pr_s\{\, \sigma \in \mathit{Path}^{\mathcal{C}} \mid \sigma \models \Phi \, \mathcal{U}^{\leqslant t} \, \Phi \,\}\,.$$

The operator $\mathcal{U}^{\leqslant t}$ is the real-time variant of the PCTL operator $\mathcal{U}^{\leqslant k}$; $\Phi \, \mathcal{U}^{\leqslant t} \, \Psi$ asserts that Ψ will be satisfied at some time instant in the interval $[0, t]$ and that at all preceding time instants Φ holds:

$$\sigma \models \Phi \, \mathcal{U}^{\leqslant t} \, \Psi \text{ iff } \exists x \leqslant t.\, (\sigma@x \models \Psi \wedge \forall y < x.\, \sigma@y \models \Phi)\ .$$

Note that the standard until operator is obtained by taking t equal to ∞.

CSL model checking [7,6] is performed in the same way as for CTL [11] and PCTL [17], by recursively computing the set $\mathit{Sat}(\Phi)$. For the boolean operators this is exactly as for CTL and for unbounded until this is exactly as for PCTL.

Model checking the $\mathcal{S}$ operator. For determining $\mathit{Sat}(\mathcal{S}_{\bowtie p}(\Phi))$, first $\mathit{Sat}(\Phi)$ is computed (as usual), and a graph analysis is carried out to determine the bottom strongly connected components (BSCCs) of $\mathcal{C}$, i.e. the set of SCCs in $\mathcal{C}$ that, once entered, cannot be left any more. The steady-state probability distribution $\underline{\pi}^B$ inside each BSCC B is determined using standard means [28]: by solving a linear equation system in the size of the BSCC at hand. Then, the probabilities of reaching a BSCC B from a given state s are computed for each B. State s now satisfies $\mathcal{S}_{\bowtie p}(\Phi)$ if:

$$\sum_{B} \left(\Pr\{\, \text{reach } B \text{ from } s \,\} \cdot \sum_{s' \in B \cap \mathit{Sat}(\Phi)} \pi^B(s') \right) \bowtie p$$

All these steps can be performed on the embedded DTMC as timing issues are not involved; for details see [7].

Model checking the $\mathcal{U}^{\leqslant t}$ operator. Checking time-bounded until formulas is based on determining the least solution of the following set of integral equations: $\mathit{Prob}^{\mathcal{C}}(s, \Phi \, \mathcal{U}^{\leqslant t} \, \Psi)$ equals 1 if $s \in \mathit{Sat}(\Psi)$,

$$\mathit{Prob}^{\mathcal{C}}(s, \Phi \, \mathcal{U}^{\leqslant t} \, \Psi) = \int_0^t \sum_{s' \in S} \mathbf{P}(s, s') \cdot E(s) \cdot e^{-E(s) \cdot x} \cdot \mathit{Prob}^{\mathcal{C}}(s', \Phi \, \mathcal{U}^{\leqslant t-x} \, \Psi)\, dx$$

if $s \in \mathit{Sat}(\Phi \wedge \neg \Psi)$, and equals 0 otherwise. Here, $E(s) \cdot e^{-E(s) \cdot x}$ denotes the probability density of taking some outgoing transition from s at time x. Note the resemblance with equation (1) for the PCTL bounded until operator. For

CTMC $\mathcal{C} = (S, \mathbf{R}, L)$ and CSL formula Φ let CTMC $\mathcal{C}[\Phi] = (S, \mathbf{R}', L)$ with $\mathbf{R}'(s, s') = \mathbf{R}(s, s')$ if $s \not\models \Phi$ and 0 otherwise. Note that $emb(\mathcal{C}[\Phi]) = emb(\mathcal{C})[\Phi]$. It has been shown in [6] that for a given CTMC $\mathcal{C}$ and state s in $\mathcal{C}$, the measure $Prob^{\mathcal{C}}(s, \Phi\,\mathcal{U}^{\leqslant t}\,\Psi)$ can be calculated by means of a transient analysis of the CTMC $\mathcal{C}'$, which can easily be derived from $\mathcal{C}$ using the $[\cdot]$ operator. Let $\pi^{\mathcal{C}}(s, t)(s')$ denote the probability of being in state s' at time t given that the system started in state s, i.e. $\pi^{\mathcal{C}}(s, t)(s') = \Pr_s\{\, \sigma \in Path^{\mathcal{C}} \mid \sigma@t = s' \,\}$.

Theorem 1. *[6] For CTMC $\mathcal{C}$: $Prob^{\mathcal{C}}(s, \Phi\,\mathcal{U}^{\leqslant t}\,\Psi) = \sum\limits_{s' \models \Psi} \pi^{\mathcal{C}[\neg\Phi\vee\Psi]}(s, t)(s')$.*

4 Faster time-bounded until verification

In this section, we present an algorithm for verifying time-bounded until formulas that is based on (i) the aforementioned reduction to transient analysis and on (ii) the algorithm for PCTL bounded until. This combination – suggested by the strong resemblance of Theorem 1 and Proposition 1 – yields an improvement of $\mathcal{O}(N)$ in time complexity over the algorithm suggested in [6], where N is the number of states in the CTMC. We first briefly describe uniformisation.

Uniformisation. Uniformisation is a transformation of a CTMC into a DTMC:

Definition 2. *For CTMC $\mathcal{C} = (S, \mathbf{R}, L)$ the* uniformised *DTMC is given by $unif(\mathcal{C}) = (S, \mathbf{P}, L)$ where $\mathbf{P} = \mathbf{I} + \mathbf{Q}/q$ for $q \geqslant \max\{E(s) \,|\, s \in S\}$ and $\mathbf{Q} = \mathbf{R} - diag(\underline{E})$.*

The *uniformisation rate* q is determined by the state with the shortest mean residence time. All (exponential) delays in the CTMC $\mathcal{C}$ are normalised with respect to q. That is, for each state $s \in S$ with $E(s) = q$, one epoch in $unif(\mathcal{C})$ corresponds to a single exponentially distributed delay with rate q, after which one of its successor states is selected probabilistically. As a result, such states have no self-loop in the DTMC. If $E(s) < q$ – this state has on average a longer state residence time than $\frac{1}{q}$ – one epoch in $unif(\mathcal{C})$ might not be "long enough". Hence, in the next epoch these states might be revisited and, accordingly, are equipped with a self-loop with probability $1 - \frac{E(s)}{q}$. Note the difference between the embedded DTMC $emb(\mathcal{C})$ and the uniformised DTMC $unif(\mathcal{C})$: whereas the epochs in $\mathcal{C}$ and $emb(\mathcal{C})$ coincide and $emb(\mathcal{C})$ can be considered as the timeless variant of $\mathcal{C}$, a single epoch in $unif(\mathcal{C})$ corresponds to a single exponentially distributed delay with rate q in $\mathcal{C}$.

Transient analysis. The probabilities $\pi^{\mathcal{C}}(s, t)(s')$ are now computed as follows:

$$\underline{\pi}(s, t) = \underline{\pi}(s, 0) \cdot \sum_{k=0}^{\infty} e^{-q\cdot t} \frac{(q\cdot t)^k}{k!} \mathbf{P}^k = \sum_{k=0}^{\infty} \gamma(k, q\cdot t) \cdot \underline{\pi}(s, k) \qquad (2)$$

where $\mathbf{P}$ is the probability matrix of the DTMC $unif(\mathcal{C})$, and $\gamma(k, q\cdot t)$ is the kth Poisson probability with parameter $q\cdot t$, i.e. $\gamma(k, q\cdot t) = e^{-q\cdot t}\cdot(q\cdot t)^k/k!$. The vector $\underline{\pi}(s,k)$ denotes the probability distribution in *unif*$(\mathcal{C})$ after k epochs when starting in s, i.e. $\underline{\pi}(s,k) = \underline{\pi}(s,0) \cdot \mathbf{P}^k$, where $\pi(s,0)(s) = 1$ and $\pi(s,0)(s') = 0$ if $s \neq s'$. Equation (2) can be understood as follows. During the time interval $[0,t)$, with probability $\gamma(k, q{\cdot}t)$ exactly k jumps have taken place in the DTMC *unif*$(\mathcal{C})$. The effect of these jumps is described by $\underline{\pi}(s,0){\cdot}\mathbf{P}^k$. Weighting this vector with $\gamma(k, q{\cdot}t)$ and summing over all possible numbers of jumps in $[0,t)$, we obtain, by the law of total probability, the probability vector $\underline{\pi}(s,t)$.

Given an accuracy ϵ, the number of terms R_ϵ of the infinite summation in (2) that have to be considered is the smallest value satisfying:

$$\sum_{n=0}^{R_\epsilon} \frac{(q{\cdot}t)^n}{n!} \geqslant \frac{1-\epsilon}{e^{-q\cdot t}} = (1-\epsilon){\cdot}e^{q\cdot t}$$

For large $q{\cdot}t$, R_ϵ is of order $O(q{\cdot}t)$.[4] As the first group of Poisson probabilities are typically very small, the first L_ϵ terms in (2) are negligible and need not be computed. L_ϵ and R_ϵ are called the left and right truncation point, respectively.

A first algorithm. The algorithm for time-bounded until as suggested in [6] is based on carrying out a computation according to equation (2) and the fact that $\underline{\pi}(s,k) = \underline{\pi}(s,0) \cdot \mathbf{P}^k$. The computation is carried out in an iterative manner per individual state s starting from the initial distribution $\underline{\pi}(s,0)$. The pseudo-code of this algorithm is presented in Fig. 1. Here, and in the subsequent algorithms in this paper, the Poisson probabilities are computed using the Fox-Glynn algorithm [15] that avoids overflow for large $q{\cdot}t$. The overall time complexity of this procedure is $\mathcal{O}(N{\cdot}q{\cdot}t{\cdot}M)$, where q is the uniformisation rate of the CTMC at hand, t the time bound of the until formula, N the number of states and M the number of non-zero entries in $\mathbf{R}$. This follows directly from the fact that for each state the number of terms of (2) that needs to be considered is $\mathcal{O}(q{\cdot}t)$, where each term requires a matrix vector multiplication with $\mathcal{O}(M)$ multiplications given a sparse data structure.

An alternative algorithm. The basic idea of the new algorithm is to use the iterative matrix vector multiplication of the PCTL bounded until operator as a basis, and impose the computation of the Poisson probabilities on top of it. This is suggested by the following observation:

Proposition 2. $\mathit{Prob}^{\mathcal{C}}(s, \Phi\,\mathcal{U}^{\leqslant t}\,\Psi) = \displaystyle\sum_{k=0}^{\infty} \gamma(k, q{\cdot}t) \cdot \mathit{Prob}^{\mathit{unif}(\mathcal{C})}(s, \Phi\,\mathcal{U}^{\leqslant k}\,\Psi)$

Recall that $\gamma(k, q{\cdot}t)$ denotes the probability of taking k jumps in the DTMC *unif*$(\mathcal{C})$ in the interval $[0,t)$. From Propositions 1 and 2 it follows that the vector $\underline{\mathit{Prob}}^{\mathcal{C}}(\Phi\,\mathcal{U}^{\leqslant t}\,\Psi)$ can be obtained in an iterative manner, cf. the pseudo-code in Fig. 2. As a result, a global transient analysis is carried out, yielding for each state s the probability measure $\mathit{Prob}^{\mathcal{C}}(s, \Phi\,\mathcal{U}^{\leqslant t}\,\Psi)$. Note that, as opposed to the

[4] Note that the DTMC *unif*$(\mathcal{C})$ may reach steady state before R_ϵ and, in this case, the summation can be truncated at this earlier point [24].

```
// compute Poisson probabilities
$\gamma, L_\epsilon, R_\epsilon := \textsc{FoxGlynn}(q \cdot t, \epsilon)$
// main loop
foreach $s \in S$
    $\underline{sol} := \underline{0}$
    $\underline{p} := \underline{\pi}(s, 0)$
    for $k = 1$ to $L_\epsilon - 1$
        $\underline{p} := \underline{p} \cdot \mathbf{P}$
    endfor
    for $k = L_\epsilon$ to $R_\epsilon$
        $\underline{p} := \underline{p} \cdot \mathbf{P}$
        $\underline{sol} := \underline{sol} + \gamma(k, q{\cdot}t) \cdot \underline{p}$
    endfor
    // $Prob(s, \Phi \mathcal{U}^{\leqslant t} \Psi) = \underline{sol} \cdot \underline{\iota}_\Psi^T$
endfor
```

Fig. 1. A first algorithm

```
// compute Poisson probabilities
$\gamma, L_\epsilon, R_\epsilon := \textsc{FoxGlynn}(q \cdot t, \epsilon)$
// main loop
$\underline{sol} := \underline{0}$
$\underline{b} := \underline{\iota}_\Psi$
for $k = 1$ to $L_\epsilon - 1$
    $\underline{b} := \mathbf{P} \cdot \underline{b}$
endfor
for $k = L_\epsilon$ to $R_\epsilon$
    $\underline{b} := \mathbf{P} \cdot \underline{b}$
    $\underline{sol} := \underline{sol} + \gamma(k, q{\cdot}t) \cdot \underline{b}$
endfor
// $\underline{Prob}(\Phi \mathcal{U}^{\leqslant t} \Psi) = \underline{sol}$
```

Fig. 2. An efficient variant

algorithm of Fig. 1, $\underline{sol}$ is not a probability vector in Fig. 2, i.e. its elements do not sum up to one. It is evident from the efficiency considerations given just before, that the time complexity of the adapted algorithm is $\mathcal{O}(q{\cdot}t{\cdot}M)$, thus yielding an improvement of $\mathcal{O}(N)$ over the previous algorithm.

5 Symbolic model checking CTMCs with PRISM

Due to the recent improvements in verification time – including our suggested improvement – space efficiency considerations become more important for CTMC model checking. In this section, we report on symbolic model checking of CTMCs (against CSL) using MTBDDs. MTBDDs have the ability to exploit structure (regularity) in models and can represent them in a far more compact way than a sparse matrix would. The success of BDD-based model checking in the non-probabilistic case serves as sufficient motivation to develop the foundations of MTBDD-based model checking and experiment with these techniques.

MTBDDs. Let $x_1 < x_2 < \ldots < x_n$ be distinct, totally ordered state variables. An MTBDD over $(x_1, \ldots, x_n)$ is a rooted directed graph with vertex set V containing two types of vertices:

- each *non-terminal vertex* v is labelled by a state variable $var(v) \in \{ x_1, \ldots, x_n \}$ and has two children $left(v)$, $right(v) \in V$
- each *terminal vertex* v is labelled by a real number $val(v)$,

such that $var(v) < var(w)$ for each non-terminal vertex v and non-terminal child w of v. The constraint requires that on any path from the root to a terminal vertex, the variables respect the ordering $<$. An MTBDD M over $(x_1, \ldots, x_n)$ represents the function $f_{\mathsf{M}} : \{ 0, 1 \}^n \to \mathrm{I\!R}$, whose values are obtained by traversing M

starting at the root vertex as follows. For non-terminal vertex v, the edge from v to $\mathit{left}(v)$ represents the case when $\mathit{var}(v)$ is false; the edge from v to $\mathit{right}(v)$ the case $\mathit{var}(v)$ is true. For efficiency reasons, MTBDDs are usually stored in a reduced form [10]. Note that a BDD is an MTBDD with $\mathit{val}(v) \in \{0, 1\}$ for all terminal vertices v.

Representing CTMCs by MTBDDs. Let $\mathcal{C} = (S, \mathbf{R}, L)$ be a CTMC with $|S| = 2^n$ and L injective. (Any labelled CTMC may be transformed into one satisfying these conditions by adding dummy states and new propositions.) Let $a_1, \ldots, a_n$ be an enumeration of the atomic propositions and identify each state s with the boolean n-tuple $(b_1, \ldots, b_n)$ where $b_i = 1$ iff $a_i \in L(s)$. This encoding of states is standard [5,7,13]. Thus, $S = \{0, 1\}^n$ where each state s is identified with its encoding and $\mathbf{R}$ with the function $F : \{0, 1\}^{2n} \to \mathbb{R}$ where $F(x_1, y_1, \ldots, x_n, y_n) = \mathbf{R}((x_1, \ldots, x_n), (y_1, \ldots, y_n))$.

Operations on MTBDDs. Model checking CTMCs can be performed with standard operations on MTBDDs. For completeness, we briefly describe these here. The operator Apply allows a point-wise application of the binary operator op (e.g. + or ×) to two MTBDDs. For MTBDDs M_1 and M_2, $\textsc{Apply}(\mathit{op}, \mathsf{M}_1, \mathsf{M}_2)$ yields an MTBDD for function $f_{\mathsf{M}_1} \, \mathit{op} \, f_{\mathsf{M}_2}$. For MTBDDs R and b representing matrix $\mathbf{R}$ and vector $\underline{b}$ respectively, MTBDD $\textsc{MVMult}(\mathsf{R}, \mathsf{b})$ represents the vector $\mathbf{R} \cdot \underline{b}$. For $q \in \mathbb{R}$, $\textsc{Const}(q)$ denotes the MTBDD consisting of a single terminal vertex v with $\mathit{val}(v) = q$. For an MTBDD M, $\textsc{FindMax}(\mathsf{M})$ returns the maximum value of the terminal vertices of M. The Comp operator takes an MTBDD M and an interval $I \subseteq \mathbb{R}$ and returns the BDD representing the function that equals 1 if $f_{\mathsf{M}}(x_1, \ldots, x_n) \in I$ and 0 otherwise. Operator $\textsc{Abstract}(\mathit{op}, \mathsf{M}, x_1, \ldots, x_n)$ returns the MTBDD which results from abstracting all of the variables $x_1, \ldots, x_n$ from M by applying op over all possible values taken by these variables.

MTBDD-based model checking of CSL. The symbolic model checking algorithm for CSL is identical to the one proposed in [7] except that we use transient analysis and uniformisation rather than numerical integration (which needs dedicated variants of MTBDDs). Let $\mathcal{C} = (S, \mathbf{R}, L)$ be a CTMC represented by MTBDD R as explained above. For each CSL formula Φ a BDD $\mathsf{Sat}[\![\Phi]\!]$ is defined that represents the characteristic function of the set $\mathit{Sat}(\Phi)$. By applying standard operators on MTBDDs we determine the MTBDDs P representing the transition probability matrix $\mathbf{P}$ of $\mathit{emb}(\mathcal{C})$, and E the vector of exit rates $\underline{E}$. Then:

$$
\begin{aligned}
\mathsf{Sat}[\![\mathrm{tt}]\!] &= \textsc{Const}(1) \\
\mathsf{Sat}[\![a_i]\!] &= \text{the BDD for the boolean function } (x_1, \ldots, x_n) \mapsto x_i \\
\mathsf{Sat}[\![\neg\Phi]\!] &= \textsc{Not}(\mathsf{Sat}[\![\Phi]\!]) \\
\mathsf{Sat}[\![\Phi \wedge \Psi]\!] &= \textsc{And}(\mathsf{Sat}[\![\Phi]\!], \mathsf{Sat}[\![\Psi]\!]) \\
\mathsf{Sat}[\![\mathcal{S}_{\bowtie p}(\Phi)]\!] &= \textsc{Comp}(\textsc{SteadyState}(\mathsf{P}, \mathsf{Sat}[\![\Phi]\!]), \bowtie p) \\
\mathsf{Sat}[\![\mathcal{P}_{\bowtie p}(\Phi \, \mathcal{U} \, \Psi)]\!] &= \textsc{Comp}(\textsc{Until}(\mathsf{P}, \mathsf{Sat}[\![\Phi]\!], \mathsf{Sat}[\![\Psi]\!]), \bowtie p) \\
\mathsf{Sat}[\![\mathcal{P}_{\bowtie p}(\Phi \, \mathcal{U}^{\leqslant t} \, \Psi)]\!] &= \textsc{Comp}(\textsc{TBUntil}(\mathsf{R}, \mathsf{E}, \mathsf{Sat}[\![\Phi]\!], \mathsf{Sat}[\![\Psi]\!], t, \epsilon), \bowtie p).
\end{aligned}
$$

```
algorithm TBUntil(R, E, Sat⟦$\Phi$⟧, Sat⟦$\Psi$⟧, $t$, $\epsilon$)
    // uniformisation
    R$'$ := Apply($\times$, Not(Or(Not(Sat⟦$\Phi$⟧), Sat⟦$\Psi$⟧)), R)
    E$'$ := Apply($\times$, Not(Or(Not(Sat⟦$\Phi$⟧), Sat⟦$\Psi$⟧)), E)
    $q$ := FindMax(E$'$)
    Q := Apply($-$, R$'$, Apply($\times$, E$'$, Identity))
    P := Apply($+$, I, Apply($\div$, Q, Const($q$)))
    // compute Poisson probabilities
    $\gamma, L_\epsilon, R_\epsilon$ := FoxGlynn($q \cdot t, \epsilon$)
    // main loop
    sol := Const(0)
    b := Sat⟦$\Psi$⟧
    for $k = 1$ to $L_\epsilon - 1$
        b := MVMult(P, b)
    endfor
    for $k = L_\epsilon$ to $R_\epsilon$
        b := MVMult(P, b)
        sol := Apply($+$, sol, Apply($\times$, Const($\gamma(k, q{\cdot}t)$), b))
    endfor
    return sol
end.
```

Fig. 3. MTBDD algorithm for CSL time-bounded until using transient analysis

Here, Sat⟦ a_i ⟧ is a BDD consisting of a single state vertex v labelled with x_i such that *left*(v) and *right*(v) are labelled with 0 and 1, respectively. The steady state and unbounded until operators are treated symbolically as described in [5] and [7] respectively.

TBUntil assigns to each state $s \in S$ the probability (with precision ϵ) of the set of paths that start in s fulfilling $\Phi \mathcal{U}^{\leqslant t} \Psi$, i.e. it represents the function $s \mapsto Prob(s, \Phi \mathcal{U}^{\leqslant t} \Psi)$. The algorithm for TBUntil is shown in Fig. 3, where Identity denotes the MTBDD representing an identity matrix of the appropriate size. In the first two lines, the CTMC $\mathcal{C}[\neg\Phi \vee \Psi]$ is computed. Note that the Apply operator filters out the states that become absorbing, i.e. the states that do not satisfy $\neg(\neg\Phi \vee \Psi)$. In the subsequent three lines, the uniformisation rate q and the DTMC *unif*$(\mathcal{C})$ are determined. The rest of the pseudo-code is the MTBDD-based counterpart of the algorithm shown earlier in Fig. 2.

PRISM. PRISM is a verifier for discrete probabilistic systems such as DTMCs (against PCTL) and MDPs (against PCTL [9] with fairness [8]). The tool is implemented using a combination of Java and C++. The high level parts of the tool, such as the user interface and the parsers, are written in Java. The engines and libraries are mostly written in C++. PRISM takes as input a model description in a probabilistic variant of reactive modules [1], constructs the model from its description and computes the set of reachable states. Model checking

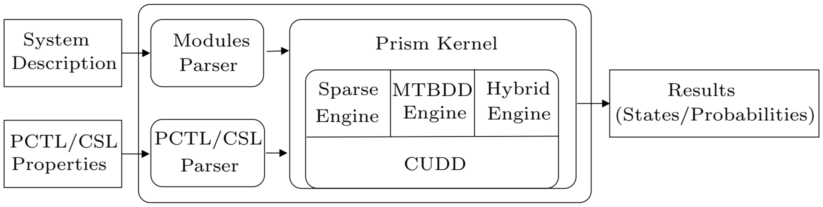

Fig. 4. The PRISM tool architecture

using different data structures is supported, cf. Fig. 4: symbolic representations using (MT)BDDs, conventional sparse matrices, and a hybrid approach using MTBDDs for storing matrices and conventional representations for probability vectors. For the manipulation of the symbolic data structures, PRISM uses the CUDD package [27] which is written in C. More information about the tool can be found at `www.cs.bham.ac.uk/~dxp/prism`.

The MTBDD based model checking algorithm for CSL has been implemented in PRISM, thus extending its applicability to *continuous* probabilistic systems. The realisation in PRISM includes an "on-the-fly" steady state detection as part of the transient analysis (as in [19]). For the sake of clarity, this mechanism is not included in the algorithm in Fig. 3.

6 Experiments

The case studies. To facilitate a comparison with $\mathsf{E} \vdash \mathsf{MC}^2$ [19], we consider two case studies that have been verified previously with CSL: a tandem network [20] and a cyclic server polling system [21].

The tandem network consists of a M/Cox$_2$/1-queue and a M/M/1-queue, both of capacity K, put in sequence. Jobs arrive at the first station with rate $4{\cdot}K$. The first server executes jobs in either done or two phases, i.e. with probability 0.9 a job is served once (with rate 2), and with probability 0.1 the job has to pass an additional phase (with rate 2). Once served by the first station, jobs are queued in the second station where service takes place with rate 4. The properties we verify for this model are the following probabilistic path properties:

- $\Diamond^{\leqslant t} \mathit{full}$, i.e. the tandem network becomes fully occupied within t time units
- $\Diamond^{\leqslant t} \mathit{fst}$, i.e. the first station of the tandem network becomes fully occupied within t time units

where $\Diamond^{\leqslant t} \Phi \equiv \mathrm{tt}\ \mathcal{U}^{\leqslant t} \Phi$.

The polling server [21] polls K stations in a cyclic fashion. The times for generating a message, for polling a station and for serving a job by a station are all distributed exponentially with rates $1/K$, 200 and 1, respectively. If the server finds a station idle, then the service time is zero. For this system, we check the property $\mathit{busy}_1 \Rightarrow \mathcal{P}_{\bowtie p}(\Diamond^{\leqslant t} \mathit{poll}_1)$, i.e. once the first station has a job to be served it will be polled within t time units with probability $\bowtie p$.

Statistics and assessment. We ran all experiments on a 440 MHz SUN Ultra 10 workstation with 512 Mb memory under the Solaris 2.7 operating system. All properties were checked with an accuracy $\varepsilon = 10^{-6}$. The verifiers PRISM and $\mathsf{E} \vdash \mathsf{MC}^2$ provide the results for the symbolic and sparse implementations respectively.

sparse matrix implementation					
model	K	# states	formula	time (in *sec*)	
				original	*improved*
tandem network	2	15	$\mathcal{P}_{\bowtie p}(\Diamond^{\leqslant 2} \mathit{full})$	0.07	0.01
			$\mathcal{P}_{\bowtie p}(\Diamond^{\leqslant 10} \mathit{full})$	0.13	0.01
			$\mathcal{P}_{\bowtie p}(\Diamond^{\leqslant 100} \mathit{full})$	0.71	0.03
			$\mathcal{P}_{\bowtie p}(\Diamond^{\leqslant 1000} \mathit{full})$	1.29	0.09
	20	861	$\mathcal{P}_{\bowtie p}(\Diamond^{\leqslant 2} \mathit{full})$	563.94	0.55
			$\mathcal{P}_{\bowtie p}(\Diamond^{\leqslant 10} \mathit{full})$	1927.59	1.01
			$\mathcal{P}_{\bowtie p}(\Diamond^{\leqslant 100} \mathit{full})$	1978.22	1.05
			$\mathcal{P}_{\bowtie p}(\Diamond^{\leqslant 1000} \mathit{full})$	1954.01	1.00
polling system	3	36	$\mathit{busy}_1 \Rightarrow \mathcal{P}_{\bowtie p}(\Diamond^{\leqslant 2} \mathit{poll}_1)$	0.96	0.02
	5	240	$\mathit{busy}_1 \Rightarrow \mathcal{P}_{\bowtie p}(\Diamond^{\leqslant 2} \mathit{poll}_1)$	59.55	0.16
	7	1,344	$\mathit{busy}_1 \Rightarrow \mathcal{P}_{\bowtie p}(\Diamond^{\leqslant 2} \mathit{poll}_1)$	2637.11	1.71
	10	15,360	$\mathit{busy}_1 \Rightarrow \mathcal{P}_{\bowtie p}(\Diamond^{\leqslant 2} \mathit{poll}_1)$	–	13.48

Table 1. Comparison of the original and improved sparse implementations of time-bounded until algorithm

Statistics and assessment for the improved method. The statistics in Table 1 compare the verification times of the original sparse implementation of time-bounded until (Fig. 1) and the improved algorithm (Fig. 2). As expected, the results confirm that the improved algorithm is a factor of N faster, where N is the size of the state space. We note that, in several cases, there is an even greater speed-up. This is possibly due to the different computation steps performed by the two algorithms: the original works via a forwards exploration of the state space, whereas the improved version works backwards. To see this, note the difference between the iteration steps $\underline{p} := \underline{p} \cdot \mathbf{P}$ in the original as opposed to $\underline{b} := \mathbf{P} \cdot \underline{b}$ in the improved.

Statistics and assessment for the symbolic implementation. We now compare our symbolic implementation of time-bounded until with its sparse counterpart. For the tandem network and polling system examples, we have constructed efficient MTBDD representations of the transition matrix using the methods presented in [14] (for further details see www.cs.bham.ac.uk/~dxp/prism). This allows us to build and store much larger models with MTBDDs (given regularity) than is feasible with a sparse implementation.

The results of the comparison of our symbolic implementation with the sparse implementation are presented in Table 2 and Fig. 5. We have measured time per

sparse versus symbolic implementation					
K	# states	time per iteration (in *sec*)			
		$\mathcal{P}_{\bowtie p}(\Diamond^{\leqslant 2} full)$		$\mathcal{P}_{\bowtie p}(\Diamond^{\leqslant 2} fst)$	
		symbolic	sparse	symbolic	sparse
63	8,128	0.08	0.01	0.02	0.01
127	32,640	0.17	0.04	0.04	0.05
255	130,816	0.37	0.55	0.06	0.15
511	523,776	0.81	1.50	0.10	0.71
1023	2,096,128	–	–	0.23	–
2047	8,386,560	–	–	0.31	–
4095	33,550,336	–	–	0.66	–

Table 2. Comparison of symbolic and sparse verification of the tandem network

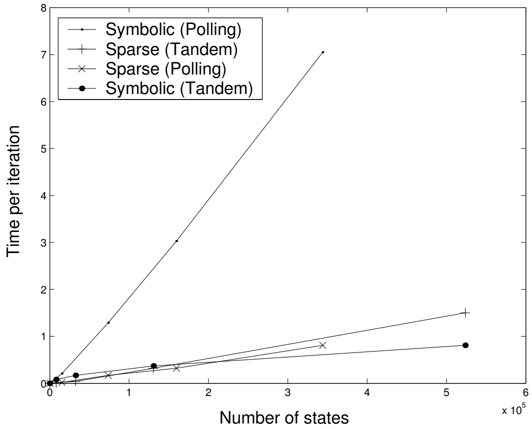

Fig. 5. Comparison of symbolic and sparse verification of both examples

iteration as both implementations follow the improved algorithm given in Fig. 2. Table 2 summarises the results for the tandem network example based on two CSL properties. Fig. 5 gives time per iteration plotted against the size of the state space for both the tandem network and polling system for the CSL properties $\mathcal{P}_{\bowtie p}(\Diamond^{\leqslant 2} full)$ and $busy_1 \Rightarrow \mathcal{P}_{\bowtie p}(\Diamond^{\leqslant 2} poll_1)$ respectively.

The efficiency of the symbolic time-bounded until implementation depends on the size of the MTBDDs representing the iteration vectors ($\underline{b}$ and $\underline{sol}$ in Fig. 2). Our experiments show that these are usually significantly larger than the MTBDD for the transition matrix, because of their relative lack of structure. For vectors to be represented compactly by MTBDDs the main requirement is a limited number of distinct elements. This condition is dependent on both the structure of the model and on the property being verified, and as such it is difficult to determine when these vectors will be represented compactly. For example, compare the difference in performance of the symbolic implementation on two different models, as shown in Fig. 5. The times for the tandem network are much faster than for the polling system for models of equivalent size.

On the other hand, in the sparse implementation the time complexity is dependent purely on the number of non-zeros in the matrix used for the computation. In Fig. 5, the times for the sparse approach can be seen to be almost identical for the tandem network and the polling system (note that in both examples the number of non-zeros in the rate matrix is linear in the size of the state space).

Comparing the results for the two implementations confirms, as expected, that we can verify larger models using a symbolic as opposed to a sparse approach; for example, we were able to verify systems with 33 million states. A more surprising observation which we note for the first time is that, for certain models and certain properties, symbolic analysis is faster than sparse. So far, see e.g. [14], the sparse implementation has always outperformed the MTBDDs on quantitative numerical calculations.

We are currently extending PRISM to improve the efficiency further by taking a *hybrid* approach which uses an MTBDD representation for storing matrices and a conventional representation for probability vectors. Early experiments show that, although slower than a sparse implementation, it is significantly faster than the pure MTBDD version. Like sparse, its performance is independent of the regularity of the model being considered, but it retains the advantage of MTBDDs, in that larger models can be represented. More information will be available in [26].

7 Concluding remarks

This paper considered both space and time efficiency issues of CSL model checking of CTMCs. We presented an improvement in time efficiency of $\mathcal{O}(N)$ for verifying time-bounded until formulas. The obtained empirical results indicate a drastic improvement in run times, making model checking of systems of realistic size feasible. In addition, we reported on symbolic model checking of CSL using MTBDD based uniformisation and transient analysis. Although, for simplicity, we have restricted the exposition in this paper to an until operator with time bounds $[0, t]$, the results of our paper carry over to $\mathcal{U}^I$ for arbitrary interval $I \subseteq \mathrm{I\!R}_{\geqslant 0}$ in a straightforward manner.

Acknowledgements. Joachim Meyer-Kayser, Hannes Bruchner and Markus Siegle (all of the University of Erlangen-Nürnberg) are kindly acknowledged for adapting the model checker $\mathsf{E} \vdash \mathsf{MC}^2$ to the new algorithm for checking time-bounded until formulas and for providing us with the new version of the tool.

The last three authors are members of the ARC project 1031 “Stochastic Modelling and Verification” funded by the British Council and DAAD.

References

1. R. Alur and T.A. Henzinger. Reactive modules. In *IEEE Symp. on Logic in Computer Science*, 207–218, 1996.
2. A. Aziz, K. Sanwal, V. Singhal and R. Brayton. Verifying continuous time Markov chains. In *Computer-Aided Verification*, LNCS 1102: 269–276, 1996.
3. A. Aziz, K. Sanwal, V. Singhal and R. Brayton. Model checking continuous time Markov chains. *ACM Trans. on Computational Logic*, **1**(1): 162–170, 2000.
4. R.I. Bahar, E.A. Frohm, C.M. Gaona, G.D. Hachtel, E. Macii, A. Pardo and F. Somenzi. Algebraic decision diagrams and their applications. *Formal Methods in System Design*, **10**(2/3): 171–206, 1997.
5. C. Baier, E. Clarke, V. Hartonas-Garmhausen, M. Kwiatkowska, and M. Ryan. Symbolic model checking for probabilistic processes. In *Automata, Languages and Programming*, LNCS 1256: 430–440, 1997.
6. C. Baier, B.R. Haverkort, H. Hermanns and J.-P. Katoen. Model checking continuous-time Markov chains by transient analysis. In *Computer Aided Verification*, LNCS 1855: 358–372, 2000.
7. C. Baier, J.-P. Katoen and H. Hermanns. Approximate symbolic model checking of continuous-time Markov chains. In *Concurrency Theory*, LNCS 1664: 146–162, 1999.
8. C. Baier and M. Kwiatkowska. Model checking for a probabilistic branching-time logic with fairness. *Distr. Comp.*, **11**(3): 125–155, 1998.
9. A. Bianco and L. de Alfaro. Model checking of probabilistic and nondeterministic systems. In *Found. of Softw. Techn. and Th. Comp. Sc.*, LNCS 1026: 499–513, 1995.
10. K. Brace, R. Rudell and R. Bryant. Efficient implementation of a BDD package. In: *27th ACM/IEEE Design Automation Conference*, 1990.
11. E. Clarke, E. Emerson and A. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. on Progr. Lang. and Sys.*, **8**: 244–263, 1986.
12. E. Clarke, M. Fujita, P.C. McGeer and J.C-Y. Yang. Multi-terminal binary decision diagrams: an efficient data structure for matrix representation. In *Formal Methods in System Design*, **10**(2/3): 149–169, 1997.
13. E. Clarke, O. Grumberg and D. Long. Verification tools for finite-state concurrent programs. In *A Decade of Concurrency*, LNCS 803: 124–175, 1993.
14. L. de Alfaro, M. Kwiatkowska, G. Norman, D. Parker and R. Segala. Symbolic model checking for probabilistic processes using MTBDDs and the Kronecker representation. In *Tools and Algorithms for the Analysis and Construction of Systems*, LNCS 1785: 395–410, 2000.
15. B.L. Fox and P.W. Glynn. Computing Poisson probabilities. *Comm. of the ACM*, **31**(4): 440–445, 1988.
16. D. Gross and D.R. Miller. The randomization technique as a modeling tool and solution procedure for transient Markov chains. *Oper. Res.* **32**(2): 343–361, 1984.
17. H.A. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Form. Asp. of Comp.*, **6**(5): 512–535, 1994.
18. V. Hartonas-Garmhausen, S. Campos and E.M. Clarke. ProbVerus: probabilistic symbolic model checking. In *Formal Methods for Real-Time and Prob. Sys.*, LNCS 1601: 96–111, 1999.
19. H. Hermanns, J.-P. Katoen, J. Meyer-Kayser and M. Siegle. A Markov chain model checker. In *Tools and Algorithms for the Construction and Analysis of Systems*, LNCS 1785: 347–362, 2000.

20. H. Hermanns, J. Meyer-Kayser and M. Siegle. Multi-terminal binary decision diagrams to represent and analyse continuous-time Markov chains. In *Proc. 3rd Int. Workshop on the Num. Sol. of Markov Chains*, pp. 188-207, 1999.
21. O.C. Ibe and K.S. Trivedi. Stochastic Petri net models of polling systems. *IEEE J. on Sel. Areas in Comms.*, **8**(9): 1649–1657, 1990.
22. J. Kemeny, J. Snell and A. Knapp. *Denumerable Markov Chains.* Van Nostrand, 1966.
23. A. Jensen. Markov chains as an aid in the study of Markov processes. *Skand. Aktuarietidskrift*, **3**: 87–91, 1953.
24. J.K. Muppala and K.S. Trivedi. Numerical transient solution of finite Markovian queueing systems. In U. Bhat, ed, *Queueing and Related Models*, Oxford University Press, 1992.
25. W.D. Obal II and W.H. Sanders. State-space support for path-based reward variables. *Perf. Ev.*, **35**: 233–251, 1999.
26. D. Parker. Implementation of symbolic model checking for probabilistic systems. Ph.D thesis, School of Computer Science, University of Birmingham, 2001 (to appear).
27. F. Somenzi. CUDD: CU decision diagram package. Public software, Colorado University, Boulder, 1997.
28. W. Stewart. *Introduction to the Numerical Solution of Markov Chains.* Princeton Univ. Press, 1994.