Model-Checking Large Structured Markov Chains

Peter Buchholz [a]

[a] *Fakultät Informatik, TU Dresden*
D-01062 Dresden, Germany
p.buchholz@inf.tu-dresden.de

Joost-Pieter Katoen [b]

[b] *Formal Methods and Tools Group*
Faculty of Computer Science, University of Twente
P.O. Box 217, 7500 AE Enschede, The Nederlands
katoen@cs.utwente.nl

Peter Kemper [c] and Carsten Tepper [c]

[c] *Informatik IV, Universität Dortmund*
D-44221 Dortmund, Germany
{kemper,tepper}@ls4.cs.uni-dortmund.de

Abstract

This paper presents algorithms and experimental results for model-checking continuous-time Markov chains (CTMCs) based on a structured analysis approach. In this approach, a CTMC is represented as a term in Kronecker algebra that reflects the component structure of the system model. Such representations can be obtained in a natural way from various high-level specification formalisms, such as stochastic extensions of Petri nets, process algebras or activity networks. Properties are expressed in Continuous Stochastic Logic (CSL) which includes means to express transient, steady-state and path performance measures. This paper describes novel model-checking algorithms for CSL that fully exploit the compositional description of the CTMC. This yields an effective way to combat the state-space explosion problem and enables the model-checking of fairly large Markov chains. Furthermore, we show how state-space aggregation (modulo bisimulation) and the elimination of vanishing states can be done in a component-wise manner. To demonstrate the applicability of the approach, and to assess the efficiency of our algorithms, we analyze a stochastic Petri net-model of a workstation cluster system and a simple queueing network.

1 Introduction

Continuous-time Markov chains (CTMCs) are the basic mathematical model underlying many formalisms for the description and analysis of stochastic discrete-event dynamic systems. Examples of significant formalisms which are mapped onto CTMCs are stochastic Petri nets (SPNs) [18], stochastic process algebras (SPAs) [9,31,35], queueing networks (QNs) [14,43] and stochastic automata networks (SANs) [42,43]. All these formalisms allow a convenient and structured model description which is mapped onto a CTMC and subsequently analyzed using established numerical techniques for the stationary or transient analysis [43]. Measures of interest are usually specified at the model description level and are then transformed into rewards at the CTMC level. This yields a stochastic process, typically a Markov reward process (MRP) [30], which can be analyzed using numerical methods similarly to a CTMC.

However, although MRPs allow an integration of process and measures of interest, the pure specification of rewards is not always the right way to describe the property one wants to evaluate for a system. Often the requirements for a system state that the system has to observe at least a specific level of performance and a model is used to check that the system fulfills this task. For (untimed) concurrent systems, a popular approach for such a verification is to employ automated verification techniques such as *model checking* [23]. Model checking amounts to a systematic check of the validity of a requirement, formulated as a formula in an appropriate temporal logic, in all states of the model. Model checking has been widely used in practice [23] and has recently been extended in various directions including an extension for model-checking CTMCs. For this purpose, the temporal logic CSL (Continuous Stochastic Logic) was developed [1,2] as a stochastic extension of the branching-time temporal logic CTL (Computation Tree Logic). In several follow-up papers this logic has been extended [6], applications have been described [29,4] and sophisticated model-checking algorithms have been introduced [36].

Like traditional CTMC analysis, the major problem of model-checking realistic examples is the *state-space explosion*. A possible approach to attack this state-space explosion problem is the use of symbolic techniques which extend approaches from standard model checking [23] and represent transition matrices using directed graph structures like MTBDDs (multi-terminal BDDs) [3,22]. For CSL model checking, this approach has recently been pursued in [36]. However, although MTBDDs may reduce the memory requirements, their operations become very slow and, typically, the model structure is not reflected by the symbolic representation. Besides, reduction using equivalences such as bisimulation does not always lead to a smaller BDD-representation of the model at hand.

Another approach to alleviate state-space explosion for the numerical analysis of CTMCs and also for several functional analysis steps are so-called *structured analysis* approaches. The idea of these approaches is to represent the huge generator or transition matrix as a sum of Kronecker products of small component matrices. The approach has been first developed for networks of stochastic automata [42,43] and has been subsequently extended in various directions [14,8,19,24,38]. Structured analysis approaches allow the efficient analysis of large Markov chains and, due to the compositional description of the transition matrix, several other steps exploiting the model structure like aggregation due to equivalence or local elimination of vanishing states can be performed efficiently. As shown in [16,40] also CTL (or LTL) model checking can be realized very efficiently in the context of structured description of (ordinary) transition systems.

In this paper, we present a structured model-checking approach for CSL, thus combining and extending structured analysis and model checking of CTL-formulas. The presented approach has been implemented and integrated in the APNN (Abstract Petri Net Notation) toolbox, a modular tool environment for the qualitative and quantitative analysis of discrete-event systems [15]. To summarize, this paper:

- presents model-checking algorithms for CSL that exploit structured analysis techniques based on a Kronecker representation of the CTMC,
- shows how bisimulation equivalence (i.e., lumping) can be used to minimize system descriptions component-wise while preserving the validity of CSL-formulas,
- allows the consideration of vanishing states in CSL model checking, and
- provides empirical results substantiating the claim that structured analysis techniques are powerful means to combat the state-space explosion problem in model-checking CSL.

The techniques presented in this paper provide ample means to exploit the compositional nature of the model description – being it a superposed GSPN, a stochastic process algebra specification, or the like – for both the model-checking procedures as well as for model reduction prior to verification.

Organization of the paper. The following two sections introduce the basic model class, the underlying structured description of the CTMC, some notations and basic algorithms for stationary and transient CTMC analysis. Section 4 reviews the logic CSL and presents the basic steps of model checking CSL. Section 5 combines both approaches and presents the model-checking approach which uses the Kronecker representation of the generator matrix of the CTMC. Section 6 shows under which conditions vanishing states can be eliminated locally in the components. Section 7 introduces how bisimulation equivalence, which preserves the validity of CSL-formulas, can be used to reduce components and how the entire components can be substituted by reduced components in the structured description. Section 8 reports on the practical experiments that we conducted. Section 9 concludes the paper. The appendix summarizes the Kronecker operations.

2 Basic Definitions and Notations

Structured descriptions have been proposed in different forms and notations vary significantly. This paper uses the model class and most of the notation of [14,19]. For the representation of the state space we exploit the extensions given in [16]. The structured model-checking approach we present here can be easily extended to other Kronecker representations of the generator matrix like the hierarchical approach [8,12,17] or matrix diagrams [19,20]. However, to avoid an overloading of the notation, we restrict ourselves to the basic model class of [14].

2.1 Notational conventions

Throughout the paper we use italic letters for scalars, calligraphic letters for sets, lower case boldface Roman or Greek letters for vectors and upper case boldface Roman letters for matrices. Vectors are usually row vectors, unless stated otherwise. $\mathbb{R}, \mathbb{R}_{\geqslant 0}$ and $\mathbb{R}_{>0}$ are the sets of real numbers, non-negative real numbers and strictly positive real numbers. Matrix and vector elements are indicated using brackets, e.g., $\mathbf{A}(i,j)$ and $\mathbf{a}(i)$ indicate the i,j-th of matrix $\mathbf{A}$ and the i-th element of vector $\mathbf{a}$, respectively. $\mathbf{A}^T$ and $\mathbf{a}^T$ denote the transposed matrix and vector, respectively. $\mathbf{I}_n$ indicates the identity matrix of order n. The value n is omitted if it is clear from the context. $\mathbf{1}$ and $\mathbf{0}$ are row vectors with 1 or 0 in all positions. Their length follows from the context. Numbering of elements in vectors and matrices starts with 0 and ends with $n-1$ for an n-dimensional vector or matrix. Square brackets are used to denote sub-matrices or sub-vectors, e.g., if $\mathbf{A} \in \mathbb{R}^{n \times n}$ and $\mathcal{A}, \mathcal{B} \subseteq \{0, \ldots, n-1\}$, then $\mathbf{A}[\mathcal{A}, \mathcal{B}]$ is a matrix including all rows with indices belonging to $\mathcal{A}$ and all columns with indices belonging to $\mathcal{B}$. $\mathbf{A}[\mathcal{A}, \mathcal{B}](i,j)$ with $0 \leqslant i < |\mathcal{A}|$ and $0 \leqslant j < |\mathcal{B}|$ denotes element i, j in the sub-matrix. In the same way sub-vectors are defined, e.g., $\mathbf{a}[\mathcal{A}]$ is the sub-vector of $\mathbf{a}$ including all elements belonging to $\mathcal{A}$.

2.2 Continuous-time Markov chains

As the basic mathematical model we consider CTMCs with state space $\mathcal{T}$ and generator matrix $\mathbf{Q}$. We are interested in stationary or transient results. Both types of results can be derived from the stationary or transient distribution vector of the CTMC. The stationary solution vector $\boldsymbol{\pi}$ is the solution of

$$\boldsymbol{\pi} \cdot \mathbf{Q} = \mathbf{0} \quad \text{subject to} \quad \boldsymbol{\pi} \cdot \mathbf{1}^T = 1 \ . \tag{1}$$

The stationary solution is unique if the CTMC is ergodic. The transient solution $\boldsymbol{\pi}_t$ at time t is the solution of the differential equation $\dot{\boldsymbol{\pi}}_t = \boldsymbol{\pi}_t \cdot \mathbf{Q}$ and is expressed as:

$$\boldsymbol{\pi}_t = \boldsymbol{\pi}_0 \cdot exp\,(\mathbf{Q} \cdot t) \tag{2}$$

where $\boldsymbol{\pi}_0$ is the initial distribution at time 0. For stationary and transient analysis of CTMCs a large number of different solution techniques exist, for an excellent overview see [43].

Results related to CTMCs are often expressed in terms of rewards which can be assigned to states (*rate-based rewards*) or transitions (*impulse-based rewards*). Rewards are usually defined with respect to some measures-of-interest of the describing high-level model, e.g., populations in a queue of a queueing network. We consider rate-based rewards. (For the model-checking algorithm these rewards are between 0 and 1.) The column vector $\boldsymbol{\rho} \in \mathbb{R}_{\geqslant 0}^{|\mathcal{T}|}$ assigns non-negative rewards to the states of a CTMC with state space $\mathcal{T}$. The mean stationary reward is given by $\boldsymbol{\pi} \cdot \boldsymbol{\rho}$, the mean transient reward at time t equals $\boldsymbol{\pi}_t \cdot \boldsymbol{\rho}$, and the accumulated reward in the interval $[t_1, t_2)$ is computed as $\int_{t_1}^{t_2} \boldsymbol{\pi}_t \cdot \boldsymbol{\rho} \; dt$.

2.3 Component-based descriptions

We consider Markov models consisting of components that interact via synchronized transitions. Such descriptions are common for stochastic process algebras [9,31,35], superposed stochastic Petri nets [24,38], stochastic automata networks [42,43] or queueing networks[14,43].

The model consists of K components numbered 1 through K that communicate via synchronization events from a set $\mathcal{E}_S$. Additionally, a set of local events $\mathcal{E}_L$ describes transitions which do not belong to a communication. The sets $\mathcal{E}_{S,k}$ and $\mathcal{E}_{L,k}$ contain the synchronized and local events in which component k participates ($\mathcal{E}_{S,k} \cap \mathcal{E}_{L,k} = \varnothing$). Each $e \in \mathcal{E}_L$ belongs to exactly one set $\mathcal{E}_{L,k}$ whereas each $e \in \mathcal{E}_S$ belongs to at least two sets $\mathcal{E}_{S,k}$ as it describes a communication between components. $rate(e)$ is the rate of transition $e \in \mathcal{E} = \mathcal{E}_S \cup \mathcal{E}_L$.

Let $\widehat{\mathcal{T}}^k$ be the state space of component k with $n_k = |\widehat{\mathcal{T}}^k|$. States in $\widehat{\mathcal{T}}^k$ are numbered 0 through $n_k - 1$. For the moment we assume that $\widehat{\mathcal{T}}^k$ contains only tangible states and consider the problem of vanishing in Section 6. $\mathbf{W}_{k,e} \in \mathbb{R}_{\geqslant 0}^{n_k \times n_k}$ is a matrix describing the transitions according to event e in component k. Define

$$\mathbf{R}_k = \sum_{e \in \mathcal{E}_{L,k}} rate(e) \cdot \mathbf{W}_{k,e}$$

the matrix of local transition rates for component k. For notational convenience let $\mathbf{W}_{k,e} = \mathbf{I}_{n_k}$ for $e \notin \mathcal{E}_{L,k} \cup \mathcal{E}_{S,k}$. Boolean matrices $\mathbf{W}_{k,e}$ are sufficient to indicate whether an event can take place or not; numerical entries in $\mathbf{W}_{k,e}$ other than 0 and 1 can be used to encode state dependent scaling factors that speed up or slow down an event. This feature is also used to account for the effect of probabilistic output bags, where an event that takes place in a state can make transitions to several, different states and the selection of a transition takes places according to a probability distribution. These probabilities give the numerical entries of $\mathbf{W}_{k,e}$. Values of $\mathbf{W}_{k,e}$ can also be used to carry path probabilities for sequences of state transitions passing through vanishing states. Vanishing states are states with a zero sojourn time. Since those states are somewhat irrelevant for the timing behavior, it is common practice in performance evaluation to eliminate vanishing states and retain only the probabilities through sequences (paths) of vanishing states. $\mathbf{W}_{k,e}$ contains probabilities of such sequences that immediately follow an initial event e.

Let $n_1^0 = 1$ and for $l \leqslant m$ let $n_l^m = \prod_{k=l}^{m} n_k$. The *potential state space* of the composed model with components $1, \ldots, K$ equals $\widehat{\mathcal{T}} = \widehat{\mathcal{T}}^1 \times \ldots \times \widehat{\mathcal{T}}^K$ and contains n_1^K states. Each state s from $\widehat{\mathcal{T}}$ is described by a K-dimensional vector $(x^1, \ldots, x^K)$ where $0 \leqslant x^k < n_k$; this can be linearized to $s = \sum_{k=1}^{K} x^k \cdot n_1^{k-1}$ $(0 < s < n_1^K)$. We use both representations interchangeably in this paper.

Based on this representation, the rate-matrix of the composed model is given by:

$$\widehat{\mathbf{R}} = \sum_{e \in \mathcal{E}} rate(e) \cdot \bigotimes_{k=1}^{K} \mathbf{W}_{k,e} = \bigoplus_{k=1}^{K} \mathbf{R}_k + \sum_{e \in \mathcal{E}_S} rate(e) \cdot \bigotimes_{k=1}^{K} \mathbf{W}_{k,e} \tag{3}$$

where $\otimes$ and $\oplus$ are the Kronecker sum and product, respectively (for details about these operations see [14,42] and the appendix). Thus, $\widehat{\mathbf{R}}$ is defined as the sum of (a Kronecker sum of) transition matrices $\mathbf{R}_k$ that refer to all transitions local to component k, and (a Kronecker product of) the transitions resulting from synchronizations between components.

Define $\widehat{\mathbf{Q}} = \widehat{\mathbf{R}} - \mathit{diag}(\widehat{\mathbf{R}} \cdot \mathbf{1}^T)$ where $\mathit{diag}(\mathbf{a})$ is a square matrix having the elements of vector $\mathbf{a}$ on the diagonal. The set of *reachable states* equals $\mathcal{T} \subseteq \widehat{\mathcal{T}}$ and we let $n = |\mathcal{T}|$ denote its cardinality. Let $\mathbf{Q} = \widehat{\mathbf{Q}}[\mathcal{T}, \mathcal{T}]$. As often $\mathcal{T} \subset \widehat{\mathcal{T}}$, we have $\mathbf{Q} \neq \widehat{\mathbf{Q}}$ [14]. The set $\mathcal{T}$ can be efficiently computed from the representation (3), since for $x \in \mathcal{T}$, we have that $\widehat{\mathbf{R}}(x, y) > 0$ implies $y \in \mathcal{T}$. For details about the corresponding algorithms we refer to [16]. We assume in the sequel that for each $y^k \in \widehat{\mathcal{T}}^k$ there exists a state $(x^1, \ldots, x^K) \in \mathcal{T}$ such that $x^k = y^k$; otherwise, we may delete y^k from $\widehat{\mathcal{T}}^k$. Observe that equation (3) implicitly also includes a description of the reachability graph of the system, where rates are neglected. If only the reachability graph (and not the generator matrix) is of interest, then it suffices to consider Boolean (instead of real) matrices. The structured representation, however, remains the same.

2.4 Representing sets of states

Different analysis steps work on different subsets of the state space, e.g., numerical analysis methods require knowledge of $\mathcal{T}$ (rather than $\widehat{\mathcal{T}}$) while model-checking algorithms typically consider states which fulfill a sub-formula of the formula to be verified. For $\mathcal{S} \subseteq \widehat{\mathcal{T}}$, let $\Xi_{\mathcal{S}}$ be a function that maps any state $s \in \mathcal{S}$ onto a natural number that is smaller than $|\mathcal{S}|$ such that $\Xi_{\mathcal{S}}$ is monotonic with respect to $<$, i.e., $s < s' \Rightarrow \Xi_{\mathcal{S}}(s) < \Xi_{\mathcal{S}}(s')$ for $s, s' \in \mathcal{S}$. This function is used to map the K-dimensional representation of reachable states with a lexicographical order to an index set $\{0, \ldots, |\mathcal{S}|-1\}$ while preserving this order.

We now describe a data structure that represents $\mathcal{S}$ in a compact way while allowing fast access to individual states. First, notice that $\widehat{\mathcal{T}}$ can be represented by a tree with K levels, where each node at level k has n_k sons. A path in the tree corresponds to a state $(x^1, \ldots, x^K)$. Elimination of all paths to states from $\widehat{\mathcal{T}} \setminus \mathcal{S}$, yields a representation of $\mathcal{S}$. The remaining nodes at level k describe states from $\widehat{\mathcal{T}}^k$ belonging to $\mathcal{S}$. Let $RS_{\mathcal{S}}(x^1, \ldots, x^k) = \{\mathbf{y} \in \mathcal{S} \mid x^1 = y^1 \wedge \ldots \wedge x^k = y^k\}$, the subset of states from $\mathcal{S}$ with fixed states for the components 1 through k. $RS_{\mathcal{S}}(x^1, \ldots, x^k)$ refers to a sub-tree with root at level k+1; this sub-tree is denoted $RS_{\mathcal{S}}^{k+1}(x^1, \ldots, x^k)$ (which is the subset of states from $\widehat{\mathcal{T}}^{k+1}$ which belong to states from $\mathcal{S}$ given that the state of components 1 through k equals $(x^1, \ldots, x^k)$). Finally, $RS_{\mathcal{S}}() = \mathcal{S}$ and $RS_{\mathcal{S}}^1()$ is the root of the tree.

Let us now discuss how we can minimize this tree representation. Sub-trees

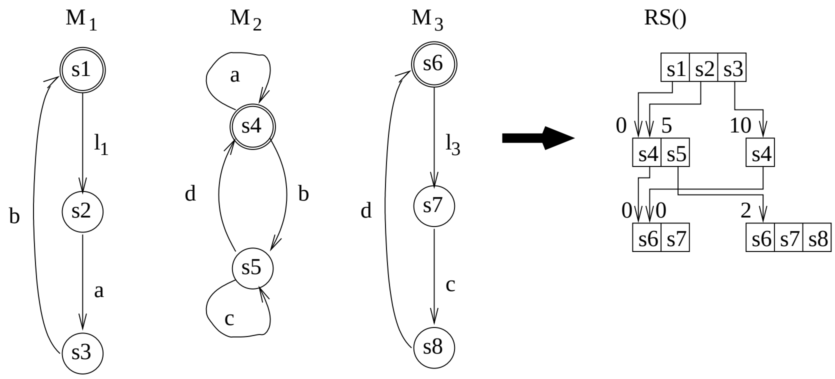

Fig. 1. Example with three components and DAG representation of $\mathcal{T}$.

$RS_{\mathcal{S}}(x^1, \ldots, x^k)$ and $RS_{\mathcal{S}}(y^1, \ldots, y^k)$ are equal, if and only if

$$RS^k_{\mathcal{S}}(x^1, \ldots, x^{k-1}) = RS^k_{\mathcal{S}}(y^1, \ldots, y^{k-1})$$

and all pairs of sub-trees $RS_{\mathcal{S}}(x^1, \ldots, x^k, x^{k+1})$, $RS_{\mathcal{S}}(y^1, \ldots, y^k, y^{k+1})$ with $x^{k+1} = y^{k+1}$ are equal. Using a folding operation in a bottom-up manner, like in ordered BDDs [7], equal sub-trees are represented once. For a given ordering of components, this yields a unique acyclic graph (DAG) that represents $\mathcal{S}$. By introducing appropriate arc inscriptions it is possible to derive for each state $s \in \mathcal{S}$ the number $\Xi_{\mathcal{S}}(s)$. To realize this mapping, one basically has to count the number of leaves to the left of the path of a state x in the tree. The cardinality of leaves in isomorphic sub-trees is equal, and thus remains invariant under folding. By assigning corresponding weights on arcs of the DAG, $\Xi_{\mathcal{S}}$ is evaluated for a path $x^1, \ldots, x^K$ in the DAG by summing arc weights at each node which leave from the left positions of x^k in $(x^1, \ldots, x^{k-1})$ plus the position of x^K in $RS_{\mathcal{S}}(x^1, \ldots, x^{K-1})$. An implementation typically precomputes such weights to avoid the local summation at each node.

Example 2.1 *Consider the transition graphs of components M_1, M_2 and M_3 depicted in Figure 1. Rates are omitted in this example, as we are only interested in a representation of the reachable state space. Labels l_i $(i = 1, 2, 3)$ refer to local transitions in component i, all remaining labels describe synchronizations. Labels a, b indicate synchronizations between M_1 and M_2, whereas c, d indicate synchronizations between M_2 and M_3. Initial states are indicated by double circles. $\widehat{\mathcal{T}}$ consists of 18 states of which 12 are reachable. The DAG on the right-hand side of Figure 1 represents $\mathcal{T}$. Values for $\Xi_{\mathcal{T}}$ are derived by traversing the path through the graph from top to bottom, adding all arc weights and adding at the end the lexicographical position of the element in the leaf node (starting with 0 for the first element), e.g., $\Xi_{\mathcal{T}}(s2, s5, s7) = 5 + 2 + 1 = 8$ and $\Xi_{\mathcal{T}}(s3, s4, s6) = 10 + 0 + 0 = 10$.*

3 Structured Analysis of Markov Chains

In the previous section, compact representations for $\widehat{\mathbf{Q}}$ and $\mathcal{T}$ have been presented. These representations are only useful if they can be exploited in analysis algorithms. Fortunately, this is the case for the Kronecker representation of the generator matrix and the DAG representation of $\mathcal{T}$. In this section, we give a brief overview of structured analysis techniques and refer to the literature for further details. Transient analysis is considered in some more detail, because it is the basic technique to verify time-bounded until operators in CSL (see Section 4).

3.1 Stationary analysis

For the determination of the stationary vector $\boldsymbol{\pi}$ according to (1), iterative numerical solution techniques are applied which compute consecutive approximations $\mathbf{p}^{(i)}$ until the iteration vector is sufficiently near to $\boldsymbol{\pi}$. A large number of iterative methods exist for the stationary analysis of CTMCs [43]. Many of them are applicable in conjunction with the Kronecker representation of the generator matrix. Distinguishing aspects of the iterative methods are whether they perform iterations on $\widehat{\mathcal{T}}$ (states from $\widehat{\mathcal{T}} \setminus \mathcal{T}$ receive zero probability during iteration) or on (preferably) $\mathcal{T}$ and whether iteration is performed row-wise or column-wise. Different algorithms to multiply a vector with a Kronecker representation of $\mathbf{Q}$ are introduced and tested in [14]. That paper also presents basic numerical solution algorithms based on these multiplication algorithms. Advanced structured solution techniques are presented in [13].

3.2 Transient analysis

For transient analysis, i.e., the computation of $\boldsymbol{\pi}_t$ according to (2), *randomization* [27,43] is the most efficient analysis technique for almost all CTMCs. Randomization in conjunction with the Kronecker representation of the generator matrix has been considered in [39]. Since randomization is also applied for model-checking CSL (as discussed in [5]), we present here the main steps. Randomization requires computing Poisson probabilities for a rate λ and a time horizon t according to $\beta_i(\lambda, t) = e^{-\lambda \cdot t} \frac{(\lambda \cdot t)^i}{i!}$, i.e., $\beta_i(\lambda, t)$ is the probability that a Poisson process with rate λ generates i events in an interval of length t. A straightforward computation of $\beta_i(\lambda, t)$ is numerically stable only for relatively small values of $\lambda \cdot t$, for arbitrary values of $\lambda \cdot t$, the approach of Fox and Glynn [26] is frequently employed, which gives left and right truncation points $0 \leqslant l < r$, such that $\beta_i(\lambda, t) = 0$ for $i < l$ and $i > r$.

Let $\mathbf{p}^{(0)} = \boldsymbol{\pi}_0$ be the initial vector. Then $\boldsymbol{\pi}_t$ is given by:

$$\boldsymbol{\pi}_t = \sum_{i=0}^{\infty} \mathbf{p}^{(i)} \cdot \beta_i(\lambda, t) \tag{4}$$

where $\mathbf{p}^{(i+1)} = \mathbf{p}^{(i)} + \lambda^{-1} \cdot \mathbf{p}^{(i)} \cdot \mathbf{Q}$ $(i \geqslant 0)$ and $\lambda \geqslant \max_{x \in \{0,\ldots,|\mathcal{T}|\}}(|\mathbf{Q}(x,x)|)$. The above sum can be truncated by defining i_ε such that $1 - \sum_{i=0}^{i_\varepsilon} \beta_i(\lambda, t) \leqslant \varepsilon$. In practice i_ε is often significantly smaller than r, the right truncation point. Truncation of the sum in (4) yields a result vector where the error can be bounded by ε.

Often measures-of-interest are computed for a reward vector $\boldsymbol{\rho}$. In this case, (4) becomes

$$\mathbf{p}^{(0)} \cdot \left(\sum_{i=0}^{i_\varepsilon} \beta_i(\lambda, t) \cdot \left(\mathbf{I} + \lambda^{-1} \cdot \mathbf{Q} \right)^i \right) \cdot \boldsymbol{\rho} \tag{5}$$

to compute the expected reward at time t. If $\mathbf{Q}$ is defined as sub-matrix of $\widehat{\mathbf{Q}}$ – which has a Kronecker representation – then $\mathbf{I} + \lambda^{-1}\mathbf{Q}$ also has a Kronecker representation, i.e.,

$$\mathbf{I} + \lambda^{-1} \cdot \mathbf{Q} = \underbrace{(\mathbf{I} - \lambda^{-1} \cdot diag(\widehat{\mathbf{R}} \cdot \mathbf{1}^T)[\mathcal{T}, \mathcal{T}])}_{= diag(\mathbf{d})} + \lambda^{-1} \cdot \widehat{\mathbf{R}}[\mathcal{T}, \mathcal{T}] \tag{6}$$

Here $\mathbf{d}$ is a vector of length $\mathcal{T}$ that contains diagonal entries. Recall that $\widehat{\mathbf{R}}$ is represented in a structured way, cf. equation (3). The scalar value λ^{-1} can be multiplied with matrices of Kronecker sums and one matrix of each Kronecker product such that a similar structure as for $\mathbf{Q}$ is obtained. This facilitates the use of the multiplication algorithms for vectors and sub matrices of $\widehat{\mathbf{R}}$ as presented in [14].

In principle, the iteration (5) can be performed from left-to-right or from right-to-left. In both cases, vector-matrix products have to be computed, either by row or by column. A multiplication from right-to-left is preferable if results have to be computed for different initial vectors $\mathbf{p}^{(0)}$, because for each vector only a vector product has to be computed (provided the product of the reward vector with the matrix has been precomputed). This will appear during CSL model-checking, cf. Section 5. Multiplication from left-to-right is preferable when results are computed for a single initial vector. In this case, the computation for different reward vectors only requires a single vector product (provided the product of the initial distribution with the matrices has been precomputed).

Further optimizations of randomization in combination with a Kronecker representation of the generator matrix are considered in [39].

4 Model-Checking Markov Chains

The temporal logic CSL, developed originally in [1,2] and extended in [6], provides a powerful means to specify path-based as well as traditional state-based measures on CTMCs in a concise, flexible and unambiguous way. This paper considers model-checking CSL based on a Kronecker representation of the CTMC under consideration. This section presents the important basic concepts of paths and state-labeling, recalls the logic CSL, and summarizes its main model-checking algorithms.

4.1 Context

CSL is based on the well-known branching-time temporal logic CTL (Computation Tree Logic [21]) and PCTL (Probabilistic CTL [28]); 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.

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 [1,2]. Approximate CSL model-checking algorithms have been studied in [6] where the satisfaction of time-bounded until formulas is shown to be based on solving a (recursive) Volterra equation system. More recently, verifying time-bounded until formulas has been reduced to the problem of computing transient-state probabilities for CTMCs [5]. This significant result employs a formula-dependent transformation of the CTMC and – more importantly – allows one to adopt efficient techniques like *randomization* [27,43] for verifying time-bounded until-formulas. A naive approach to model-check time-bounded until properties requires to perform this procedure for each state. An improvement suggested in [36] accumulates the required probabilities for all states simultaneously, yielding an improvement of $O(|\mathcal{T}|)$ time. Two prototype implementations for CSL model checking do exist: ETMCC [32] that is based on a representation of the CTMC by sparse matrices, and PRISM [36,41], a symbolic CSL model checker using multi-terminal binary decision diagrams (MTBDDs).

4.2 State-labeling and paths

For model-checking purposes, CTMCs are extended with a state-labeling that indicates which elementary propositions hold in a state. In practice, atomic propositions connect a logic with a specific modeling formalism like stochastic Petri nets or stochastic process algebras. Here, we consider state-based propositions, i.e., an atomic proposition is an (in)equality over functions with state variables as their parameters. A concrete example of a set of atomic propositions are (in)equalities of arithmetic expressions that are constructed from state variables, constants (e.g., reals) and arithmetic operations (such as $\{+,-,\cdot\}$). In a Petri net context, state variables are places and the value of a state variable in a given state is defined by the number of tokens at the place. Example atomic propositions are, e.g., "the number of tokens in place P exceeds 4", or "$P1$ and $P2$ together contain at least 2 tokens". More concrete examples are provided in Section 8.

Let AP be a fixed, finite set of atomic propositions. Function L is a *labeling* function which assigns to each state $s \in \mathcal{T}$ the set $L(s)$ of atomic propositions that are valid in s. To allow for a standard interpretation of temporal operators such as the until-operator in the untimed case, we do allow self-loops. We thus allow the system to occupy the same state before and after a transition. This does neither alter the transient nor the stationary behavior of the CTMC since diagonal entries of $\mathbf{Q}$ contain the negative row sums of off-diagonal values, so presence or absence of self-loops does not make a difference and is visible in $\mathbb{R}$, but not in $\mathbf{Q}$. Let $\mathbf{R} : \mathcal{T} \times \mathcal{T} \to \mathbb{R}_{\geqslant 0}$ be the rate matrix of the CTMC, with $\mathbf{R}(s,s) > 0$ for states equipped with a self-loop, and $\mathbf{e}(s) = \sum_{s' \in \mathcal{T}} \mathbf{R}(s,s')$. The probability of taking a transition from s within t time units thus equals $1 - e^{-\mathbf{e}(s)\cdot t}$. A state s is absorbing if $\mathbf{R}(s,s') = 0$ for all states s'. We have $\mathbf{Q} = \mathbf{R} - diag(\mathbf{e})$.

A path through a CTMC M 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 $Path^M$ denote the set of paths through M. $\sigma[i]$ denotes the $(i{+}1)$th state of σ, i.e., $\sigma[i] = s_{i+1}$. $\sigma@t$ denotes the state of σ occupied at time t, i.e., $\sigma@t = \sigma[i]$ with i the largest index such that $t < \sum_{j=0}^{i} t_j$. Let $\Pr_s$ denote the unique probability measure on sets of paths that start in s [6].

4.3 The temporal logic CSL

Let $a \in AP$, $p \in [0,1]$, comparison operator $\trianglelefteq \ \in \{\leqslant, \geqslant\}$ and $I \subseteq \mathbb{R}_{\geqslant 0}$. The syntax of CSL is:

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

The other Boolean connectives are derived in the usual way, i.e., $\mathrm{ff} = \neg\mathrm{tt}$, $\Phi \vee \Psi = \neg(\neg\Phi \wedge \neg\Psi)$, and $\Phi \rightarrow \Psi = \neg\Phi \vee \Psi$. $\mathcal{S}_{\trianglelefteq p}(\Phi)$ asserts that the steady-state probability for a Φ-state meets the bound $\trianglelefteq p$. $\mathcal{P}_{\trianglelefteq p}(\Phi \,\mathcal{U}^{I}\, \Psi)$ asserts that the probability measure of the paths satisfying $\Phi \,\mathcal{U}^{I}\, \Psi$ (see below) meets the bound given by $\trianglelefteq p$. (For the sake of simplicity, we do not consider the next state operator in this paper). The semantics of CSL is defined by [6,5]:

$$\begin{array}{ll}
s \models \mathrm{tt} & \text{for all } s \in \mathcal{T} \\
s \models a & \text{iff } a \in L(s) \\
s \models \neg\Phi & \text{iff } s \not\models \Phi \\
s \models \mathcal{S}_{\trianglelefteq p}(\Phi) & \text{iff } \lim_{t\to\infty} \Pr_s\{\, \sigma \in \mathit{Path}^M \mid \sigma@t \models \Phi \,\} \trianglelefteq p \\
s \models \mathcal{P}_{\trianglelefteq p}(\Phi \,\mathcal{U}^{I}\, \Psi) & \text{iff } \mathrm{Prob}^M(s, \Phi \,\mathcal{U}^{I}\, \Psi) \trianglelefteq p
\end{array}$$

The limit in the fourth equation always exists as M contains finitely many states [43]. $\mathrm{Prob}^M(\cdot)$ is defined by:

$$\mathrm{Prob}^M(s, \Phi \,\mathcal{U}^{I}\, \Psi) = \Pr_s\{\, \sigma \in \mathit{Path}^M \mid \sigma \models \Phi \,\mathcal{U}^{I}\, \Psi \,\}\,.$$

$\Phi \,\mathcal{U}^{I}\, \Psi$ asserts that Ψ will be satisfied at some time instant in the interval I and that at all preceding time instants Φ holds:

$$\sigma \models \Phi \,\mathcal{U}^{I}\, \Psi \text{ iff } \exists t \in I.\ (\sigma@t \models \Psi \wedge \forall \tau < t.\, \sigma@\tau \models \Phi)\ .$$

Note that for $I = \varnothing$ the formula $\Phi \,\mathcal{U}^{I}\, \Psi$ is not satisfiable The standard until operator is obtained by taking I equal to $[0,\infty)$.

4.4 Model-checking algorithms

Model-checking CSL [6,5] is performed in the same way as for CTL [21] and PCTL [28], by recursively computing the set $Sat(\Phi)$, i.e., the set of states that satisfy Φ. For the Boolean operators this is exactly as for CTL.

For determining $Sat(\mathcal{S}_{\trianglelefteq p}(\Phi))$, first $Sat(\Phi)$ is computed, and a graph analysis is carried out to determine the bottom strongly connected components (BSCCs) of M, i.e. the set of SCCs in M that, once entered, cannot be left any more. The steady-state probability distribution $\boldsymbol{\pi}^B$ inside each BSCC B is determined using standard means [43]: 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 $\mathcal{S}_{\trianglelefteq p}(\Phi)$ if:

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

Checking time-bounded until formulas is based on determining the least solution of a set of integral equations. For instance, for the simplest case, $I = [0, t]$ we have that $\mathrm{Prob}^M(s, \Phi\,\mathcal{U}^{[0,t]}\,\Psi)$ equals 1 if $s \in Sat(\Psi)$,

$$\mathrm{Prob}^M(s, \Phi\,\mathcal{U}^{[0,t]}\,\Psi) = \int_0^t \sum_{s' \in \mathcal{T}} \frac{\mathbf{R}(s,s')}{\mathbf{e}(s)} \cdot \mathbf{e}(s) \cdot e^{-\mathbf{e}(s)\cdot\tau} \cdot \mathrm{Prob}^M(s', \Phi\,\mathcal{U}^{[0,t-\tau]}\,\Psi)\,d\tau$$

if $s \in Sat(\Phi \wedge \neg\Psi)$, and equals 0 otherwise. Here, $\mathbf{e}(s) \cdot e^{-\mathbf{e}(s)\cdot\tau}$ denotes the probability density of taking some outgoing transition from s at time τ. For intervals $I = [t, t']$ with $t > 0$ a similar, but more complicated equation is obtained, see [4].

For CTMC $M = (\mathcal{T}, \mathbf{R}, L)$ and CSL-formula Φ let CTMC $M[\Phi] = (\mathcal{T}, \mathbf{R}', L)$ with $\mathbf{R}'(s,s') = \mathbf{R}(s,s')$ if $s \not\models \Phi$ and 0 otherwise. We have $M[\Phi][\Psi] = M[\Phi \vee \Psi]$.

It has been shown in [5] that for a given CTMC M and state s in M, the measure $\mathrm{Prob}^M(s, \Phi\,\mathcal{U}^I\,\Psi)$ can be calculated by means of a transient analysis of the CTMC M', which can easily be derived from M using the $[\cdot]$ operator. Let $\mathbf{\Pi}_t^M(s,s')$ denote the probability of being in state s' at time t given that the system started in state s, i.e. $\mathbf{\Pi}_t^M(s,s') = \Pr_s\{\,\sigma \in Path^M \mid \sigma@t = s'\,\}$. For intervals $I = [0, t]$ the following result applies [5]:

Theorem 4.1 $\mathrm{Prob}^M(s, \Phi\,\mathcal{U}^{[0,t]}\,\Psi) = \sum_{s' \models \Psi} \mathbf{\Pi}_t^{M[\neg\Phi \vee \Psi]}(s,s')$.

Note that $M[\neg\Phi \vee \Psi] = M[\neg(\Phi \vee \Psi)][\Psi]$, i.e., all $\neg(\Phi \vee \Psi)$-states and all Ψ-states in M are made absorbing. The former is correct since $\Phi\,\mathcal{U}^{[0,t]}\,\Psi$ 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 M has been reached (along a Φ-path) before time t, then $\Phi\,\mathcal{U}^{[0,t]}\,\Psi$ holds, regardless of which states will be visited later on.

For intervals $I = [t, t']$ with $t > 0$ the following result applies [5]:

Theorem 4.2 *For CTMC M and $0 < t \leqslant t'$:*

$$\mathrm{Prob}^M(s, \Phi\,\mathcal{U}^{[t,t']}\,\Psi) = \sum_{s' \models \Phi} \sum_{s'' \models \Psi} \mathbf{\Pi}_t^{M[\neg\Phi]}(s, s') \cdot \mathbf{\Pi}_{t'-t}^{M[\neg\Phi \vee \Psi]}(s', s'')$$

Corollary 4.1 *For CTMC M:* $\mathrm{Prob}^M(s, \Phi\,\mathcal{U}^{[t,t]}\,\Psi) = \sum_{s' \models \Phi \wedge \Psi} \mathbf{\Pi}_t^{M[\neg\Phi]}(s, s')$.

5 Structured Model-Checking of CSL

This section details how the Kronecker representation of the generator matrix can be exploited for model-checking CSL. It presents the basic algorithms for the several operators (the propositional fragment, the steady-state operator, and the time-bounded until operator). Empirical results for the presented algorithms are given in Section 8.

Since any CSL-formula Φ describes a Boolean function $f_\Phi : \mathcal{T} \to \{\mathrm{ff}, \mathrm{tt}\}$, resp. $\{0, 1\}$, model-checking amounts to a transformation of the parse tree of Φ into an explicit representation of f_Φ. This is done by a recursive post-fix traversal of the parse tree, that transforms each node of the tree into its Boolean function. There are different ways to represent Boolean functions by data structures. A bit-vector/array representation is straightforward and will be used in the following. BDDs or MDDs (multi-valued decision diagrams) are more space-efficient, but not employed here for several reasons. As the numerical computations for formulas of $\mathcal{P}_{\unlhd p}(\Phi\,\mathcal{U}^I\,\Psi)$ and $\mathcal{S}_{\unlhd p}(\Phi)$ are the dominating factor in terms of space and time consumption for CSL model-checking, the complexity for Boolean functions is of minor importance. Exchanging Boolean vectors by BDDs or MDDs is possible but is outside the scope of this paper.

5.1 *The propositional fragment of CSL*

For the propositional part of CSL, model-checking is performed as for CTL [21]. CTL model-checking using Kronecker representations has initially been presented in [40] and we use these results here. Evaluation of atomic proposition $a \in AP$ yields a Boolean function $f_a : \mathcal{T} \to \{\mathrm{ff}, \mathrm{tt}\}$. As atomic propositions are state-based propositions, their evaluation requires the values of state variables. To evaluate an atomic proposition for state $s = (x^1, \ldots, x^K)$ in a DAG representation of $\mathcal{T}$ (cf. Section 2.4) the values of state variables related to each x^k for $0 < k \leqslant K$ have to be determined. For an explicit representation of f_a by a bit-vector, each state of $\mathcal{T}$ needs to be considered. A straightforward approach uses $RS_{\mathcal{T}}()$, evaluates $s = (x^1, \ldots, x^K)$ for each path $(x^1, \ldots, x^K)$ and stores the resulting Boolean value in a bit-vector of length $|\mathcal{T}|$ at position $\Xi_{\mathcal{T}}(s)$.

As often state propositions are defined in terms of the state variables of one or a few components, this scheme can be improved by exploiting *locality of expressions*. This works as follows. The decomposition of a model into K components implies that state variables are split among components, so any component k is aware of a certain subset of state variables to define a local state x^k. Proposition a is *determined* by component k if all its state variables belong to component k. Then, $f_a(x^1, \ldots, x^k) = f_a^k(x^k)$. Proposition a is *independent* of component k if none of its state variables belongs to component k. If proposition a is *determined* by component k, then its evaluation can be performed in two steps. First, we evaluate a locally on $\mathcal{T}^k$ with $f_a^k : \mathcal{T}^k \to \{\mathrm{ff}, \mathrm{tt}\}$. Then, f_a^k is projected to f_a by assigning the same value $f_a^k(x^k)$ to all elements of subset $RS_{\mathcal{T}}(x^1, \ldots, x^k)$. This has the advantage that the second step can proceed without evaluating the atomic proposition (i.e., no consideration of state variables) and that for subsets with $|RS_{\mathcal{T}}(x^1, \ldots, x^k)| > 1$, f_a is piece-wise constant.

If atomic propositions are combined by Boolean operators and both operands are local to component k, the second step can be omitted. In this case, the Boolean operator can be performed locally and the projection to the overall state space is postponed until evaluation of a formula requires it.

Evaluating formulas from the propositional fragment of CSL only uses the DAG representation of the reachable state space $\mathcal{T}$, but does not involve the Kronecker representation. This is different for steady-state and path-formulas.

5.2 Structured analysis of steady-state formulas

The evaluation of formula $\Omega = \mathcal{S}_{\trianglelefteq p}(\Phi)$ involves numerical computations resulting in a column vector $\mathbf{q}$ (of length $|\mathcal{T}|$), such that for any state $s \in \mathcal{T}$, $s \models \Omega \iff \mathbf{q}(s) \trianglelefteq p$. Since the evaluation of $\mathbf{q}(s) \trianglelefteq p$ is trivial once $\mathbf{q}$ is known, we focus on the derivation of $\mathbf{q}$.

If $\mathcal{T}$ is strongly connected, we can employ steady-state analysis for Kronecker representations [14] to compute the solution of (1). Distribution $\boldsymbol{\pi}$ is used to derive $r = \sum_{s \in Sat(\Phi)} \boldsymbol{\pi}(s)$, and we have $\mathbf{q}(s) = r$ for all $s \in \mathcal{T}$. If $\mathcal{T}$ consists of m ($m > 1$) BSCCs $\{B_1, \ldots, B_m\}$ and transient states $B^t = \mathcal{T} \setminus \cup_{i=1}^m B_i$, we need a steady-state analysis m times, once for each BSCC, and, finally, a step to compute $\mathbf{q}[B^t]$. The partition of the state space into BSCCs and B^t is obtained by adapting the classical depth-first-search algorithm by Tarjan [44] to Kronecker representations. To that end, a function $succ : \widehat{\mathcal{T}} \to 2^{\widehat{\mathcal{T}}}$ is used which gives the set of successor states $succ(s)$ for any state $s = (x^1, \ldots, x^K) \in \widehat{\mathcal{T}}$. For a Kronecker representation (3), the successor function of $s = (x^1, \ldots, x^K)$ is defined by:

$$succ(s) = \{\, (y^1, \ldots, y^K) \mid \exists e \in \mathcal{E}, \forall 0 < k \leqslant K : \mathbf{W}_{k,e}(x^k, y^k) \neq \mathbf{0} \,\}$$

This function is used for all kinds of search algorithms on Kronecker representations, e.g., for computing $\mathcal{T}$ in [37] and for model-checking CTL path-formulas in [40]. Worst-case time complexity of Tarjan's algorithm is $O(|\mathcal{T}| + e)$, where $O(e)$ is the effort to compute all successors (outgoing edges) of states in $\mathcal{T}$. Roughly, $O(e) = O(|\mathcal{T}| \cdot |\mathcal{E}| \cdot K)$. Due to (i) relatively small values of K, (ii) special and more efficient treatment of Kronecker sums, and (iii) extreme sparseness of the involved matrices $\mathbf{W}_{k,e}$, this value is not critical in practice.

For each BSCC B_i a DAG is constructed for set Ξ_{B_i} and a steady-state analysis is carried out:

$$\boldsymbol{\pi}_{B_i} \cdot \mathbf{Q}[B_i, B_i] = 0 \quad \text{subject to} \quad \boldsymbol{\pi}_{B_i} \cdot \mathbf{1}^T = 1 \tag{7}$$

where $\boldsymbol{\pi}_{B_i}$ is the stationary distribution of BSCC B_i. This is done by the Kronecker-based solution approach of [14]. The resulting distribution $\boldsymbol{\pi}_{B_i}$ is then used to achieve a reward $r_i = \sum_{s \in Sat(\Phi)} \boldsymbol{\pi}_{B_i}(s)$. This yields the vector $\boldsymbol{\rho}$, defined by $\boldsymbol{\rho}(s) = r_i$ if state $s \in B_i$ $(0 < i \leqslant m)$ and $\boldsymbol{\rho}(s) = 0$ otherwise. For states of BSCCs, r_i also gives the resulting values for $\mathbf{q}$, i.e., $\mathbf{q}(s) = \boldsymbol{\rho}(s)$ if $s \in B_i$.

It remains to determine $\mathbf{q}(s)$ for transient states $s \in B^t$. Let $\mathbf{D}$ be the transition probability matrix of the embedded DTMC of $\mathbf{Q}$, i.e., $\mathbf{D} = \mathbf{I} + diag(\mathbf{z}^{-1}) \cdot \mathbf{Q}$ where $\mathbf{z}(s) = -\mathbf{Q}(s,s)$ if $\mathbf{Q}(s,s) \neq 0$ and 1 otherwise. The Kronecker representation of $\mathbf{D}$ is obtained from $\mathbf{R}$ as in (6). For states of B^t we compute $\mathbf{q}$ from path probabilities towards BSCCs as follows:

$$\mathbf{q}[B^t] = \sum_{i=0}^{\infty} \mathbf{D}^i[B^t, B^t] \cdot \boldsymbol{\rho}' \tag{8}$$

where $\boldsymbol{\rho}' = \mathbf{D}[B^t, \mathcal{T} \backslash B^t] \cdot \boldsymbol{\rho}$. As B^t is not strongly connected, the matrix potentials converge towards $\mathbf{0}$ and the infinite sum naturally truncates at a finite point within the numerical precision. Note that equation (8) is computed by successive matrix-vector multiplications using the equivalent recursive formulation $\mathbf{x}^{(i+1)} = \mathbf{D}[B^t, B^t] \cdot \mathbf{x}^{(i)}$ and $\mathbf{x}^{(0)} = \boldsymbol{\rho}'$, such that $\mathbf{q}[B^t] = \sum_{i=0}^{\infty} \mathbf{x}^{(i)}$ and $\mathbf{x}^{(i)}$ becomes $\mathbf{0}$ for $i \to \infty$. The use of $\mathbf{D}[B^t, B^t]$ requires a DAG with function Ξ_{B^t} to allow for matrix-vector multiplications with Kronecker representations.

According to Corollary 4.1 and Theorems 4.1, and 4.2, model-checking time-bounded until-formulas can be done using randomization [27,43] provided the CTMC is appropriately transformed. Three cases for interval $I = [t, t']$ are distinguished: (i) $0 < t = t'$, (ii) $0 = t < t'$, and (iii) $0 < t < t'$. Randomization is based on successive matrix-vector multiplications which makes it amenable to Kronecker representations, i.e., one can use matrix-vector multiplication algorithms of various kind as in [14]. Let $\mathbf{P} = \mathbf{I} + \lambda^{-1} \cdot \mathbf{Q}$ denote the matrix used in randomization with a Kronecker representation as given in (6).

Evaluating $\mathcal{P}_{\trianglelefteq p}(\Phi \, \mathcal{U}^I \, \Psi)$ requires a transformation of the CTMC that makes certain sets of states absorbing. Let $\mathcal{S}_1$ denote the set of non-absorbing states. Since $\mathbf{P}$ is given by a Kronecker representation, we need to avoid matrix transformations in computations, e.g., we select a relevant sub-matrix like $\mathbf{P}[\mathcal{S}_1, \mathcal{S}_1]$ in a matrix-vector multiplication involving some vector $\mathbf{x}$ of dimension $|\mathcal{S}_1|$ using a DAG for $\Xi_{\mathcal{S}_1}$ (as in multiplication algorithms presented in [14]). A straightforward extension uses two DAGs $\Xi_{\mathcal{S}_1}$ and $\Xi_{\mathcal{S}_2}$ to perform multiplications on arbitrary sub-matrices $\mathbf{P}[\mathcal{S}_1, \mathcal{S}_2]$ for sets $\mathcal{S}_1, \mathcal{S}_2 \subseteq \mathcal{T}$.

A key observation in [36] is to perform randomization backwards, i.e., one fixes a reward vector and evaluates equations like (5) from the right side for an unknown, arbitrary initial distribution. In combination with the preceding remarks on absorbing states, it is clear that only absorbing states that have a positive reward can have an impact on the computed values for $\mathcal{S}_1$. Let $\mathcal{S}_2$ denote the set of such absorbing states and $\boldsymbol{\rho}$ a reward vector. The impact of absorbing states on $\mathcal{S}_1$ is then given by $\mathbf{P}[\mathcal{S}_1, \mathcal{S}_2] \cdot \boldsymbol{\rho}[\mathcal{S}_2]$. We will use this observation in the following whenever absorbing states contribute positive rewards.

5.3.1 Point intervals $I = [t, t']$ with $0 < t = t'$.

This case basically requires computing the transient distribution π_t at a time point t and quantify it with 1-rewards for states in $Sat(\Phi \wedge \Psi)$, cf. Corollary 4.1. To avoid computing the transient distribution for each state $s \in \mathcal{T}$, we follow the observation of [36] and evaluate an equation like (5) from right-to-left. Let $\mathcal{S}_1 = Sat(\Phi)$ and $\mathcal{S}_2 = \mathcal{T} \backslash \mathcal{S}_1$. Reward values of states are $\boldsymbol{\rho}(s) = 1$ if $s \in Sat(\Psi) \cap \mathcal{S}_1$ and 0 otherwise. States of $\mathcal{S}_2$ are made absorbing according to Corollary 4.1, i.e., $\mathbf{q}(s) = 0$ if $s \in \mathcal{S}_2$. It remains to determine $\mathbf{q}[\mathcal{S}_1]$. Since $\boldsymbol{\rho}(s) = 0$ for $s \in \mathcal{S}_2$, states of $\mathcal{S}_2$ have no impact on the results for states in $\mathcal{S}_1$ and can be ignored. We have:

$$\mathbf{q}[\mathcal{S}_1] = \sum_{i=0}^{\infty} \beta_i(\lambda, t) \cdot \mathbf{P}^i[\mathcal{S}_1, \mathcal{S}_1] \cdot \boldsymbol{\rho}[\mathcal{S}_1] \tag{9}$$

The matrix potentials are computed by successive matrix-vector multiplication using the equivalent recursive definition $\mathbf{x}^{(i+1)} = \mathbf{P}[\mathcal{S}_1, \mathcal{S}_1] \cdot \mathbf{x}^{(i)}$ and $\mathbf{x}^{(0)} = \boldsymbol{\rho}[\mathcal{S}_1]$, such that $\mathbf{q}[\mathcal{S}_1] = \sum_{i=0}^{\infty} \beta_i(\lambda, t) \cdot \mathbf{x}^{(i)}$. The infinite summation is truncated as soon as at least one of the following truncation (termination) criteria holds.

First, the derivation of the Poisson distribution by the approach of [26] provides only a finite number of values, so the right truncation point r imposed by [26] limits the summation in a natural way. Since for $i > r, \beta_i(\lambda, t) = 0$ no further summation is necessary. Secondly, if the iteration reaches a fix-point at some step j, i.e., $\mathbf{x}^{(j+1)} = \mathbf{x}^{(j)}$, then we need not compute further iteration vectors since they remain the same. So we can directly compute the value of the infinite sum $\mathbf{q}[\mathcal{S}_1] = \sum_{i=0}^{j} \beta_i(\lambda, t) \cdot \mathbf{x}^{(i)} + (1 - \sum_{i=0}^{j} \beta_i(\lambda, t)) \cdot \mathbf{x}^{(j)}$. Thirdly, if the intermediate results are sufficiently precise to determine whether the constraint $\trianglelefteq p$ holds or not, then further iterations would only improve on the numerical precision of the transient distribution but not change the Boolean result we want to compute for the CSL formula. Note that this phenomenon is possible because the iterative computation of $\mathbf{q}[\mathcal{S}_1]$ is a summation of non-negative vectors which implies that vector entries are non-decreasing. Let $\mathbf{q}^i[\mathcal{S}_1]$ denote the sum of the first i terms. By construction we have $\mathbf{q}^i[\mathcal{S}_1] \leqslant \mathbf{q}^{i+1}[\mathcal{S}_1]$ for all $i > 0$. On the other hand, $\mathbf{P}$ is a probabilistic matrix and $\boldsymbol{\rho}(s) \in \{0, 1\}$ for all $s \in \mathcal{T}$, so $\mathbf{x}^{(j)}(s) \in [0, 1]$ for all $j \geqslant 0$. This allows us to conclude that $\sum_{i=j+1}^{\infty} \beta_i(\lambda, t) \cdot \mathbf{x}^{(i)}(s) \leqslant (1 - \sum_{i=0}^{j} \beta_i(\lambda, t)) \cdot 1 = \delta(j)$ for any $s \in \mathcal{T}$. We obtain:

$$\underbrace{\mathbf{q}^i[\mathcal{S}_1](s)}_{=lb(s)} \leqslant \mathbf{q}[\mathcal{S}_1](s) \leqslant \underbrace{\mathbf{q}^i[\mathcal{S}_1](s) + \delta(i)}_{=ub(s)}. \tag{10}$$

For formula $\Omega = \mathcal{P}_{\trianglelefteq p}(\Phi\, \mathcal{U}^I\, \Psi)$ with $p \notin [lb(s), ub(s)]$ (for all $s \in \mathcal{S}_1$) we can now terminate, since the evaluation of Ω is fully determined. Note that the computation of bounds is computationally inexpensive, since $\delta(j)$ is simply the remaining probability $1 - \sum_{i=0}^{j} \beta_i(\lambda, t)$. All termination criteria together help to minimize the number of iteration steps.

5.3.2 Anchored intervals $I = [t, t']$ for $0 = t < t'$

Let $\mathcal{S}_1 = Sat(\Phi) \cap Sat(\neg\Psi)$ and $\mathcal{S}_2 = Sat(\Psi)$. States of $\mathcal{T} \backslash \mathcal{S}_1$ are made absorbing, cf. Theorem 4.1. The reward values are $\boldsymbol{\rho}(s) = 1$ if $s \in \mathcal{S}_2$ and 0 otherwise. By construction, $\mathbf{q}(s) = 1$ if $s \in \mathcal{S}_2$ and 0 if $s \in \mathcal{T} \backslash (\mathcal{S}_1 \cup \mathcal{S}_2)$. So it remains to determine $\mathbf{q}[\mathcal{S}_1]$. Since $\boldsymbol{\rho}$ provides positive rewards for states that are not in $\mathcal{S}_1$, we cannot simply use $\boldsymbol{\rho}[\mathcal{S}_1]$ for the iteration as in the case above, but we have to account for the impact of $\boldsymbol{\rho}[\mathcal{S}_2]$. Nevertheless, the iteration can focus on $\mathcal{S}_1$ if we define $\boldsymbol{\rho}'[\mathcal{S}_1] = \mathbf{P}[\mathcal{S}_1, \mathcal{S}_2] \cdot \boldsymbol{\rho}[\mathcal{S}_2]$. Note that $\mathcal{S}_2$ is made absorbing, such that $\mathbf{P}[\mathcal{S}_2, \mathcal{S}_2] = \mathbf{I}$ and the sum of rewards gained so far by $\boldsymbol{\rho}'[\mathcal{S}_1]$ adds up in each step. Using Theorem 4.1 we get:

$$\mathbf{q}[\mathcal{S}_1] = \beta_1(\lambda, t') \cdot \boldsymbol{\rho}'[\mathcal{S}_1] + \sum_{i=2}^{\infty} \beta_i(\lambda, t') \cdot \sum_{j=0}^{i-1} \mathbf{P}^j[\mathcal{S}_1, \mathcal{S}_1] \cdot \boldsymbol{\rho}'[\mathcal{S}_1] \qquad (11)$$

Note that for i=0, the reward is $\mathbf{0}$ for $\mathcal{S}_1$, and for i=1 vector $\boldsymbol{\rho}'$ gives the reward earned by reaching $\mathcal{S}_2$ in a single transition. The matrix potentials and their summation are computed by successive matrix-vector multiplications using the equivalent recursive definition $\mathbf{x}^{(i+1)} = \mathbf{P}[\mathcal{S}_1, \mathcal{S}_1] \cdot \mathbf{x}^{(i)} + \boldsymbol{\rho}'[\mathcal{S}_1]$ and $\mathbf{x}^{(1)} = \boldsymbol{\rho}'[\mathcal{S}_1]$, such that $\mathbf{q}[\mathcal{S}_1] = \sum_{i=1}^{\infty} \beta_i(\lambda, t') \cdot \mathbf{x}^{(i)}$. The iteration is truncated by the termination criteria mentioned above. The specific observation in equation (11) is that once a Ψ-state is reached along a Φ-path, this path remains in the Ψ-state forever and contributes to the reward accumulation in each subsequent step. If we write out the summation of (11) in a matrix where index i refers to a row, we obtain a lower tridiagonal matrix of infinite dimension:

$$\begin{array}{ccccc}
0 & \cdots & & & \\
\beta_1(\lambda,t')\cdot\boldsymbol{\rho}'[\mathcal{S}_1] & 0 & \cdots & & \\
\beta_2(\lambda,t')\cdot\boldsymbol{\rho}'[\mathcal{S}_1] & \beta_2(\lambda,t')\cdot\mathbf{P}^1[\mathcal{S}_1,\mathcal{S}_1]\cdot\boldsymbol{\rho}'[\mathcal{S}_1] & 0 & \cdots & \\
\beta_3(\lambda,t')\cdot\boldsymbol{\rho}'[\mathcal{S}_1] & \beta_3(\lambda,t')\cdot\mathbf{P}^1[\mathcal{S}_1,\mathcal{S}_1]\cdot\boldsymbol{\rho}'[\mathcal{S}_1] & \beta_3(\lambda,t')\cdot\mathbf{P}^2[\mathcal{S}_1,\mathcal{S}_1]\cdot\boldsymbol{\rho}'[\mathcal{S}_1] & 0 & \cdots \\
\vdots & \vdots & \vdots & \ddots & \ddots
\end{array}$$

Table 1
Matrix of terms in equation (11)

We can also write up these terms according to columns using the exponent of $\mathbf{P}$ as an index of terms. Since values of β_i are probabilities, $\sum_{i=0}^{\infty} \beta_i(\lambda, t') = 1$ and we define weights $\gamma_i(\lambda, t') = 1 - \sum_{j=0}^{i-1} \beta_j(\lambda, t')$ to obtain $\sum_{j=i}^{\infty} \beta_j(\lambda, t') \cdot \mathbf{P}^{i-1}[\mathcal{S}_1, \mathcal{S}_1] \cdot \boldsymbol{\rho}'[\mathcal{S}_1] = \gamma_i(\lambda, t') \cdot \mathbf{P}^{i-1}[\mathcal{S}_1, \mathcal{S}_1] \cdot \boldsymbol{\rho}'[\mathcal{S}_1]$ for the sum of terms in column $i-1$. This yields the following alternative formulation of (11):

$$\mathbf{q}[\mathcal{S}_1] = \gamma_1(\lambda, t') \cdot \boldsymbol{\rho}'[\mathcal{S}_1] + \sum_{i=2}^{\infty} \gamma_i(\lambda, t') \cdot \mathbf{P}^{i-1}[\mathcal{S}_1, \mathcal{S}_1] \cdot \boldsymbol{\rho}'[\mathcal{S}_1] \qquad (12)$$

The latter variant is more efficient, since it saves a vector addition per iteration step. In addition, the test for termination uses an adapted $\delta'(i) = \sum_{j=i+1}^{\infty} \gamma_j(\lambda, t')$; note that $\gamma_j = 0$ if j exceeds the right truncation point of the computed Poisson distribution. Since the implementation of (12) iterates different vectors $\mathbf{x}$ than the one of (11), a fix-point may be reached at a different number of steps (if reached at all) and the termination criterion using the upper and lower bound is less effective due to possibly larger values of δ'. So the gain in efficiency per step can be outweighed by an increased number of iteration steps, as discussed in Section 8.

5.3.3 Non-anchored intervals $I = [t, t']$ with $0 < t < t'$

This case is the most complex one. It encompasses a randomization for $[0, t'{-}t]$ and a randomization for $[t, t]$, cf. Theorem 4.2. Let $\mathcal{S}_1 = Sat(\Phi) \cap Sat(\neg\Psi)$, $\mathcal{S}_2 = Sat(\Psi)$, and $\mathcal{S}_3 = Sat(\Phi)$. It is clear, that $\mathbf{q}(s) = 0$ if $s \notin \mathcal{S}_3$ and that we need to compute $\mathbf{q}[\mathcal{S}_3]$. We need different reward vectors for each randomization. Let $\boldsymbol{\rho}(s) = 1$ if $s \in \mathcal{S}_2$ and 0 otherwise; $\boldsymbol{\rho}(s) = 1$ thus denotes that terminal state s fulfills Ψ. Such states shall be reached within $t'{-}t$ time units via Φ-states. Hence, we define $\boldsymbol{\rho}'[\mathcal{S}_1] = \mathbf{P}[\mathcal{S}_1, \mathcal{S}_2] \cdot \boldsymbol{\rho}[\mathcal{S}_2] = \mathbf{P}[\mathcal{S}_1, \mathcal{S}_2] \cdot \mathbf{1}^T$, since $\boldsymbol{\rho}[\mathcal{S}_2] = \mathbf{1}$ (as in the case for the anchored interval above). At time point t, the reward for each state in $\mathcal{S}_1$ to reach $\mathcal{S}_2$ within time $t'{-}t$ follows from (12):

$$\boldsymbol{\rho}''[\mathcal{S}_1] = \gamma_1(\lambda, t'{-}t) \cdot \boldsymbol{\rho}'[\mathcal{S}_1] + \sum_{i=2}^{\infty} \gamma_i(\lambda, t'{-}t) \cdot \mathbf{P}^{i-1}[\mathcal{S}_1, \mathcal{S}_1] \cdot \boldsymbol{\rho}'[\mathcal{S}_1]$$

Computing the latter reward requires basically a randomization for interval $[0, t'{-}t]$. Let $\boldsymbol{\rho}''[\mathcal{S}_3](s) = 0$ for states $s \in \mathcal{S}_3 \backslash \mathcal{S}_1$. Then $\mathbf{q}$ is obtained by using the approach for a point interval (9):

$$\mathbf{q}[\mathcal{S}_3] = \sum_{i=0}^{\infty} \beta_i(\lambda, t) \cdot \mathbf{P}^i[\mathcal{S}_3, \mathcal{S}_3] \cdot \boldsymbol{\rho}''[\mathcal{S}_3] \tag{13}$$

The computation takes place in two phases using the above solutions for interval $[0, t'{-}t]$ and $[t, t]$ but with adapted reward vectors of different dimensions.

6 Eliminating Vanishing States

In addition to transitions with an exponential delay (*timed transitions*), many modeling formalisms include transitions without delay, i.e., transitions that take place immediately (*immediate transitions*). Examples of such formalisms are GSPNs [18] and stochastic process algebras with immediate transitions [31,33]. For such models the state space contains two classes of states: *vanishing* states where immediate transitions are enabled and *tangible* states where no immediate transition is enabled. Typically, immediate transitions are eliminated a priori. This results in a state space of a CTMC with only tangible states. Such elimination is possible, provided the model fulfills some quite natural conditions [18]. However, as we will clarify in this section, vanishing states may cause some problems in structured analysis approaches and in model-checking CSL.

In structured analysis it is often assumed that all immediate transitions are local, i.e., they do not participate in a synchronization. If this is the case, then vanishing states can be eliminated locally in the component state spaces. This yields state spaces $\widehat{\mathcal{T}}^k$ and matrices $\mathbf{W}_{k,e}$ as introduced above. The problem of synchronization via immediate transitions in structured analysis has been considered in [25]. In this case, stationary analysis is performed on an embedded DTMC with a state space that contains vanishing and tangible states. As this approach cannot be applied for transient analysis, we require that all synchronized transitions are timed and immediate transitions occur only locally in the components.

For CSL model-checking the problem of vanishing states has not been considered yet. We present here a first approach for structured model description which, however, can be easily adopted to a conventional description of a CTMC without any structure (i.e., the degenerated structured case with one component only).

First, we describe how vanishing states can be eliminated from component state spaces, such that the resulting component k is described by state space $\widehat{\mathcal{T}}^k$ and matrices $\mathbf{W}_{k,e}$. After state-space generation, the state space consists of $\widehat{\mathcal{T}}^k$, the set of tangible states, and $\widehat{\mathcal{V}}^k$, the set of vanishing states. We assume w.l.o.g. that states are ordered such that vanishing states precede tangible ones. Furthermore, we assume that all immediate transitions are labeled with $\tau \notin \mathcal{E}$ where τ is not used for synchronization. Consequently, the set $\mathcal{E}'$ of events of the complete model equals $\mathcal{E} \cup \{\, \tau \,\}$. Let $n_k = |\widehat{\mathcal{T}}^k|$ and $m_k = |\widehat{\mathcal{V}}^k|$. The dynamics of component k are described by several matrices. Matrix $\mathbf{V}^{tt}_{k,e} \in \mathbb{R}^{n_k \times n_k}$ contains rates of transitions labeled with $e \in \mathcal{E}$ which end in a tangible state. Similarly, matrix $\mathbf{V}^{tv}_{k,e} \in \mathbb{R}^{n_k \times m_k}$ contains rates of transitions labeled with $e \in \mathcal{E}$ which end in a vanishing state. Matrix $\mathbf{U}^{vv}_{k,\tau} \in \mathbb{R}^{m_k \times m_k}$ contains transition weights of immediate transitions which end in vanishing states and $\mathbf{U}^{vt}_{k,\tau} \in \mathbb{R}^{m_k \times n_k}$ is a matrix of transition weights of immediate transitions which end in tangible states. We assume that the weights of immediate transitions are normalized such that:

$$\mathbf{U}^{vv}_{k,\tau} \cdot \mathbf{1}^T + \mathbf{U}^{vt}_{k,\tau} \cdot \mathbf{1}^T = \mathbf{1}^T$$

This can always be achieved by normalization of the matrix rows. Remember that all weights are non-negative and in each vanishing state at least one immediate transition is enabled.

The required matrices $\mathbf{W}_{k,e}$ can be computed as

$$\mathbf{W}_{k,e} = \mathbf{V}^{tt}_{k,e} + \mathbf{V}^{tv}_{k,e} \cdot \left(\mathbf{I}_{n_k} - \mathbf{U}^{vv}_{k,\tau}\right)^{-1} \cdot \mathbf{U}^{vt}_{k,\tau} \ . \tag{14}$$

The inverse matrix exists, if the model includes no trap of immediate transitions which is assumed here. Different methods to generate $\mathbf{W}_{k,e}$ exist, often it is not necessary to really perform the matrix inversion.

Matrices $\mathbf{W}_{k,e}$ and state space $\widehat{\mathcal{T}}^k$ can then be used in structured analysis approaches as described above.

6.2 Component-wise elimination of vanishing states

The question is whether we can apply the same approach to CSL model-checking. Propositional formulas are uncritical because they can be evaluated in vanishing states like in tangible states. Slightly more complex is model checking of $\mathcal{S}_{\trianglelefteq p}(\Phi)$ with vanishing states. Assume, like in Section 5, that the states space $\mathcal{T}$ consists of m BSCCs $\{D_1, \ldots, D_m\}$ and transient states D^t. In contrast to Section 5 we now assume that D_i contains tangible and vanishing states, i.e., $D_i = B_i \cup C_i$ (and $D^t = B^t \cup C^t$) where C_i (C^t) is the set of vanishing states in the i-th BSCC and B_i (B^t) the set of transient states, respectively. $\mathcal{S}_{\trianglelefteq p}(\Phi)$ provides no problems for vanishing states inside a BSCC, because the stationary probability of each vanishing state is 0. Vanishing states in C_i thus receive the same values $\boldsymbol{\rho}(s)$ as tangible states B_i. All states from D_i (or no state in D_i) fulfill $\mathcal{S}_{\trianglelefteq p}(\Phi)$ and this can be derived from the stationary results by considering only tangible states. For vanishing states in C^t path probabilities have to be computed. For $y^k \in \widehat{\mathcal{V}}^k$ and $x^k \in \widehat{\mathcal{T}}^k$ let $AProb(y^k, x^k)$ be the probability of reaching x^k as the first tangible state when the component is in vanishing state y^k. The probability $AProb(y^k, x^k)$ can be computed as:

$$\mathbf{e}_{y^k} \cdot \sum_{i=0}^{\infty} \left(\mathbf{U}^{vv}_{k,\tau}\right)^i \cdot \mathbf{U}^{vt}_{k,\tau} \left(\mathbf{e}_{x^k}\right)^T = \mathbf{e}_{y^k} \cdot \left(\mathbf{I}_{n_k} - \mathbf{U}^{vv}_{k,\tau}\right)^{-1} \cdot \mathbf{U}^{vt}_{k,\tau} \left(\mathbf{e}_{x^k}\right)^T$$

where $\mathbf{e}_x$ is a row vector of length n_k with 1 in position x and 0 elsewhere. Let $s = (y^1, \ldots, y^K) \in C^t$, then:

$$\mathbf{q}[C^t](s) = \sum_{s'=(x^1,\ldots,x^K)\in\mathcal{T}} \left(\prod_{k=1}^{K} AP(y^k, x^k)\right) \cdot \mathbf{q}(s')$$

Knowing $\mathbf{q}(s)$ for vanishing states $\mathcal{S}_{\trianglelefteq p}(\Phi)$ can be easily evaluated, i.e., $s \models \mathcal{S}_{\trianglelefteq p}(\Phi)$ if $\mathbf{q}(s) \trianglelefteq p$.

The situation is more complex for $\mathcal{P}_{\trianglelefteq p}(\Phi\,\mathcal{U}^I\,\Psi)$, e.g., consider $\mathcal{P}_{\trianglelefteq p}(\Phi\,\mathcal{U}^I\,\Psi)$ and assume that Φ or Ψ hold in all tangible states, but in vanishing states neither Φ nor Ψ do hold. Consequently, each path which passes through vanishing states does not fulfill $\mathcal{P}_{\trianglelefteq p}(\Phi\,\mathcal{U}^I\,\Psi)$ because Φ and Ψ are not observed for a zero amount of time. This, unfortunately, is not visible after the elimination of vanishing states.

A straightforward solution to the above problem would be to say that the system cannot be observed during a time interval of length zero and therefore it does not matter which CSL formulas hold or do not hold in vanishing states because results are only interesting if they are observed for some time. This solution is easy and allows us to eliminate vanishing states locally without taking care of the CSL formulas to be checked with the model. However, the solution is against the definition of CSL path-formulas where results are defined according to the states and transitions which are observed independently of the time. Thus, we consider in the sequel the case where immediate states have to be considered for the verification of CSL path-formulas. Let $Path_{im}(s, s')$ be the set of all paths σ that start in state s, end in state s' and pass only through vanishing states.

Definition 6.1 $\mathcal{P}_{\trianglelefteq p}(\Phi\,\mathcal{U}^I\,\Psi)$ *is* invariant under immediate transitions*, iff*

- *for all* $s, s' \in \mathcal{T}$ *with* $s, s' \models \Phi \wedge s, s' \models \neg\Psi$*, and all* $\sigma \in Path_{im}(s, s')$*:* $\sigma[i] \models \Phi \wedge \sigma[i] \models \neg\Psi$ *for all* $0 < i \leqslant |\sigma|$*, and*
- *for all* $s, s' \in \mathcal{T}$ *with* $s \models \Phi \wedge s \models \neg\Psi$*,* $s' \models \Psi$ *and all* $\sigma \in Path_{im}(s, s')$*: exists* $i \leqslant |\sigma|$ *such that* $\sigma[j] \models \Phi \wedge \sigma[i] \models \neg\Psi$ *for* $j < i$ *and* $\sigma[k] \models \Psi$ *for* $k \geqslant i$*,*

where $|\sigma|$ *is the length of path* σ*.*

In general it is hard to check at state-space level whether a CSL-formula is invariant under immediate transitions or not. However, such a check can be done locally for some component k if Φ and Ψ are logical combinations of atomic propositions which are determined by k. In this case, invariance under immediate transitions can be checked using the matrices for component k only, since the functions f^k_Φ and f^k_Ψ can be evaluated for each state of k and the properties can be proved locally.

Let $\models_M$ denote the satisfaction relation $\models$ (on CSL) for M, and let $\widehat{M}$ denote the result of removing vanishing states in M. Then:

Corollary 6.1 *If* $\Omega = \mathcal{P}_{\trianglelefteq p}(\Phi\,\mathcal{U}^I\,\Psi)$ *is invariant under immediate transitions:*

$$\forall s \in \widehat{M} : s \models_{\widehat{M}} \Omega \;\; \textit{iff} \;\; s \models_M \Omega.$$

Alternatively, for formulas whose validity is determined by one component only, vanishing states can be eliminated. We consider two extreme cases for the dependency of a CSL-formula on the state of a component, viz. either it is independent or (fully) determined by such state.

Definition 6.2 Φ *is* independent *of component* k *iff*

$$(x^1, \ldots, x^{k-1}, x^k, x^{k+1}, \ldots, x^K) \models \Phi \implies (x^1, \ldots, x^{k-1}, y^k, x^{k+1}, \ldots, x^K) \models \Phi$$

for all $y^k \in \widehat{\mathcal{T}}^k$*.*

Φ *is* determined *by component* k *iff*

$$(x^1, \ldots, x^{k-1}, x^k, x^{k+1}, \ldots, x^K) \models \Phi \implies (y^1, \ldots, y^{k-1}, x^k, y^{k+1}, \ldots, y^K) \models \Phi$$

for all $y^l \in \widehat{\mathcal{T}}^l$ *and* $l \in \{1, \ldots, K\} \setminus \{k\}$*.*

As before it is in general hard to decide whether Φ is *independent* or *determined* by a component. However, we can define two simple sufficient conditions, namely if Φ contains only atomic propositions which are *determined* by component k, then Φ is *determined* by component k and if Φ contains only atomic propositions which are *independent* of component k, then Φ is *independent* of component k. Observe that the conditions always hold in cases where model checking is used to analyze the behavior of some component embedded in an environment. If $\mathcal{P}_{\trianglelefteq p}(\Phi\, \mathcal{U}^I\, \Psi)$ is independent of component k, then its validity is preserved under the elimination of vanishing states locally in k.

More interesting, though, is the case where the formula is determined by k. Elimination of vanishing states can be made independently of the interval I. We first add two new absorbing tangible states x_s^k and x_f^k with $x_s^k \models \Psi$ and $x_f^k \models \neg(\Phi \vee \Psi)$. State x_s^k is used to model a successful transition into a state where Ψ holds and state x_f^k models the situation that a state is entered where Φ and Ψ both do not hold. Both states are used to preserve the effects which occur in vanishing states after elimination of vanishing states. The set of vanishing states $\widehat{\mathcal{V}}^k$ is decomposed into three subsets, $\mathcal{S}_1$ contains all states where Φ but not Ψ holds, $\mathcal{S}_2$ contains all states where Ψ holds and $\mathcal{S}_3$ contains the remaining states. All transitions from $x \in \mathcal{S}_1$ to $y \in \mathcal{S}_2$ are substituted by a transition from x to tangible state x_s^k. The weights remain the same and are possibly added, if x has more than one successor $\mathcal{S}_2$. Similarly, all transitions from $x \in \mathcal{S}_1$ to $y \in \mathcal{S}_3$ are substituted by a transition from x to tangible state x_f^k. Furthermore, transitions from $x \in \mathcal{T}^k$ into $\mathcal{S}_2$ and $\mathcal{S}_3$ are substituted by transitions into x_s^k and x_f^k, respectively. Let $\mathbf{V}_{k,e}^{tt}$, $\mathbf{V}_{k,e}^{tv}$, $\mathbf{U}_{k,\tau}^{vv}$ and $\mathbf{U}_{k,\tau}^{vt}$ be the matrices prior to the transformations. We add states x_s^k and x_f^k as the last and second last state to $\widehat{\mathcal{T}}^k$ and obtain matrices $\mathbf{V}^{*tt}_{k,e}$ and $\mathbf{U}^{*vt}_{k,\tau}[\mathcal{S}_1, \widehat{\mathcal{T}}^k]$ on the new state space with:

$$\mathbf{V}^{*tt}_{k,e} = \begin{pmatrix} \mathbf{V}^{tt}_{k,e} & \mathbf{V}^{tv}_{k,e}[\widehat{\mathcal{T}}^k, \mathcal{S}_2] \cdot \mathbf{1}^T & \mathbf{V}^{tv}_{k,e}[\widehat{\mathcal{T}}^k, \mathcal{S}_3] \cdot \mathbf{1}^T \\ \mathbf{0} & 0 & 0 \\ \mathbf{0} & 0 & 0 \end{pmatrix}$$

$$\mathbf{U}^{*vt}_{k,\tau}[\mathcal{S}_1, \widehat{\mathcal{T}}^k] = \Big(\mathbf{U}^{vt}_{k,\tau}[\mathcal{S}_1, \widehat{\mathcal{T}}^k] \ \mathbf{U}^{vv}_{k,\tau}[\mathcal{S}_1, \mathcal{S}_2] \cdot \mathbf{1}^T \ \mathbf{U}^{vv}_{k,\tau}[\mathcal{S}_1, \mathcal{S}_3] \cdot \mathbf{1}^T \Big)$$

The remaining matrices need not be modified. Matrices $\mathbf{W}_{k,e}$ can then be computed by:

$$\mathbf{W}^*_{k,e} = \mathbf{V}^{*tt}_{k,e} + \mathbf{V}^{tv}_{k,e}[\mathcal{T}, \mathcal{S}_1] \cdot \left(\mathbf{I}_{n_k} - \mathbf{U}^{vv}_{k,\tau}[\mathcal{S}_1, \mathcal{S}_1]\right)^{-1} \cdot \mathbf{U}^{*vt}_{k,\tau}[\mathcal{S}_1, \widehat{\mathcal{T}}^k] \quad (15)$$

Observe that $\mathbf{W}_{k,e}$ has dimension $n_k + 2$ instead of n_k because the two additional states have been added to make immediate events visible for CTMC analysis. Since the two additional states make changes according to formulas Φ and Ψ visible in the tangible state space, the following corollary holds.

Let M^k denote the CTMC obtained from M by representing its generator matrix in a structured form using matrices $\mathbf{W}_{l,e}$ for components $l \neq k$ and matrices $\mathbf{W}^*_{k,e}$ for component k. Then:

Corollary 6.2 *If $\Omega = \mathcal{P}_{\trianglelefteq p}(\Phi\, \mathcal{U}^I\, \Psi)$ is determined by component k, then*

$$\forall s \in M^k : s \models_M \Omega \ \textit{ iff } \ s \models_{M^k} \Omega.$$

7 On the Role of Bisimulations

As shown in [5], the validity of CSL-formulas is preserved under stochastic bisimulation equivalence (i.e., lumping). Thus it is possible to first build a minimal representation of a CTMC up to bisimulation and then model-check the reduced instead of the entire CTMC. Moreover, stochastic bisimulation is a congruence with respect to the composition of components by synchronization [9,31]. Bisimulation minimization can thus be applied in a compositional way. This approach can be naturally combined with structured analysis techniques where the generator matrix is represented as the sum of Kronecker products of component matrices. Components are first reduced modulo bisimulation, and a reduced representation of the generator matrix of the entire model is then given in a structured form by replacing the entire component matrices by the matrices for the respective reduced component. This section adapts this approach to model-checking CSL.

We first recall some notions and results from [5] where we confine ourselves to bisimulations over sets of atomic propositions. As we consider structured CTMC descriptions, we start with bisimulation at a component level as our basic notion.

Definition 7.1 *For component k with state space $\widehat{\mathcal{T}}^k$ as part of a composed model consisting of components 1 through K and set of atomic propositions AP, let $\simeq_{AP}$ be defined as the equivalence relation on $\widehat{\mathcal{T}}^k$ such that for all $x^k, y^k \in \widehat{\mathcal{T}}^k$ and all $a \in AP$: $x^k \simeq_{AP} y^k$ iff*

$$(x^1, \ldots, x^k, \ldots, x^K) \models a \;\Leftrightarrow\; (x^1, \ldots, y^k, \ldots, x^K) \models a \;\; \textit{for all} \;\; x^l \in \widehat{\mathcal{T}}^l, l \neq k.$$

Note that $\simeq_{AP}$ depends on the embedding realized by the components 1 through K. In many cases, it is possible to define $\simeq$ independent from the environment, e.g., if all propositions from AP are either determined by k or are independent of k. We now define Φ-bisimulation, a somewhat simplified version of F-bisimulation [4]. For CSL-formula Φ, let AP_Φ denote the set of atomic propositions in Φ.

Definition 7.2 *A Φ-bisimulation is an equivalence relation Rel on $\widehat{\mathcal{T}}^k$ such that whenever $(x, y) \in Rel$, then:*

$$x \simeq_{AP_\Phi} y \;\; \textit{and} \;\; \sum_{z \in \mathcal{C}} \mathbf{W}_{k,e}(x, z) = \sum_{z \in \mathcal{C}} \mathbf{W}_{k,e}(y, z)$$

for all $\mathcal{C} \in \widehat{\mathcal{T}}^k/Rel$ and $e \in \mathcal{E}_{L,k} \cup \mathcal{E}_{S,k}$.

States x and y are Φ-bisimilar iff there exists a Φ-bisimulation Rel that contains (x, y).

The largest Φ-bisimulation, denoted by $\sim_\Phi$, can be defined in the standard way [9]. Algorithms for computing this equivalence are given in [11,34]; the algorithm in [11] has been implemented in the APNN-toolbox [15].

Let $\tilde{n}_k$ ($\leqslant n_k$) be the number of equivalence classes of $\sim_\Phi$. The Φ-bisimilar quotient of component k has state space $\widehat{\mathcal{T}}^k/\sim_\Phi = \{0, \ldots, \tilde{n}_k - 1\}$, and is characterized by matrices $\widetilde{\mathbf{W}}_{k,e} \in \mathbb{R}^{\tilde{n}_k \times \tilde{n}_k}$ defined element-wise by:

$$\widetilde{\mathbf{W}}_{k,e}([x]_{\sim_\Phi}, [y]_{\sim_\Phi}) = \sum_{z \in [y]_{\sim_\Phi}} \mathbf{W}_{k,e}(x, z). \tag{16}$$

Reduced components may be computed for all components. This yields a CTMC described by (3) where matrices $\mathbf{W}_{k,e}$ are replaced by $\widetilde{\mathbf{W}}_{k,e}$. The resulting state space has $\prod_{k=1}^{K} \tilde{n}_k$ instead of $\prod_{k=1}^{K} n_k$ potential reachable states.

The following result follows from the facts that bisimulation preserves the validity of CSL-formulas [5] and is a congruence for the composition operators used here [9]. Let $M/\sim_\Phi$ denote the CTMC obtained from M by replacing each component in M by its Φ-bisimilar equivalent. Then:

Theorem 7.1 *For all* $x \in M : x \models_M \Phi$ *iff* $[x]_{\sim_\Phi} \models_{M/\sim_\Phi} \Phi$.

Thus, model-checking CSL-formulas can be carried out on the modularly obtained reduced representation of the structured CTMC under consideration.

If several formulas are to be verified, several reduced representations have to be computed. Fortunately, this can often be made more efficient using the following observation.

Corollary 7.1 *For CSL-formulas* $\Phi, \Psi : AP_\Phi \subseteq AP_\Psi \Rightarrow \sim_\Phi \subseteq \sim_\Psi$.

Proof. Since $AP_\Phi \subseteq AP_\Psi$ and the transition weights/rates are the same, every Ψ-bisimulation is also a Φ-bisimulation.

Assume that formulas $\Phi_1, \ldots, \Phi_M$ have to be checked. The above result suggests to compute $\sim_{\Phi_i}$ before $\sim_{\Phi_j}$ if $AP_{\Phi_i} \subset AP_{\Phi_j}$. $\sim_{\Phi_j}$ can then be computed by refining the equivalence classes of $\sim_{\Phi_i}$. This often makes the bisimulation computations and the generation of reduced components more efficient.

To summarize the results of this section, we present the following algorithm:

Input: Structured description with components $1, \ldots, K$ and $\mathcal{T}$
        Formulas $\Phi_1, \ldots, \Phi_M$.
  for ($i = 1$ to M) do
    for ($k = 1$ to K) do
      compute $\sim_{\Phi_i}$ for component k
      generate matrices $\widetilde{\mathbf{W}}_{k,e}$ according to $\sim_{\Phi_i}$ using (16)
    check formula Φ_i on the reduced CTMC to construct $Sat(\Phi_i)$

To model-check formulas in the reduced model, the structured algorithms of Section 5 are used that exploit the reduced Kronecker representation. The reduction factor depends on both the structure of the components and the formulas.

A different strategy would be to exploit bisimulations among different components rather than on individual components (as we did here). For instance, it is well known that for identical components which are symmetrically connected, bisimulations can be generated by abstracting from the order of the

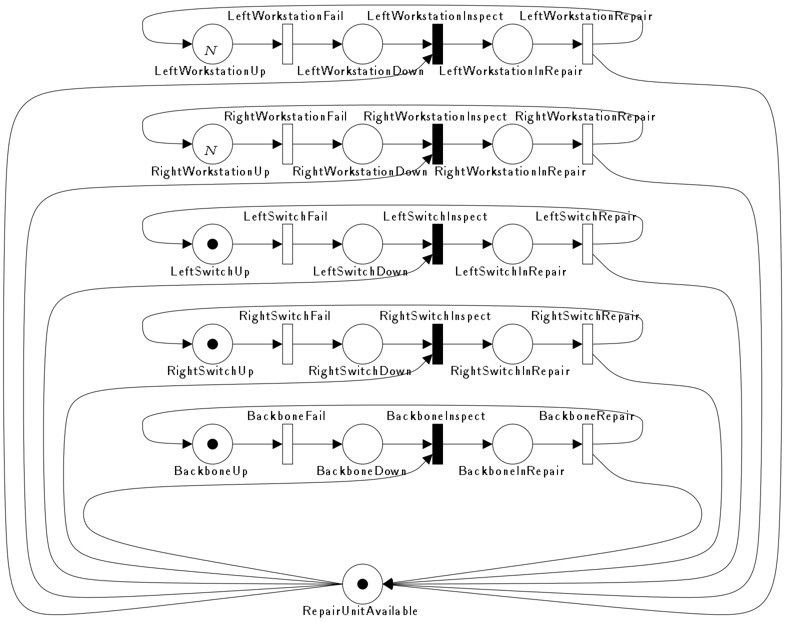

Fig. 2. GSPN model of the workstation cluster (taken from [29])

components. Such approach can be integrated with the structured analysis approach, see [10].

8 An Application Example

We illustrate our approach by model-checking a cluster of workstations consisting of two sub-clusters connected via a backbone connection (taken from [29]), and model-checking a simple tandem queueing network (taken from [32]). All reported experiments are carried out on a Sun enterprise 250 with 400 MHz CPU and 2 Gb main memory running Solaris 5.7. The reported execution times are all wall clock times.

8.1 Workstation cluster

Each sub-cluster consists of N workstations, connected in a star topology with a central switch that provides the interface to the backbone. Each system of the component (workstations, switches, and backbone) can break down. There is single repair unit that takes care of repairing failed components. The computing power of the cluster is over-dimensioned, in order to be able to accommodate varying levels of traffic volume, as well as to cope with component failures. The system operation is subject to the following informal constraints [29]:

- In order to provide *minimum* quality of service (QoS), at least k ($k < N$) workstations have to be operational, and these workstations have to be connected to each other via operational switches. If in each sub-cluster the number of operational workstations drops below k then an operational backbone is required to ensure that in total at least k operational workstations are still connected to provide minimum service.
- *Premium* quality of service requires at least N operational workstations, with the same connectivity constraints as mentioned above.

The CTMC of the system model is obtained from the generalized stochastic Petri net (GSPN) depicted in Figure 2 using similar assumptions as in [29], so the immediate transitions are modeled by timed events with short delays in order to obtain CTMCs of same size. For simplicity, we assume that transitions rates are not state-dependent. The first two "rows" in the GSPN represent the two groups of workstations. In each group, individual workstations can fail (transition WorkstationFail). Once failed, the workstation is Down, and the repair unit is needed to repair the workstation. The repair unit is depicted at the bottom of the figure. If a repair unit (token) is Available, repair starts almost immediately (transition WorkstationInspect), the workstation is InRepair. Once repaired successfully (transition WorkstationRepair), the workstation is Up again. The evolution of the other components of the system is very similar. The next two "rows" in Figure 2 represent the behavior of the two switches, and the last row represents the backbone.

The CSL-formulas for the workstation cluster example are constructed from atomic propositions that range over the individual markings of the place of the GSPN. For place P and natural number n, proposition $\#\mathsf{P} \trianglelefteq n$ is valid if and only if place P contains $\trianglelefteq n$ tokens. Appropriate abbreviations for the case study at hand are listed in Table 2.

Table 2
Abbreviations of basic formulas for the workstation cluster

$$
\begin{aligned}
\mathit{LeftOperational}_i &= (\#\mathsf{LeftWorkstationUp} \geqslant i) \wedge (\#\mathsf{LeftSwitchUp} > 0)\\
\mathit{RightOperational}_i &= (\#\mathsf{RightWorkstationUp} \geqslant i) \wedge (\#\mathsf{RightSwitchUp} > 0)\\
\mathit{Conn} &= (\#\mathsf{LeftSwitchUp} > 0) \wedge (\#\mathsf{BackboneUp} > 0)\\
&\quad \wedge(\#\mathsf{RightSwitchUp} > 0)\\
\mathit{Operational}_i &= (\#\mathsf{LeftWorkstationUp} + \#\mathsf{RightWorkstationUp} \geqslant i) \wedge \mathit{Conn}\\
\mathit{Minimum} &= \mathit{LeftOperational}_k \vee \mathit{RightOperational}_k \vee \mathit{Operational}_k\\
\mathit{Premium} &= \mathit{LeftOperational}_N \vee \mathit{RightOperational}_N \vee \mathit{Operational}_N
\end{aligned}
$$

We decompose the model into three components: the left workstations, the right workstations, and the remaining parts (switches, backbone and repair unit). We consider the model for increasing numbers of N (up to 1024) workstations. Note that only the size of the state space of the first two components increases in dimension while the rest remains constant. For analysis, the Kronecker representation and the set of reachable states $\mathcal{T}$ are generated in a pre-processing step. We apply the technique of [16]: first, state spaces of components are generated in isolation to achieve a Kronecker representation of $\mathbf{Q}$ (as given by equation (3)). Then, for each component, the reachability graph is minimized using an inverse bisimulation [12]. The set of reachable states is generated on the minimized components. For this model this procedure results in an aggregated overall state space with 3 states for all considered values of N, i.e., there are three equivalence classes reachable in the aggregated model. Inverse bisimulations are equivalence relations that preserve reachability. This means that from the set of reachable equivalence classes in the minimized model, the set of reachable states can be easily recreated by a dis-aggregation step where a reachable aggregated state (i.e., a reachable equivalence class) is substituted by all detailed states belonging to the equivalence class. After a dis-aggregation step of its DAG representation, we achieve a DAG of $RS_{\mathcal{T}}()$. The generation of a Kronecker representation and $RS_{\mathcal{T}}()$ takes less than 10 sec for all considered values of N. The first three columns of Table 3 list the values considered for N, the size of the state space, and the size of the Kronecker representation (in Kb), respectively. We model-check $\Omega_s = \mathcal{S}_{\geqslant 0.7}(\mathit{Premium})$ and three different formulas of type $\mathcal{P}_{\trianglelefteq p}(\Phi\,\mathcal{U}^I\,\Psi)$. We focus on the latter since these formulas are most complex. Let $k = 3$.

As the underlying CTMC of the workstation cluster is strongly connected, model-checking Ω_s reduces to the computation of a stationary distribution for a single BSCC $B = \mathcal{T}$. This is performed by a Jacobi iteration (with a relaxation factor of 0.9) for a model with $N = 1024$. The iteration vector requires 302.4 Mb, i.e., space consumption for iteration vectors clearly dominates memory requirements for the CTMC. A single iteration step takes 113.2 sec, resulting in a total time of $100,985$ sec to converge to a maximum residual of $1.13 \cdot 10^{-5}$ in 890 iteration steps.

We consider $\Omega_1 = \mathcal{P}_{<0.1}(\mathrm{tt}\,\mathcal{U}^I\,\mathit{Minimum})$ with $I = [0, 85]$ to allow for a direct comparison with results presented in [29] and a corresponding case study with PRISM [1]. It asserts that the chance that QoS drops below minimum quality within 85 time units is less than 10%. To model-check this formula, one randomization is required (case $I = [0, t]$) with an iterative procedure that

[1] http://www.cs.bham.ac.uk/~dxp/prism/cluster.html

considers states in $\mathcal{S}_1 = \mathcal{T} \backslash Sat(\neg Minimum)$ since $\Phi = \mathrm{tt}$ for all states $s \in \mathcal{T}$. The fourth and fifth column of Table 3 list the space requirements (to represent set $\mathcal{S}_1$ for Ω_1; in Kb) and the time (in seconds) per iteration for verifying Ω_1.

Formula $\Omega_2 = \neg Minimum \Rightarrow \mathcal{P}_{<0.3}(\mathrm{tt}\, \mathcal{U}^I \neg Minimum)$ for $I = [2, 2]$ is build upon a logical implication that involves the solution of a formula $\mathcal{P}_{\trianglelefteq p}(\Phi\, \mathcal{U}^I\, \Psi)$ as an intermediate result. It is an interval of kind $I = [t, t]$ with $t = 2$, where the iteration considers a set $\mathcal{S}_1 = \mathcal{T}$ as $\Phi = \mathrm{tt}$ for all states $s \in \mathcal{T}$. Since $\mathcal{S}_1$ is slightly larger for Ω_2 than for Ω_1, computation times increase but remain within the same order of magnitude (cf. Table 3, sixth and seventh column).

Finally, we consider formula $\Omega_3 = \mathcal{P}_{>0.8}(Minimum \mathcal{U}^I\, Premium)$, where $I = [t, t'] = [20, 40]$. (These formulas can neither be checked with the current implementation of ETMCC [32] nor with PRISM [41].) This formula requires a double randomization: one for the interval $[0, t'-t] = [0, 20]$ and another for the interval $[0, t] = [0, 20]$. Randomization in both phases is carried out with different sets $\mathcal{S}_1$ and $\mathcal{S}_3$, e.g., for $N = 1024$ we need 1029 steps for the first phase and 895 steps for the second phase. The entire computation requires 112,757 seconds. In the second phase, the iteration does not reach a fix-point but stops since the resulting Boolean values are determined for all states.

N	$\lvert\mathcal{T}\rvert$	$\otimes$	Ω_1		Ω_2		Ω_3	
			$RS_{\mathcal{S}_1}()$	time	$RS_{\mathcal{S}_1}()$	time	$RS_{\mathcal{S}_1}()$	time
32	38,676	67.0	10.5	0.02	3.1	0.03	30.3	0.02
256	2,373,652	104.5	78.6	2.61	20.1	3.57	1385.1	2.43
512	9,465,876	190.5	156.4	10.63	39.6	14.7	5390.9	13.61
1024	37,806,100	362.5	312.1	43.55	78.5	59.90	21181.2	58.42

Table 3
Experimental data for the workstation cluster

8.3 A simple tandem queue

This example is taken from [32]. It is a simple tandem queuing network of a $M/Cox_2/1$ queue and a $M/M/1$ queue which both have finite capacity c, such that a blocking phenomenon can be observed. We use this model to illustrate the effect of the different termination criteria we employ. Table 4 gives the number of iteration steps to evaluate $\mathcal{P}_{>p}(\mathrm{tt}\, \mathcal{U}^I\, full)$ with $I = [0, t]$ for different values of p and t but fixed capacity $c = 255$ which implies $|\mathcal{T}| = 130,816$. The atomic proposition *full* identifies situations in which both queues reach their capacities. The iterative computation of vectors $x^{(i)}$ reaches a fix point after 526 steps independently from values of p and t if we use the implementation of (11). We exercised the tandem model for different values of p and different time horizons t and both implementations of (11) and (12). Results are given in Table 4, where columns # steps either refer to the implementation of (11) or (12) respectively. In this model, the termination criterion appears to be rather insensitive to the selection of p but very sensitive to the selection of t in case of (11) and and neither sensitive to p nor t for (12). If $t > 20$ the mode of the Poisson distribution exceeds 526 and the fix point termination criterion turns out to be the most effective in case of (11), for small values of t the decision criterion is effective and causes termination much ahead of the right truncation point of Fox and Glynn [26]. It is interesting to see, that for this example the number of steps for the iteration according to (11) is significantly less than for (12) which is more efficient per step. In summary, we see that the variety of different termination criteria is useful and it is not clear in general whether (11) or (12) is more efficient in terms of computation time, as far as space is concerned, (12) is clearly superior to (11) since it uses one vector less.

p	t	# steps (11)	# steps (12)	p	t	# steps (11)	# steps (12)
0.1	5	173	228	0.3	5	165	228
0.1	10	315	394	0.3	10	317	394
0.1	15	375	552	0.3	15	389	552
0.1	20	492	706	0.3	20	492	706
0.7	5	169	228	0.9	5	174	228
0.7	10	318	394	0.9	10	332	394
0.7	15	413	552	0.9	15	481	552
0.7	20	526	706	0.9	20	526	706

Table 4
Tandem Queueing Network with Blocking

Finally, we consider the tandem network for a large configuration with $c = 4095$ to achieve results comparable to [36]. The time to generate the Kronecker representation and $RS_{\mathcal{T}}()$ is 242 seconds; the bisimulation approach is not applied due to the size of component state spaces, i.e. the first queue results in a state space with 8191 states, the second has 4096 states. The whole model has $|\mathcal{T}| = 33,550,336$ states. We compute a solution of formula $\Omega = \mathcal{P}_{>0.5}(\mathrm{tt}\,\mathcal{U}^I\, \mathit{full})$ for interval $I = [0, t]$ with different values of t. Table 5 give results obtained over a range of parameter values t (column 1). Column 2 gives the number of iteration steps, column 3 the time per step in seconds, and the last column gives the total time in seconds using the implementation of (11).

The DAG $RS_{\mathcal{S}_1}()$ encodes 33,542,144 states, consists of 2 nodes and uses 163844 bytes. The randomization procedure requires several vectors, where

t	# steps	time per step	total time
5	168	41.62	7,217
10	313	44.80	14,236
15	462	41.85	19,552
20	607	42.64	26,100
25	753	41.86	31,737
30	881	41.96	37,205

Table 5
Tandem Queueing Network with Blocking

each uses 268,337,152 bytes (double precision), which creates the bottleneck for this approach in terms of space. For the considered values of t, termination happens in all cases due to the fact that intermediate results are sufficient to determine the formula. This kind of termination is effective in case of relatively small time horizons or a slow rate of convergence.

9 Conclusions

This paper considers a new approach for model checking CSL formulas based on a combination of CSL model checking algorithms and structured analysis approaches for large Markov chains. It is shown that with this combination CSL model checking can be applied for fairly large models which are structured as the parallel composition of interacting components. We presented in detail the evaluation of $\mathcal{P}_{\trianglelefteq p}(\Phi\, \mathcal{U}^I\, \Psi)$ in structured models which is the most complex formula. Furthermore, we show that due to the compositional model structured state space reduction by minimization of components using bisimulation and the elimination of vanishing states can be performed very efficiently.

Since the presented algorithms have been implemented and integrated in the APNN toolbox, they are ready to be used for realistic examples. First results are encouraging, but there is still the need to perform more experiments to compare the different realizations of CSL model checking algorithms that have been proposed.

References

[1] A. Aziz, K. Sanwal, V. Singhal and R. Brayton. Verifying continuous time Markov chains. In R. Alur and T.A. Henzinger (eds), *Computer-Aided Verification*, LNCS 1102, pp. 269–276, Springer-Verlag, 1996.

[2] 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.

[3] I. Bahar, E. Frohm, C. Gaona, G. 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.

[4] C. Baier, B.R. Haverkort, H. Hermanns and J.-P. Katoen. On the logical characterisation of performability properties. In U. Montanari, J.D.P. Rolim, and E. Welzl (eds.), *Automata, Languages, and Programming (ICALP)*, LNCS 1853, pp. 780–792, Springer-Verlag, 2000.

[5] C. Baier, B.R. Haverkort, H. Hermanns and J.-P. Katoen. Model checking continuous-time Markov chains by transient analysis. In E.A Emerson and A.P. Sistla (eds), *Computer Aided Verification*, LNCS 1855, pp. 358–372, Springer-Verlag, 2000.

[6] C. Baier, J.-P. Katoen and H. Hermanns. Approximate symbolic model checking of continuous-time Markov chains. In J.C.M. Baeten and S. Mauw (eds), *Concurrency Theory*, LNCS 1664, pp. 146–162, Springer-Verlag, 1999.

[7] R. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Tr. on Comp.*, **35**(8): 677–691, 1986.

[8] P. Buchholz. *Die Strukturierte Analyse Markovscher Modelle (in German)*. IFB 282. Springer, 1991.

[9] P. Buchholz. Markovian process algebra: composition and equivalence. In U. Herzog and M. Rettelbach (eds), *Proc. of the 2nd Work. on Process Algebras and Perf. Modelling*, pages 11–30. Arbeitsberichte des IMMD, University of Erlangen, no. 27, 1994.

[10] P. Buchholz. Hierarchical Markovian models: symmetries and reduction. *Perf. Ev.*, **22**:93–110, 1995.

[11] P. Buchholz. Efficient computation of equivalent and reduced representations for stochastic automata. *Int. J. Comp. Systems Science and Eng.*, **15**(2):93–103, 2000.

[12] P. Buchholz. Hierarchical structuring of superposed GSPNs. *IEEE Trans. on Softw. Eng.*, **25**(2):166–181, 1999.

[13] P. Buchholz. Structured analysis approaches for large Markov chains. *Applied Numerical Mathematics*, **31**(4):375–404, 1999.

[14] P. Buchholz, G. Ciardo, S. Donatelli, and P. Kemper. Complexity of Kronecker operations and sparse matrices with applications to the solution of Markov models. *INFORMS J. on Computing*, **12**(3):203–222, 2000.

[15] P. Buchholz, M. Fischer, P. Kemper, and C. Tepper. New features in the APNN toolbox. In P. Kemper (ed), *Tools of Aachen 2001 Int. Multiconference on Measurement, Modeling and Evaluation of Computer-Communication Systems*, pages 62–68. Universität Dortmund, Fachbereich Informatik, Forschungsbericht Nr. 760, 2001.

[16] P. Buchholz and P. Kemper. Efficient computation and representation of large reachability sets for composed automata. *Discrete Event Dynamic Systems - Theory and Applications*, **12**(3):265-286, 2002.

[17] J. Campos, M. Silva, and S. Donatelli. Structured solution of asynchronously communicating stochastic modules. *IEEE Trans. on Softw. Eng.*, **25**(2):147–165, 1999.

[18] G. Chiola, M. Ajmone-Marsan, G. Balbo, and G. Conte. Generalized stochastic Petri nets: a definition at the net level and its implications. *IEEE Trans. on Softw. Eng.*, **19**(2):89–107, 1993.

[19] G. Ciardo. Distributed and structured analysis approaches to study large and complex systems. In E. Brinksma, H. Hermanns, and J.-P. Katoen (eds), *Lectures on Formal Methods and Perf. Analysis*, LNCS 2090, pages 344–374. Springer-Verlag, 2001.

[20] G. Ciardo and A. S. Miner. A data structure for the efficient Kronecker solution of GSPNs. In P. Buchholz and M. Silva (eds), *Proc. 8th Int. Workshop on Petri Nets and Perf. Models*, pages 22–31. IEEE CS-Press, 1999.

[21] E. Clarke, E. Emerson and A. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. on Programming Languages and Systems*, **8**: 244–263, 1986.

[22] E. Clarke, M. Fujita, P.C. McGeer and J.C-Y. Yang. Multi-terminal binary decision diagrams: an efficient data structure for matrix representation. *Formal Methods in System Design*, **10**(2/3): 149–169, 1997.

[23] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking.* MIT Press, 1999.

[24] S. Donatelli. Superposed stochastic automata: a class of stochastic Petri nets amenable to parallel solution. *Perf. Ev.*, **18**:21–36, 1994.

[25] S. Donatelli and P. Kemper. Integrating synchronization with priority into a Kronecker representation. *Perf. Ev.*, **44**(1-4):73–96, 2001.

[26] B.L. Fox and P.W. Glynn. Computing Poisson probabilities. *Comm. ACM*, **31**:440–445, 1988.

[27] 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.

[28] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing* **6**: 512–535, 1994.

[29] B.R. Haverkort, H. Hermanns, and J.-P. Katoen. The use of model checking techniques for quantitative dependability evaluation. In *IEEE Symp. on Reliable Distr. Sys.*, pp. 228–238, IEEE CS Press, 2000.

[30] B. R. Haverkort and K.S. Trivedi. Specification techniques for Markov reward models. *Discrete Event Systems: Theory and Application*, **3**:219–247, 1993.

[31] H. Hermanns, U. Herzog and J.-P. Katoen. Process algebra for performance evaluation. *Th. Comp. Sc.*, **274**:43–87, 2002.

[32] H. Hermanns, J.-P. Katoen, J. Meyer-Kayser and M. Siegle. A Markov chain model checker. In S. Graf and M. Schwartzbach (eds), *Tools and Algorithms for the Construction and Analysis of Systems*, LNCS 1785, pp. 347–362, Springer-Verlag, 2000.

[33] H. Hermanns, M. Rettelbach and T. Weiss. Formal characterisation of immediate actions in SPA with non-deterministic branching. *The Comp. J.*, **38**(7): 530–542, 1995.

[34] H. Hermanns and M. Siegle. Bisimulation algorithms for stochastic process algebras and their BDD-based implementation. In J.-P. Katoen (ed), *Formal Methods for Real-Time and Probabilistic Systems*, LNCS 1601, pp. 244–265, Springer-Verlag, 1999.

[35] J. Hillston. *A Compositional Approach to Performance Modelling.* Cambridge University Press, 1996.

[36] J.-P. Katoen, M. Kwiatkowska, G. Norman, and D. Parker. Faster and symbolic CTMC model checking. In L. de Alfaro and S. Gilmore (eds), *Process Algebra and Probabilistic Methods*, LNCS 2165, pp. 23–38. Springer-Verlag, 2001.

[37] P. Kemper. Reachability analysis based on structured representations. In J. Billington and W. Reisig (eds), *Application and Theory of Petri Nets 1996*, LNCS 1091, pp. 269-288. Springer-Verlag, 1996.

[38] P. Kemper. Numerical analysis of superposed GSPNs. *IEEE Trans. on Softw. Eng.*, **22**(9):615–628, 1996.

[39] P. Kemper. Transient analysis of superposed GSPNs. *IEEE Trans. on Softw. Eng.*, **25**(2):182–193, 1999.

[40] P. Kemper and R. Lübeck. Model checking based on Kronecker algebra. Technical Report 669, Fachbereich Informatik, Universität Dortmund (Germany), 1998.

[41] M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic symbolic model checking with PRISM: a hybrid approach. In J.-P. Katoen and P. Stevens (eds), *Tools and Algorithms for the Construction and Analysis of Systems*, LNCS 2280, pp. 52–67, Springer-Verlag, 2002.

[42] B. Plateau. On the stochastic structure of parallelism and synchronisation models for distributed algorithms. *Perf. Ev. Review*, **13**:142–154, 1985.

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

[44] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM. J. Comp.*, **1**:146–160, 1972.

Appendix: Kronecker Operations for Matrices

Kronecker products and sums are widely used in different branches of mathematics and system analysis. Their application for the analysis of stochastic automata dates back to Plateau [42]. This appendix gives a brief overview of the operations.

Definition 9.1 *The Kronecker product of two matrices* $\mathbf{A} \in \mathbb{R}^{r_1 \times c_1}$ *and* $\mathbf{B} \in \mathbb{R}^{r_2 \times c_2}$ *is defined as* $\mathbf{C} = \mathbf{A} \otimes \mathbf{B}$, $\mathbf{C} \in \mathbb{R}^{r_1 r_2 \times c_1 c_2}$, *where*

$$\mathbf{C}(i_1 r_2 + i_2, j_1 c_2 + j_2) = \mathbf{A}(i_1, j_1)\mathbf{B}(i_2, j_2)$$

($0 \leqslant i_x < r_x,\ 0 \leqslant j_x < c_x,\ x \in \{1, 2\}$*).*

The Kronecker sum of two matrices $\mathbf{A} \in \mathbb{R}^{n_1 \times n_1}$ *and* $\mathbf{B} \in \mathbb{R}^{n_2 \times n_2}$ *is defined as*

$$\mathbf{A} \oplus \mathbf{B} = \mathbf{A} \otimes \mathbf{I}_{n_2} + \mathbf{I}_{n_1} \otimes \mathbf{B} \ .$$

For Kronecker products and sums the following properties hold if the ordinary matrix products are defined (see [42,43]).

(1) $(\mathbf{A} \otimes \mathbf{B}) \otimes \mathbf{C} = \mathbf{A} \otimes (\mathbf{B} \otimes \mathbf{C})$ and $(\mathbf{A} \oplus \mathbf{B}) \oplus \mathbf{C} = \mathbf{A} \oplus (\mathbf{B} \oplus \mathbf{C})$ associativity,
(2) $(\mathbf{AB}) \otimes (\mathbf{CD}) = (\mathbf{A} \otimes \mathbf{C})(\mathbf{B} \otimes \mathbf{D})$ compatibility with multiplication,
(3) $(\mathbf{A} + \mathbf{B}) \otimes (\mathbf{C} + \mathbf{D}) = \mathbf{A} \otimes \mathbf{C} + \mathbf{A} \otimes \mathbf{D} + \mathbf{B} \otimes \mathbf{C} + \mathbf{B} \otimes \mathbf{D}$ distributivity over addition,
(4) $(\mathbf{A} \otimes \mathbf{B})^T = \mathbf{A}^T \otimes \mathbf{B}^T$ compatibility with transposition,
(5) $(\mathbf{A} \otimes \mathbf{B})^{-1} = \mathbf{A}^{-1} \otimes \mathbf{B}^{-1}$ compatibility with inversion.

Since Kronecker products and sums are associative, the following generalisations are well-defined.

$$\bigotimes_{j=1}^{J} \mathbf{A}_j = \mathbf{A}_1 \otimes \bigotimes_{j=2}^{J} \mathbf{A}_j \text{ and } \bigoplus_{j=1}^{J} \mathbf{A}_j = \mathbf{A}_1 \oplus \bigoplus_{j=2}^{J} \mathbf{A}_j$$

Generalised Kronecker products and sums can be transformed into some standardized form applying the above properties several times.

$$\bigotimes_{j=1}^{J} \mathbf{A}_j = \prod_{j=1}^{J} \mathbf{I}_{l_j} \otimes \mathbf{A}_j \otimes \mathbf{I}_{u_j} \text{ and } \bigoplus_{j=1}^{J} \mathbf{A}_j = \sum_{j=1}^{J} \mathbf{I}_{l_j} \otimes \mathbf{A}_j \otimes \mathbf{I}_{u_j} \ ,$$

where $\mathbf{A}_j \in \mathbb{R}^{r_j, c_j}$, $l_j = \prod_{i=1}^{j-1} c_i$ and $u_j = \prod_{i=j+1}^{J} r_i$.