Efficient model checking of the stochastic logic CSL^{TA}

E. G. Amparore[a], S. Donatelli[a]

[a]*Dipartimento di Informatica, Università di Torino, Italy*

Abstract

CSL^{TA} is a stochastic temporal logic for continuous-time Markov chains (CTMCs), which includes the more known CSL. CSL^{TA} properties are defined through single-clock Deterministic Timed Automata (DTAs). The model checking of CSL^{TA} amounts, in the worst-case, to the computation of the steady-state probability of a non-ergodic Markov Regenerative Process (MRgP) in the size of $|\mathrm{CTMC}|\times|\mathrm{DTA}|$. Various MRgP solution techniques are available in the literature, and we shall use the Component Method, that computes the steady state distribution of a non-ergodic MRgP by recognizing that, in a MRgP, certain components may actually be solved at a lower cost (same cost as that of a CTMC solution). Unfortunately the technique still requires the construction of the whole MRgP. This paper applies the Component Method to devise various CSL^{TA} model checking algorithms, forward and backward. The Component Method can be applied to the MRgP constructed from the CTMC and the DTA, which is a rather straightforward application of the method, or to the MRgP constructed from the CTMC and the region graph of the DTA, a construction that accounts for timed reachability in the DTA and that allows, in most cases, a significant reduction in the considered MRgP states. In both cases the whole MRgP is built. The primary result of this paper is instead to devise a model-checking algorithm in which the component identification is based only on the region graph of the DTA. The MRgP components are generated "on-the-fly", when needed, starting from the components of the region graph; they are then solved with the cheapest available solution method. Once a component has been solved it is discarded, therefore the whole MRgP is never constructed nor solved. The on-the-fly algorithm is "adaptive": the time and space depend on the formula, and, when the DTA actually expresses a CSL property, the algorithm reduces, seaming-less, to that of standard CSL model checking algorithms.

Keywords: Stochastic Model Checking, stochastic model checking tools, CSL^{TA}, stochastic logic, Markov Regenerative Process (MRgP), MRgP component method, path properties, timed automata, Flexible Manufacturing Systems, CLUE protocol.

1. Introduction

Model-checking of Markov chains is an answer to two different needs in performance evaluation: to gain confidence that we are modelling the right behaviour, and to be able to express and compute performance indices for a subset of model behaviours. In this context "behaviours" are model executions expressed in terms of state sequences and/or event properties. If we consider a simple model of a manufacturing system with two machines, an example of the first type is to check whether it is true that a manufactured piece may experience a breakdown in both machines and still be delivered as a completed piece, while an example of the second type is to compute the probability that a piece will face a breakdown in both machines and still be completed before time T. Temporal logics like CTL [1] and LTL [2] provide a language to express properties about model executions, while stochastic logics like CSL [3] allows to express assertions about the probability of timed executions when the model is a Continuous Time Markov Chain (CTMC). In a performance evaluation context a stochastic logic like CSL can be used to provide also qualitative answers, when the check of the presence of certain model executions is reduced to the assessment of a probability greater than zero for those executions.

Email addresses: amparore@di.unito.it (E. G. Amparore), susi@di.unito.it (S. Donatelli)

In CSL model executions model execution requirements – reviewer1 (typically called "paths") are specified by two operators: timed neXt and timed Until. From a computational point of view the model checking of a CSL property for a CTMC $\mathcal{M}$ requires to solve one or more CTMCs $\mathcal{M}_i$ derived from $\mathcal{M}$ by making certain states absorbing. The solution is the computation of the probability distribution of the states at a specific time instant t, or in steady state, depending on the formula. CSL has been extended in several ways that include action names (name of the events in paths) and path properties specified using regular expressions leading to asCSL [4], or rewards, leading to CSRL [5]. Note that asCSL can specify rather complex path behaviour, but this complexity cannot involve the timed behaviour. GCSRL [6] is an extension of CSRL to model check CTMC generated from Generalized Stochastic Petri nets (GSPN) [7] taking into account both stochastic and immediate events. CSL model checking is included in the tool Prism [8] that has a widespread acceptance in the research community, and that has found interest and application in several industrial contexts. CSL can be verified using MRMC [9], a successor of the tool E$\vdash$MC^2 [10]. Reward measures associated to paths can be computed in the recently released tool Storm [11], in the tool Marcie [12] as well as in Prism. In all cases the model checking of a formula is reduced to the solution (in transient or steady-state) of a set of CTMCs.

The logic CSL$^{\text{TA}}$ [13] is an extension of CSL to include more complex requirements on timed paths, that are specified through a single clock Deterministic Timed Automaton (DTA). The use of a DTA in CSL$^{\text{TA}}$ allows to specify paths in terms of state propositions and action names associated with the state changes of the CTMC, but, in contrast to the various forms of CSL listed above, also the timed behaviour of *portions of the paths* can be specified. For example a CSL$^{\text{TA}}$ path property could specify that an action a should happen before time t_1, followed by an action b happening before time t_2 (either absolute or relative to the time of event a). In CSL we can only specify that action a should happen before time t_1 and in asCSL we can specify that action a should happen before action b and that b should happen before time t. CSL$^{\text{TA}}$ model checking is included in the MC4CSL$^{\text{TA}}$ tool [14] and was part of the CoDeMoC tool [15], which is currently not available. Statistical model checking (model checking through simulation) of CSL$^{\text{TA}}$ is provided by the tool Cosmos [16].

It was shown in [13] that model checking of CSL$^{\text{TA}}$ can be reduced to the computation of the absorption probability of a Markov Regenerative Process (MRgP) with absorbing states, in particular to the computation of the absorption probability of the accepting state $\top$. If nesting of the CSL$^{\text{TA}}$ operators is allowed the cost is that of the solution of one MRgP per DTA included in the formula. Since a MRgP solution technique is in general much more expensive than a CTMC one, it is clear that the increased power of CSL$^{\text{TA}}$ over CSL comes at a price. The objective of this paper is to devise a set of CSL$^{\text{TA}}$ model checking algorithms that are efficient and *adaptive to the formula*: formulas that are also expressible in CSL should only require CTMC solutions, while formulas that have no equivalents in CSL should be treated in an efficient way by adapting the cost of the solution to the "complexity" of the formula. The proposed solution is based on the Component Method for non-ergodic MRgP solution given in [17].

1.1. Paper contribution

Figure 1 summarizes the different algorithms for model checking a CSL$^{\text{TA}}$ formula specified by a DTA $\mathcal{A}$, for a Markov chain $\mathcal{M}$. The 12 algorithms are listed on the right of Figure 1. They are split in two main categorizes: the algorithms that use a *forward* approach, to determine if a given state satisfies a formula, and the ones that use a *backward* approach, to compute the states that, when considered as initial CTMC state, satisfy the formula. Another distinction is based on the applied solution approach: whether the whole MRP is solved as a single monolithic process (*Full* approach) or using a component-based solution (*Comp* approach). We also distinguish whether the MRgP is generated by using the $\mathcal{M}\times\mathcal{A}$ construction defined in the original CSL$^{\text{TA}}$ paper [13] or the $\mathcal{M}\times\mathcal{Z}$ one, where $\mathcal{Z}$, for the time being, can be thought of as the region graph of the timed automaton $\mathcal{A}$. This difference is indicated by the presence of a superscript $\mathcal{A}$ or $\mathcal{Z}$. The 4 algorithms on the two bottom rows of the table are based on the "on-the-fly" approach discussed above. All of them are built starting from $\mathcal{Z}$, so the superscript is omitted. All algorithms have been implemented in the MC4CSL$^{\text{TA}}$ [14] tool and will be experimentally compared in Section 7.

We can summarize the paper contribution by following Figure 1. The top flow represents the model checking as defined in the original paper [13]: the synchronized process $\mathcal{M}\times\mathcal{A}$ combines exponential transitions from the CTMC $\mathcal{M}$ and fixed duration transitions from the DTA $\mathcal{A}$, resulting in a MRgP $\mathcal{R}$, which is then solved with a forward or backward approach, leading to $\text{Full}^{\mathcal{A}}_{\text{fwd}}$ and $\text{Full}^{\mathcal{A}}_{\text{bwd}}$. In both cases the major limitation lies in in the steady-state solution of $\mathcal{R}$ to compute the probability of the accepting $\top$ state. Since $\mathcal{R}$ has absorbing states, we can apply the Component Method defined in [17]: $\mathcal{R}$ is decomposed into multiple, smaller components $\{\mathcal{R}_i\}$ coupled with a precedence relation

that identifies a directed acyclic graph (DAG) of components. This process enjoys the advantage of solving smaller components, which is easier than solving a single large instance since solution complexity is non-linear, and, more importantly, it has been observed that for certain component types the solution reduces to a transient solution of a CTMC. This approach still requires us to build the full MRgP $\mathcal{R}$, to then solve it one component at a time. The first two flows in Figure 1 mainly describe existing work (although the application of the component method with in - reviewer1 a backward approach is new), and will be briefly summarized in this paper for notational consistency.

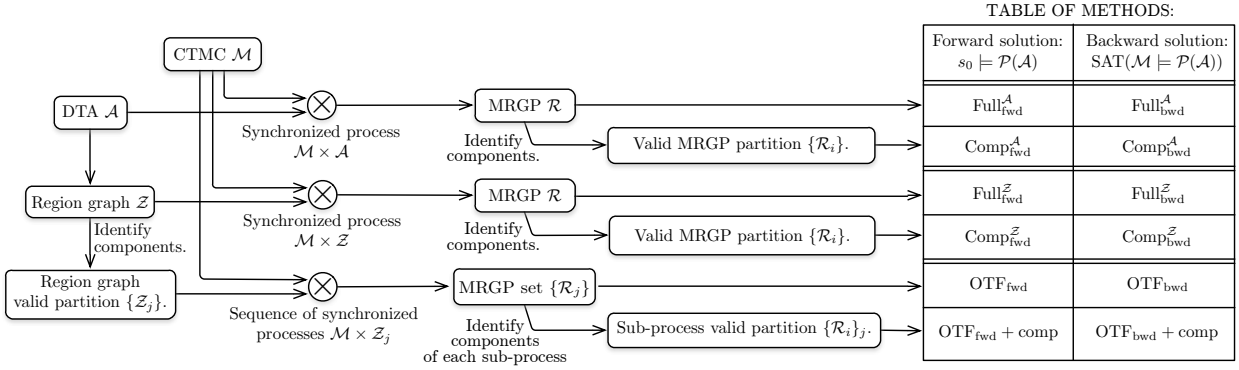


Figure 1: Solution workflow.

In [18] and in [19], it was observed that the same MRgP $\mathcal{R}$ is produced by computing first the region graph $\mathcal{Z} = \mathcal{G}(\mathcal{A})$ of $\mathcal{A}$, which makes explicit the timed reachability of the automaton locations, and by then taking the synchronized product $\mathcal{M} \times \mathcal{Z}$. This solution workflow corresponds to the third flow, leading to the two algorithms $\mathrm{Full}^{\mathcal{Z}}_{\mathrm{fwd}}$ and $\mathrm{Full}^{\mathcal{Z}}_{\mathrm{bwd}}$. Again, the Component Method can be applied, as described by the fourth flow, leading the two algorithms $\mathrm{Comp}^{\mathcal{Z}}_{\mathrm{fwd}}$ and $\mathrm{Comp}^{\mathcal{Z}}_{\mathrm{bwd}}$. Here the formalization of an adequate $\mathcal{Z}$ and $\mathcal{M} \times \mathcal{Z}$ and the application of the Component Method to $\mathcal{M} \times \mathcal{Z}$ represents a new contribution, as it is new the state space reduction that we have devised to better exploit the information available in $\mathcal{Z}$ (not represented in the figure for simplicity).

Finally the last two flows represents the main contribution of the paper, that stems from the observation that the component structure of $\mathcal{R}$, and the order in which the components are considered by the Component Method, is mainly determined by the structure of the region graph $\mathcal{Z}$. The methods in the last two flows of Figure 1 therefore compute first the set $\{\mathcal{Z}_j\}_{j=1}^{J}$ of the J components of $\mathcal{Z}$, and an associated DAG structure. Each component $\mathcal{Z}_j$ is then used for the synchronized product with the CTMC $\mathcal{M}$, to produce a MRgP component $\mathcal{R}_j$: the single components are generated and solved only when required by the precedence relation of the DAG and they can be immediately deleted afterward. Again the technique is applied using both forward and backward approaches, leading to algorithms$\mathrm{OTF}_{\mathrm{fwd}}$ and $\mathrm{OTF}_{\mathrm{bwd}}$, respectively. These two techniques are what we term altogether "on-the-fly" model checking of $\mathrm{CSL}^{\mathrm{TA}}$. Finally the last flow stems from the observation that, since each component $\mathcal{R}_j$ is actually a non-ergodic MRgP, it is possible to re-apply the Component Method on each component, to further increase the efficiency. These variations are named $\mathrm{OTF}_{\mathrm{fwd}}$+comp and $\mathrm{OTF}_{\mathrm{bwd}}$+comp.

1.2. Paper outline

The paper develops as follows: Section 2 defines the necessary background on the MRgP Component Method and on the definition of $\mathrm{CSL}^{\mathrm{TA}}$ and of its model checking procedure. Section 3 discusses the application of the MRgP Component Method to $\mathrm{CSL}^{\mathrm{TA}}$ model checking (Comp methods of Figure 1). Section 4 formalizes the use of a region graph to optimize the model checking process (methods with $\mathcal{Z}$ superscript in Figure 1), to pave the way to the presentation of the on-the-fly model checking of $\mathrm{CSL}^{\mathrm{TA}}$ formulas (OTF methods of Figure 1), presented in Section 5. Section 6 compares, in a theoretical framework, the OTF approach with the other component approaches, while Section 7 is devoted to the assessment of the 12 algorithms on a set of numerical experiments, for different Markov chains and different types of properties. A short introduction to the $\mathrm{MC4CSL}^{\mathrm{TA}}$ tool used for the experiments and a comparison with Prism and Storm on CSL formulas is also provided. Section 8 reviews the literature and Section 9 concludes the paper and outlines future possible extensions.

2. Preliminaries

The material of this section is taken from the literature. It is reported here to make the paper self-contained, to provide a unified language and notation for the reader, and to introduce a running example. Readers familiar with the topics can simply go through the definitions and/or the examples.

2.1. MRgP and the Component Method

The definition of a MRgP in terms of its continuous time stochastic process $X(t)$ can be found in [20]: for this paper we only recall the MRgP representation. MRgPs arise in many contexts in performance evaluation, for example it is the stochastic process underlying Deterministic Stochastic Petri Nets (DSPNs) [21] and Non-Markovian Stochastic Petri Nets [22] when at most one general transition is enabled in any state.

In our context, a MRgP is represented as a discrete event system (like in [23]) with a finite state space, where in each state a general event g is taken from a set G. As time flows, the age of g being enabled is kept, until either g fires ($\mathbf{\Delta}$ event), or a Markovian transition, concurrent with g, fires. Markovian events may actually disable g ($\bar{\mathbf{Q}}$, preemptive event), clearing its age, or keep g running with its accumulated age ($\mathbf{Q}$, non-preemptive event).

Definition 1 (MRgP). *A representation of a* Markov Regenerative Process *(MRgP) stochastic process is a tuple* $\mathcal{R} = \langle \mathcal{S}, G, \Gamma, \mathbf{Q}, \bar{\mathbf{Q}}, \mathbf{\Delta} \rangle$ *where* $\mathcal{S}$ *is a finite set of states,* $G = \{g_1 \dots g_m\}$ *is a set of general events,* $\Gamma : \mathcal{S} \to G \cup E$ *is a function that assigns to each state the single general event enabled in that state, if any, or* E *if only Markovian events can take place.* $\mathbf{Q} : \mathcal{S} \times \mathcal{S} \to \mathbb{R}_{\geq 0}$ *is the non-preemptive transition rates function (rate of non-preemptive Markovian events),* $\bar{\mathbf{Q}} : \mathcal{S} \times \mathcal{S} \to \mathbb{R}_{\geq 0}$ *is the preemptive transition rates function (rate of preemptive Markovian events),* $\mathbf{\Delta} : \mathcal{S} \times \mathcal{S} \to \mathbb{R}_{[0..1]}$ *is the branching probability distribution (probability of reaching a state after the firing of a general event).*

The firing of a *non-preemptive* Markovian event does not affect the enabling of general transitions. The firing of a *preemptive* event in state s resets the age memory of the general event $\Gamma(s)$ enabled in s. A state s is *absorbing* iff $\forall\, s' \in \mathcal{S}$ it holds that $s \neq s' \Rightarrow \mathbf{Q}(s, s') = 0 \wedge \bar{\mathbf{Q}}(s, s') = 0 \wedge \mathbf{\Delta}(s, s') = 0$.

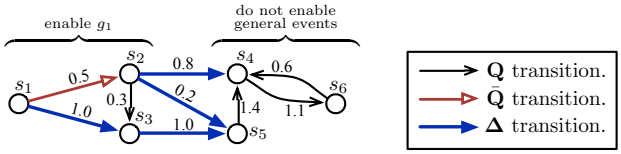


Figure 2: A sample MRgP with 6 states.

Example 1 (MRgP). *Figure 2 depicts an example of a MRgP with 6 states (*$\mathcal{S} = \{s_1, s_2, s_3, s_4, s_5, s_6\}$*), a single general event* g_1 *enabled in* s_1*,* s_2*, and* s_3 *(*$\Gamma(s_1) = \Gamma(s_2) = \Gamma(s_3) = g_1$*), and matrices* $\mathbf{Q}$*,* $\bar{\mathbf{Q}}$ *and* $\mathbf{\Delta}$ *depicted with different graphic styles for the edges. Transition rates and branching probabilities are written close to each arc.*

The steady-state solution of a MRgP can be computed using either standard techniques, that require the construction of the embedded DTMC $\mathbf{P}$, that accounts for the transition probabilities among regeneration points, or using the matrix free technique proposed in [24]. The latter is significantly more efficient in space, and usually also in time. The work in [17] shows that a non-ergodic MRgP can be solved using the *Component Method*, which is the basis for our on-the-fly technique and that we therefore recall in the following. If the MRgP is non-ergodic, it is possible to identify a partition $\{\mathcal{S}_i\}_{1 \leq i \leq n}$ of the MRgP states that induces a directed acyclic graph (DAG) among the $\mathcal{S}_j$ components. In this case we assume, without loss of generality, that the embedded DTMC $\mathbf{P}$ has k transient components $\mathbf{T}_i$, $n - k$ recurrent components $\mathbf{R}_i$. For this paper we assume that the $\mathbf{R}_n$ component is just a single absorbing state $\top$. All other entries in $\mathbf{P}$ are zero. By rearranging the numbering of the components to account for the DAG structure, matrix $\mathbf{P}$ can then be expressed in *reducible normal form* (RNF):

$$\mathbf{P} = \begin{bmatrix} \mathbf{T}_1 & \mathbf{F}_1 & & \\ & \ddots & \vdots & \\ & \mathbf{T}_k & \mathbf{F}_k & \\ & & \mathbf{R}_{k+1} & \\ & & & \ddots \\ & & & \mathbf{R}_{n-1} \\ & & & & \top \end{bmatrix} \tag{1}$$

The component method allows to compute the steady-state probability of all recurrent components, but in this paper we are only interested in the probability of the $\top$ state, $\boldsymbol{\pi}_\top$. Let $\boldsymbol{\mu}_i$ be the vector of *outgoing probabilities* [25] from the states of $\mathcal{S}_i$, which means that for each $s' \in (\mathcal{S} \setminus \mathcal{S}_i)$ the value $\boldsymbol{\mu}_i(s')$ is the one-jump probability of reaching s' after leaving the states of $\mathcal{S}_i$. Since we are only concerned with the probability of reaching the $\top$ state, and given that the $(n-k)$ recurrent subclasses have a zero probability of reaching $\top$, the general formula in [17] can be simplified as follows.

Definition 2 (Outgoing probability for $\mathcal{S}_i$ and probability of $\top$ state). *Given a non-ergodic MRgP $\mathcal{R}$ of embedded DTMC $\mathbf{P}$, as defined above, we define the vectors $\boldsymbol{\mu}_i$ and $\boldsymbol{\pi}_\top$ as:*

$$\begin{aligned} \boldsymbol{\mu}_i &= \left(\boldsymbol{\alpha}_i + \sum_{h<i} (\mathbf{I}_i \cdot \boldsymbol{\mu}_h)\right) \cdot (\mathbf{I} - \mathbf{T}_i)^{-1} \cdot \mathbf{F}_i, \quad i \leq k \\ \boldsymbol{\pi}_\top &= \boldsymbol{\alpha}_\top + \sum_{h=1}^{k} (\mathbf{I}_\top \cdot \boldsymbol{\mu}_h) \end{aligned} \tag{2}$$

where $\boldsymbol{\alpha}_i$ is the initial probability vector for the states in $\mathcal{S}_i$, i.e. $\boldsymbol{\alpha}_i = \mathbf{I}_i \cdot \boldsymbol{\alpha}$, and $\mathbf{I}_i$ is the identity matrix where rows corresponding to $\mathcal{S} \setminus \mathcal{S}_i$ states are set to zero.

The probability of reaching the $\top$ state is then given by the sum of all the outgoing probabilities from the transient subclasses, and each of these outgoing probabilities $\boldsymbol{\mu}_i$ may depend on the $\boldsymbol{\mu}_h$ vectors, $h < i$. If we want to compute instead the probability of reaching $\top$ from *any* given state s of a component $\mathcal{S}_i$ (backward approach), we can fix a reward 1 to the $\top$ state and compute the expected reward vector $\boldsymbol{\xi}_i$. The value $\boldsymbol{\xi}_i(s)$ is then the probability of reaching $\top$ when s is considered as initial state.

Definition 3. *Given a non-ergodic MRgP $\mathcal{R}$ of embedded DTMC $\mathbf{P}$, as defined above, the vector of* state rewards $\boldsymbol{\xi}_i$ *is computed as:*

$$\boldsymbol{\xi}_i = (\mathbf{I} - \mathbf{T}_i)^{-1} \cdot \sum_{h=i+1}^{n} (\mathbf{F}_i \cdot \boldsymbol{\xi}_h) \qquad i \leq k \tag{3}$$

with $\boldsymbol{\xi}_n = \boldsymbol{\xi}_\top = 1$ and $\boldsymbol{\xi}_i = \mathbf{0}, k < i < n$.

Given a DAG of components, the Component Methods takes the component in the topological order induced by the DAG and repetitively applies Eq. 2 to compute the probability of reaching the $\top$ state from a given initial state, or Eq. 3 to compute the probability, for each state, of reaching $\top$. The computation of the outgoing probability vector and of the vector of state rewards may require a steady state or a transient solution of a CTMC or a complete MRgP solution, depending on the component characteristics.

2.2. MRgP matrix-free solution

The solution techniques for MRgP used in this paper are of the *matrix-free* type. The standard solution of a MRgP requires us to build and store a DTMC (the stochastic process observed at regeneration points) and to solve it. Even if the MRgP representation is sparse, typically that of the embedded chain is not, moreover each row in the DTMC may correspond to the transient solution of a CTMC (the subordinated process), which makes the standard

MRgP solution impractical for more than a few thousands states. In [24] a matrix-free approach has been devised, in which the DTMC is never explicitly computed and stored. This allows us to solve much larger MRgPs (hundreds of thousands of states). The work in [26] extends the matrix-free approach to the case of non-ergodic MRgP (like the ones generated by CSL$^{\text{TA}}$ model checking). Later work [17] shows that a non-ergodic MRgP can be solved by taking one component at a time, while still preserving the matrix-free approach. Note that in [26] and [17], following the notational choice of the seminal work in [24] for matrix-free solution, Markov Regenerative processes are indicated with the acronym MRP. Since in previous work, see for example [27], MRgP was used and MRP was reserved to refer to Markov Renewal Processes, we prefer in this paper to stick to the original definition, to avoid propagation of a double definition of the same acronym.

2.3. *The logic* CSL$^{\text{TA}}$.

CSL$^{\text{TA}}$ defines properties to be verified on ASMC (continuous-time Markov chain with actions and state labels) Properties are expressed through Deterministic Timed Automata (DTA). The cross-product $\mathcal{M}\times\mathcal{A}$ of an ASMC $\mathcal{M}$ with a DTA $\mathcal{A}$ is the MRgP whose solution is at the basis of CSL$^{\text{TA}}$ model checking. These three elements are recalled in the following, through their definition, and illustrated by the running example of Figure 3.

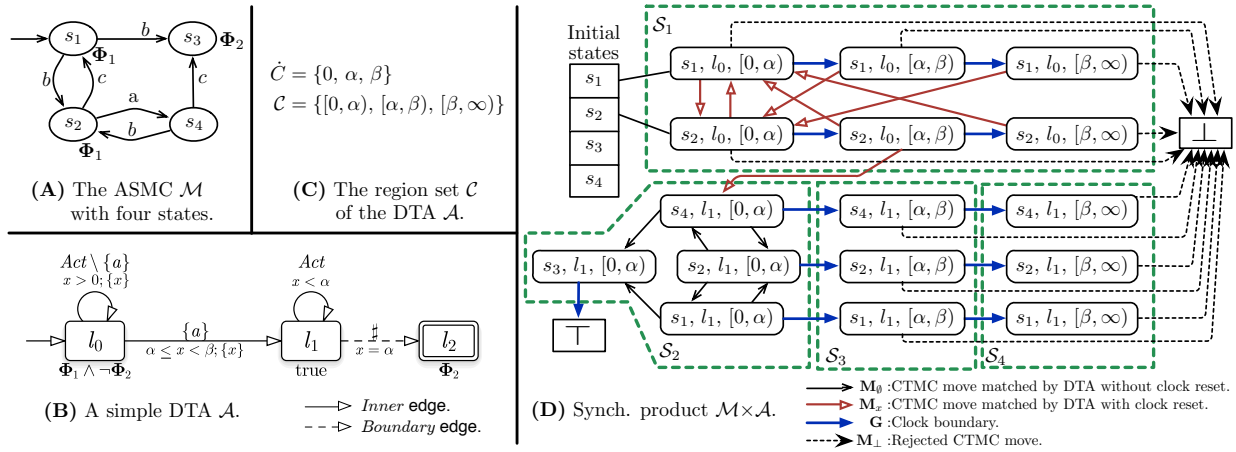


Figure 3: An example of: (A) an ASMC $\mathcal{M}$, (B) a DTA $\mathcal{A}$ and (C) its region set, (D) the MRgP of the synchronized process $\mathcal{M}\times\mathcal{A}$. ASMC and MRgP rates are omitted.

Definition 4 (ASMC representation). *A* continuous time Markov chain *with state and action labels is represented by a tuple* $\mathcal{M} = \langle S, Act, AP, lab, R\rangle$*, where* S *is a finite set of* states*,* Act *is a finite set of* action names*,* AP *is a finite set of* atomic propositions*,* $lab : S \to 2^{AP}$ *is a* state-labeling *function that assigns to each state a set of atomic propositions,* $R \subseteq S \times Act \times S \to \mathbb{R}_{\geq 0}$ *is a* rate function*. If* $\lambda = R(s,a,s') \wedge \lambda > 0$*, we write* $s \xrightarrow{a,\lambda} s'$*.*

Example 2 (ASMC example). *Figure 3(A) is the representation of an ASMC* $\mathcal{M}$ *. It comprises* 4 *states* s_1, s_2, s_3*, and* s_4*;* $\mathbf{\Phi}_1$ *and* $\mathbf{\Phi}_2$ *are atomic propositions associated to states.* $Act = \{a, b, c\}$ *and* $AP = \{\mathbf{\Phi}_1, \mathbf{\Phi}_2\}$*. Each transition is labeled with its action. For instance, the transition from* s_2 *to* s_4 *triggers an* a *action.*

Definition 4 is based on rate matrices, to allow self-loops in state s for action a, when $R(s,a,s) > 0$. A state s is then called *absorbing* if $R(s,a,s') = 0$ for all possible (a,s'), with $s' \neq s$. An *infinite path* in a ASMC $\mathcal{M}$ is a sequence: $\sigma = s_0 \xrightarrow{a_0,t_0} s_1 \xrightarrow{a_1,t_1} s_2 \xrightarrow{a_2,t_2} \dots$ with $s_k \in S$, $a_k \in Act$, $t_k \in \mathbb{R}_{>0}$ and $R(s_k,a_k,s_{k+1}) > 0$, for all $k \in \mathbb{N}$. A *finite path* of length l is a sequence: $\sigma = s_0 \xrightarrow{a_0,t_0} s_1 \xrightarrow{a_1,t_1} s_2 \xrightarrow{a_2,t_2} \dots \xrightarrow{a_{l-1},t_{l-1}} s_l$ such that s_l is absorbing and $R(s_i,a_i,s_{i+1}) > 0$ for all $i < l$. From now on whenever we write CTMC we refer to ASMC.

CSL$^{\text{TA}}$ properties are defined through DTAs and, as in [13], we consider DTA with a single clock x.

Definition 5 (DTA). *A* single-clock Deterministic Timed Automaton *is defined by a tuple* $\mathcal{A} = \langle L, \Lambda_{\mathcal{A}}, L_0, L_F, AP,$ *Inner*, *Boundary*$\rangle$ *where* L *is a finite set of* locations*,* $\Lambda_{\mathcal{A}} : L \to \mathcal{B}_{AP}$ *is a function that assigns to each location a*

boolean expression over the set of atomic propositions AP, $L_0 \subseteq L$ is the set of initial locations*, $L_F \subseteq L$ is the set of* final locations*, Inner $\subseteq L \times \mathsf{InC} \times 2^{Act} \times \{\emptyset, \{x\}\} \times L$ is the set of* inner edges*, and Boundary $\subseteq L \times \mathsf{BoundC} \times \{\sharp\} \times \{\emptyset, \{x\}\} \times L$ is the set of* boundary edges*, where the inner constraints* InC *take the form $\alpha \leq x < \beta$ and the boundary constraints* BoundC*, take the form $x = \alpha$.*

We shall use the notations $l \xrightarrow{\hat{c}, A, r} l'$ to denote the inner edge $(l, \hat{c}, A, r, l')$, and $l \overset{\delta_k, \sharp, r}{\dashrightarrow} l'$ to denote the boundary edge $(l, \delta_k, \sharp, r, l')$, respectively. With respect to standard timed automata, there are no location invariants, while the requirement of determinism pertains to the use of the timed automaton as a recognizers of CTMC paths: there should be a unique way to accept a CTMC path, if not non-determinism will come into play in the underlying stochastic process. A precise definition of DTA determinism can be found in [13, Def. 2.3]. In this paper we assume that a DTA deterministically recognizes a CTMC path. Inner edges, which are labeled by a clock interval and a set of actions, are triggered by CTMC actions. Boundary edges, which are labeled by a clock constant and the special symbol $\sharp$, are triggered as soon as the clock reaches the boundary. Both edges have a *reset set* which is either the empty set (no clock reset), or $\{x\}$ (clock is reset to 0 when the edge is taken).

The DTA reads the transitions of the CTMC, therefore edges are labeled with a condition over the actions of the CTMC (for example $\{a\}$ indicates action a and $Act \setminus \{a, b\}$ means any action but a or b). A DTA can enter and stay in a location l only if $\Lambda(l)$, evaluated over the atomic proposition of the current CTMC state, is satisfied. Moreover *Boundary* edges are urgent and have priority over *Inner* edges. A DTA accepts all CTMC timed paths that take the DTA to a final location. A formal definition of acceptance can be found in [13, sect. 2.3].

Example 3 (DTA example). *Figure 3(B) shows a DTA with three locations: l_0, l_1, and l_2. There is a single initial location, l_0, and a single final location l_2. The DTA is equipped with its clock x; an edge of the DTA can be taken only when the clock expression associated to the edge is true. The clock is reset if $\{x\}$ is indicated on the arc. In the graphical representation of the DTA the edge has a two-levels inscription. Taking as an example the self-loop edge over l_0, the upper inscription is the condition over the CTMC actions (any action but a), while the lower inscription is a condition over the clock $(x > 0)$ and a clock reset $(\{x\})$. The arc from l_0 to l_1 is an Inner edge with an associated clock reset, and it may be taken only when the value of x is in between α and β, and the CTMC performs a transition labeled with action a. The arc from l_1 to l_2 is a boundary edge with no associated clock reset, and can be taken only when the clock is equal to α. By definition there is no CTMC action associated to boundary edges. Locations have an associated boolean expression: in the example $(\mathbf{\Phi}_1 \wedge \neg \mathbf{\Phi}_2)$, $\mathbf{true}$ and $\mathbf{\Phi}_2$ are associated to l_0, l_1, and l_2, respectively.*

This DTA accepts the CTMC timed paths with the following structure: any prefix not including action a (the prefix is accepted by the self loop on the initial location l_0) which passes exclusively over $(\mathbf{\Phi}_1 \wedge \neg \mathbf{\Phi}_2)$-states, followed by an a-labeled transition that happens between α and β time units since the last clock reset, followed by a path that finds the CTMC in a $\mathbf{\Phi}_2$-state exactly α time units after the last clock reset.

Definition 6 (CSL$^{\text{TA}}$). *A* CSL$^{\text{TA}}$ *property over a set AP of atomic propositions is defined as*

$$\Phi ::= p \mid \neg\Phi \mid \Phi \wedge \Phi \mid \mathcal{P}_{\bowtie\lambda}(\mathcal{A}) \mid \mathcal{S}_{\bowtie\lambda}(\Phi)$$

where $p \in AP$, $\bowtie \in \{\leq, <, >, \geq\}$, and $\mathcal{A}$ is a DTA.

A state s of a CTMC $\mathcal{M}$ satisfies $\mathcal{P}_{\bowtie\lambda}(\mathcal{A})$ (written $(\mathcal{M}, s) \models P_{\bowtie\alpha}(\mathcal{A})$) if the probability of the set of paths stemming from s, accepted by the DTA $\mathcal{A}$, is $\bowtie \lambda$. The $\mathcal{S}_{\bowtie\lambda}$ is a steady-state property: since its satisfaction only requires the steady-state solution of CTMCs, we do not consider this operator in the rest of the paper. A formal definition of acceptance can be found in [13, Def. 2.9]. The computation of $\mathcal{P}_{\bowtie\lambda}(\mathcal{A})$ is reduced in [13] to the computation of $\boldsymbol{\pi}(\top)$, the probability of reaching the accepting state $\top$ in the cross-product of $\mathcal{M}$ with $\mathcal{A}$, called $\mathcal{M} \times \mathcal{A}$, that identifies all and only the timed paths of $\mathcal{M}$ that are accepted by $\mathcal{A}$.

2.4. CSL$^{\text{TA}}$ *model checking*

The construction of the synchronized process $\mathcal{M} \times \mathcal{A}$ is heavily dependent on the *Inner* and *Boundary* constraints on the DTA clock x The constraints on the clock induce a partitioning of the time interval $[0, \infty)$.

Definition 7 (Region set). *Given a finite set of positive real constants $\dot{C} = \{\delta_0{=}0, \delta_1, \ldots, \delta_m\}$, with $\delta_k < \delta_{k+1}$ for all $0 \leq k < m$, the* region set $\mathcal{C}$ *induced by $\dot{C}$ is defined as $\mathcal{C} = \{[\delta_k, \delta_{k+1}) \mid 0 \leq k < m\} \cup \{[\delta_m, \infty)\}$. Each interval $c = [\delta_k, \delta_{k+1}) \in \mathcal{C}$ is a* region.

When the constants in $\dot{C}$ are those appearing on the DTA $\mathcal{A}$, including 0 if not already present, then Def. 7 is a simplified version of the standard region definition (see for example [28]) for single-clock automata (as used in [29] and [19]).

Example 4 (Region set example). *The clock of the DTA in Figure 3(B) is compared against three possible values: $0, \alpha, \beta$ with $\alpha \leq \beta$, giving rise to the region set of Figure 3(C).*

Any *clock interval* of type $[\delta_j, \delta_h)$, with $j \leq h$ can be considered as $\hat{c}$, the union of the regions $[\delta_k, \delta_{k+1}), j \leq k < h$. Since an inner constraint c has the form $\delta_j \leq x < \delta_h$, with $j < h$, we can write inner constraints as clock intervals (union of regions), therefore the inner edge $(l, \delta_j \leq x < \delta_h, A, r, l')$ is re-written as $(l, \hat{c}, A, r, l')$, where $\hat{c}$ is the union of the clock regions of the time interval $[\delta_j, \delta_h)$. Given a region $[\delta_k, \delta_{k+1})$, we denote with $[\delta_k, \delta_{k+1})[r := 0]$ the *region after the reset* r, that is $[\delta_k, \delta_{k+1})$ if $r = \varnothing$, or $[0, \delta_1)$ if $r = \{x\}$.

The synchronized product $\mathcal{M}{\times}\mathcal{A}$ is a MRgP whose states are triplets $\langle s, l, c\rangle$ of a CTMC state s, a DTA location l, and a clock region c, plus two special absorbing states $\top$ and $\bot$. Let $\mathcal{S}$ be the set of MRgP states. The product is built by two rules. Type **(G)** accounts for the situation in which the Markov chain does not move and the time elapses: by letting the time elapse the system can reach a system boundary, where a boundary edge of the DTA may be taken: this is accounted by the definition of the *closure* function. Type **(M)** accounts for the effect of a CTMC transition that may not be accepted by the DTA (rule $\mathbf{M}_\bot$), or that is accepted by an edge with a reset (rule $\mathbf{M}_x$) or without (rule $\mathbf{M}_\varnothing$). The definition is slightly complicated by the fact that even rules that do not necessarily include a clock reset, like $\mathbf{M}_\varnothing$ or $\mathbf{M}_\bot$ may actually preempt a clock if the reached state is $\top$ or $\bot$ and if there is a clock active. The construction makes use of two functions: $fin : \mathcal{S} \to \mathcal{S} \cup \{\top\}$ to check whether we have reached a final state in the DTA, and therefore the MRgP has reached the $\top$ state; And $closure : \mathcal{S} \to \mathcal{S} \cup \{\top\}$ to perform a transitive closure over boundary edges, that are taken, when possible, as soon as a region is entered (boundary edges are urgent in DTA). Remember that we assume that the DTA deterministically recognizes the CTMC paths (as in [13, Def. 2.3]), which also implies that *initial locations....* . In this paper we assume that a DTA deterministically recognizes a CTMC path.

Definition 8 (Synchronized product $\mathcal{M}{\times}\mathcal{A}$). *The synchronized product of a CTMC $\mathcal{M}$ with a DTA $\mathcal{A}$ of region set $\mathcal{C}$, built on the constants $\dot{C} = \{\delta_0 = 0, \delta_1, \ldots, \delta_m\}$ of $\mathcal{A}$, is an MRgP $\mathcal{R} = \langle \mathcal{S}, G, \Gamma, \mathbf{Q}, \bar{\mathbf{Q}}, \mathbf{\Delta}\rangle$ where*

- *$\mathcal{S} \subseteq (S \times L \times \mathcal{C}) \cup \{\top, \bot\}$. We indicate with $\langle s, l, c\rangle$ a generic state in $S \times L \times \mathcal{C}$.*
- *The set G has one event $g_k, k \geq 1$ of deterministic duration $(\delta_k - \delta_{k-1})$ for each finite clock region $[\delta_{k-1}, \delta_k) \in \mathcal{C}$.*
- *Let $\Gamma(s, l, c)$ be defined as g_k if $c = [\delta_{k-1}, \delta_k), 1 \leq k \leq m$, and E otherwise.*
- *Let $fin(s, l, c)$ be $\top$ if $l \in L_F$, and $\langle s, l, c\rangle$ otherwise.*
- *Let $closure(s, l, c)$, with $c = [\delta_k, \delta_{k+1})$, be defined recursively to $closure(s, l', c[r := 0])$ if there is in $\mathcal{A}$ a Boundary edge $l \xrightarrow{\delta_k, \sharp, r} l'$ with $s \models \Lambda(l')$; otherwise $closure(s, l, c) = fin(s, l, c)$.*
- *The tuples $\langle s, l, c\rangle \in \mathcal{S}$ and the matrices $\mathbf{Q}, \bar{\mathbf{Q}}, \mathbf{\Delta}$ are defined through the following rules:*
  - *For each $s \in S$, if there is an initial location $l_0 \in L$ with $s \models \Lambda(l_0)$, then $closure(s, l_0, c_0) \in \mathcal{S}$.*
  - ***(G:** let time elapse). Given $\langle s, l, c\rangle$ with $c = [\delta_{k-1}, \delta_k), k \leq m$, then $closure\big(s, l, [\delta_k, \delta_{k+1}))\big) \in \mathcal{S}$ and $\mathbf{\Delta}(\langle s, l, c\rangle, closure\big(s, l, [\delta_k, \delta_{k+1})\big)) = 1$ (and we assume $\delta_{m+1} = \infty$).*
  - ***(M:** CTMC transition). Given $\langle s, l, c\rangle \in \mathcal{S}$ and the CTMC transition $s \xrightarrow{a, \lambda} s'$, then:*
    - ***($\mathbf{M}_\varnothing$)** if $\exists l \xrightarrow{\hat{c}, A, \varnothing} l'$ with $c \in \hat{c} \wedge a \in A \wedge s' \models \Lambda(l')$, then $fin(s', l', c) \in \mathcal{S}$ and $\mathbf{Q}$ and $\bar{\mathbf{Q}}$ are modified as follows:* $\begin{cases} \mathbf{Q}[\langle s, l, c\rangle, fin(s', l', c)] = \lambda & \text{if } \Gamma(\langle s, l, c\rangle) = E \text{ or } fin(s', l', c) \neq \top \\ \bar{\mathbf{Q}}[\langle s, l, c\rangle, \top] = \lambda & \text{if } \Gamma(\langle s, l, c\rangle) \neq E \text{ and } fin(s', l', c) = \top \end{cases}$
    - ***($\mathbf{M}_x$)** if $\exists l \xrightarrow{\hat{c}, A, \{x\}} l'$ with $c \in \hat{c} \wedge a \in A \wedge s' \models \Lambda(l')$, then $closure(s', l', [0, \delta_1)) \in \mathcal{S}$ and $\mathbf{Q}$ and $\bar{\mathbf{Q}}$ are modified as follows:* $\begin{cases} \mathbf{Q}[\langle s, l, c\rangle, closure(s', l', [0, \delta_1))] = \lambda & \text{if } c = [\delta_m, \infty) \\ \bar{\mathbf{Q}}[\langle s, l, c\rangle, closure(s', l', [0, \delta_1))] = \lambda & \text{if } c \neq [\delta_m, \infty) \end{cases}$

$*$ *($\boldsymbol{M}_\perp$) otherwise (no edge in $\mathcal{A}$ matches the CTMC transition), and $\mathbf{Q}$ and $\bar{\mathbf{Q}}$ are modified as follows:*

$$\begin{cases} \mathbf{Q}[\langle s,l,c\rangle,\perp] = \lambda & \text{if } c = [\delta_m,\infty) \\ \bar{\mathbf{Q}}[\langle s,l,c\rangle,\perp] = \lambda & \text{if } c \neq [\delta_m,\infty) \end{cases}$$

Example 5 (Example of a synchronized product $\mathcal{M}\times\mathcal{A}$). *Figure 3(D) shows the $\mathcal{M}\times\mathcal{A}$ construction for our running example. Transition rates and branching probabilities are omitted, for clarity. The boxes with dotted lines represent the MRgP components used by the component-method. The MRgP is constructed considering all possible initial states, in this case s_1 and s_2 which are the only CTMC states that satisfy the $(\boldsymbol{\Phi}_1 \wedge \neg\boldsymbol{\Phi}_2)$ condition of l_0. The MRgP has two general events: g_1, deterministic of duration α and g_2, deterministic of duration $\beta-\alpha$, $\Gamma(\langle s,l,c\rangle) = g_1$ if $c = [0,\alpha)$, $\Gamma(\langle s,l,c\rangle) = g_2$ if $c = [\alpha,\beta)$, and $\Gamma(\langle s,l,c\rangle) = E$ otherwise. From the MRgP graph it is easy to identify the paths that lead to the $\top$ and $\perp$ states. For example, from $\langle s_1,l_0,[\alpha,\beta)\rangle$ if we let the time elapse the MRgP moves to $\langle s_1,l_0,[\beta,\infty)\rangle$. If the CTMC moves from s_1 to s_3 then the MRgP moves to $\perp$, since s_3 does not satisfy $(\boldsymbol{\Phi}_1 \wedge \neg\boldsymbol{\Phi}_2)$ and the DTA does not accept this transition in location l_0. If the CTMC moves from s_1 to s_2, which is a $\boldsymbol{\Phi}_1$-state, then the DTA accepts the transition through the self-loop over l_0, as a consequence the clock is reset and the MRgP moves to $\langle s_2,l_0,[0,\alpha)\rangle$. Note that the $\top$ state is reached from $\langle s_3,l_1,[0,\alpha)\rangle$ through the boundary DTA edge from l_1 to l_2, when the $x = \alpha$ constraint is satisfied.*

2.5. Forward and backward MRgP solution for CSL$^{\text{TA}}$ *model checking*

Model-checking of a property $\varphi = \mathcal{P}_{\bowtie\lambda}(\mathcal{A})$ for a CTMC $\mathcal{M}$ may come in two flavors: to determine if a state s of the CTMC (typically the initial one) satisfies φ, written as $s \models \varphi$, or to compute the satisfiability set $Sat(\varphi)$, the set of the CTMC states that satisfy the formula ($Sat(\varphi) = \{s \in \mathcal{S} \mid s \models \varphi\}$). This distinction leads to:

- *Forward approach:* $s \models \varphi$: compute the long run probability distribution $\boldsymbol{\pi}$ of all MRgP states, given a fixed initial distribution π_0 with $\pi_0(s) = 1$. Property is satisfied if $\boldsymbol{\pi}(\top) \bowtie p$.
- *Backward approach*: compute the probability vector $\boldsymbol{\xi}$ that each state, considered as initial, will reach the fixed target state $\top$, i.e. $\boldsymbol{\xi}(s) = \lim_{t\to\infty} Pr\{X(t) = \top \mid X(0) = s\}$, with $X(t)$ the MRgP state at time t. $Sat(\varphi)$ is then the set of all states for which $\boldsymbol{\xi}(s) \bowtie p$

3. Component method for CSL$^{\text{TA}}$

This section illustrates the structure of the MRgP $\mathcal{M}\times\mathcal{A}$ and revisits the forward Component Method in a more formal setting than that provided in [17], to make it in a form suitable for the on-the-fly extension of next section. The method is also extended to work backward. We start by analyzing the $\mathcal{M}\times\mathcal{A}$ structure to see if it is suitable for the Component Method.

The $\mathcal{M}\times\mathcal{A}$ structure. The MRgP $\mathcal{M}\times\mathcal{A}$ has one deterministic event g_k per clock region and its state space $\mathcal{S}$ can be partitioned accordingly into $m+3$ sets: m sets $\mathcal{S}_{g_k}$, the set $\mathcal{S}_E$ of states in the last clock region $[\delta_m,\infty)$ in which no general event is enabled, and the two absorbing states $\top$ and $\perp$. The structure of the MRgP matrices is shown in Figure 4, in gray the portions of the $\mathbf{Q}$, $\bar{\mathbf{Q}}$, and $\boldsymbol{\Delta}$ matrices that can be non-zero. For each gray portion it is indicated the identifier of the corresponding $\mathcal{M}\times\mathcal{A}$ rule of Definition 8; For readability $\mathbf{M}_*$ indicates the contribution of both $\mathbf{M}_x$ and $\mathbf{M}_\emptyset$ rules, $\mathcal{S}_{g_k}$ is shortened into g_k and $\mathcal{S}_E$ into E.

According to the $\mathcal{M}\times\mathcal{A}$ construction **(G)** rules contribute only to $\boldsymbol{\Delta}$: a transition caused by a let time elapse event followed by a boundary edge with an associated clock reset takes the MRgP back to the first time region (where g_1 is enabled), and this is represented by the sub-matrices $\mathbf{G}_x$ in the Figure, while all other transitions caused by a "let time elapse" event are indicated as $\mathbf{G}_\emptyset$. Rows for states in $\mathcal{S}_E$ are zero (no general event enabled in $\mathcal{S}_E$ states) and column to $\perp$ is also zero ($\perp$ can be reached only by a rejected CTMC transition).

Accepted CTMC moves without clock reset ($\mathbf{M}_\emptyset$ rule) contribute to the diagonal blocks of $\mathbf{Q}$ ($\mathbf{M}_\emptyset$ events do not change the current region). Accepted CTMC with clock reset ($\mathbf{M}_x$ rule) contribute to the $\mathcal{S}_{g_1}$ column of $\bar{\mathbf{Q}}$, when the MRgP is in a finite region ($\mathcal{S}_{g_k}$ states), or of $\mathbf{Q}$, when the MRgP is in the infinite region $[\delta_m,\infty)$ ($\mathcal{S}_E$ states).

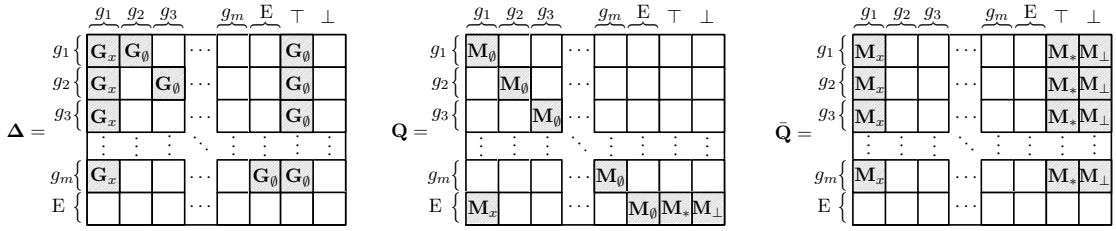


Figure 4: MRgP matrices generated by the $\mathcal{M}\times\mathcal{A}$ synchronized product.

Any accepted CTMC transition, with or without clock reset ($\mathbf{M}_x$ and $\mathbf{M}_\emptyset$ rules) may lead to $\top$ and the contribution goes into either $\mathbf{Q}$ or $\bar{\mathbf{Q}}$, as in the previous case. Similarly, rejected CTMC moves ($\mathbf{M}_\perp$ rule) contribute to the $\perp$ column of either $\mathbf{Q}$ or $\bar{\mathbf{Q}}$.

The MRgP $\mathcal{M}\times\mathcal{A}$ has indeed some peculiarities: since general events represents the time elapsing, they are in causal relationship (the firing of g_k causes g_{k+1} to be enabled, or g_1 if there is a a clock reset); moreover a Markovian event never newly enables a general event, other than g_1, while it may disable it due to a clock reset. To apply the Component Method to $\mathcal{M}\times\mathcal{A}$, the state space $\mathcal{S}$ needs to be partitioned into a DAG of components $\mathcal{S}_i$. It is immediate to observe from Figure 4 that, despite the regular structure of the $\mathbf{Q}, \bar{\mathbf{Q}}$, and $\mathbf{\Delta}$ matrices, the partition of states into $\mathcal{S}_{g_k}$ sets does not lead to a DAG of components, since the sum of the three matrices is not in RNF. Therefore, there is a need for computing a DAG of components of $\mathcal{M}\times\mathcal{A}$, as in the general MRgP case, illustrated in the following.

Solution of $\mathcal{M}\times\mathcal{A}$ with the Component Method. Equation (2) and (3) assumes that the embedded Markov chain $\mathbf{P}$ of the MRgP is available. When the components are generated in a matrix-free setting, hence $\mathbf{P}$ is not available, the Component Method requires some additional care. Indeed if we consider $\boldsymbol{\mu}_i$ and $\boldsymbol{\xi}_i$ of Equation (2) and (3), we can observe that the matrices $\mathbf{T}_i$ and $\mathbf{F}_i$ refer only to the states of the i-th component itself, and that all these states correspond to regeneration points of the MRgP. In the matrix-free approach, the computation of $\boldsymbol{\mu}_i$ and $\boldsymbol{\xi}_i$ is based on $\mathbf{Q}, \bar{\mathbf{Q}}$, and $\mathbf{\Delta}$. To compute the outgoing probability of a component, the process must reach a regeneration point. Therefore the computation should take into account an *augmented set* (as in [17]). The augmented set includes the set itself and all states reachable from the component until a regeneration point is reached. All events leading to an $\mathcal{S}_E$ state correspond to a regeneration point, while events leading to an $\mathcal{S}_{g_k}$ state correspond to a regeneration point only if they disable a previously enabled general event (contribution of the event is in the $\bar{\mathbf{Q}}$ and $\mathbf{\Delta}$ matrices). To precisely define the components used by the algorithm, we introduce the additional notion of *frontier set* (not present in [17]), which is the set of states reached at the next regeneration point.

Definition 9 (Augmented set of a MRgP subset). *Let $\mathcal{S}_i \subseteq \mathcal{S}$ be a set of states of a MRgP $\mathcal{R}$, and suppose $s \xrightarrow{*}_{\mathbf{Q}} s'$ denotes that there exists a path between s and s' made by $\mathbf{Q}$ transitions only. The* augmented set *$\widehat{\mathcal{S}}_i$ of $\mathcal{S}_i$ is defined as the largest set such that:*

$$\widehat{\mathcal{S}}_i = \mathcal{S}_i \cup \left\{ s' \in \mathcal{S} \setminus \mathcal{S}_i \;\middle|\; \exists s \in \mathcal{S}_i \;:\; \Gamma(s) \neq E \;\wedge\; s \xrightarrow{*}_{\mathbf{Q}} s' \right\}$$

Definition 10 (Frontier of a MRgP subset). *Let $\mathcal{S}_i \subseteq \mathcal{S}$ be a set of states of a MRgP $\mathcal{R}$. The* frontier *of $\mathcal{S}_i$ is defined as the largest set such that :*

$$\begin{aligned} \mathit{frontier}(\mathcal{S}_i) = \Big\{ s' \notin \mathcal{S}_i \;\Big|\; \exists\, s \in \widehat{\mathcal{S}}_i : (\Gamma(s) = E \wedge \mathbf{Q}(s,s') \neq 0) \vee \\ \big(\Gamma(s) \neq E \wedge \big(\bar{\mathbf{Q}}(s,s') \neq 0 \vee \mathbf{\Delta}(s,s') \neq 0\big)\big) \Big\} \end{aligned} \tag{4}$$

Example 6 (Augmented set and frontier examples). *Figure 5 shows a component $\mathcal{S}_i = \{s_1, s_2\}$ with its augmented set $\widehat{\mathcal{S}}_i = \{s_1, s_2, s_3\}$ and its frontier set $\mathit{frontier}(\mathcal{S}_i) = \{s_3, s_4, s_5\}$. State s_3 is both in the augmented set and in the frontier of $\mathcal{S}_i$: $s_3 \in \widehat{\mathcal{S}}_i$ (there is a $\mathbf{Q}$ transition from s_1 to s_3) and $s_3 \in \mathit{frontier}(\mathcal{S}_i)$ (there is a $\bar{\mathbf{Q}}$ transition from s_2 to s_3), therefore s_3 can be a non-regenerative state (if entered from s_1) or a regenerative state (if entered from s_2). In the $\mathbf{Q}_i$, $\bar{\mathbf{Q}}_i$ and $\mathbf{\Delta}_i$ matrices state s_3 is duplicated and the non-regenerative s_3 state is indicated with s_3'. Transition rates*

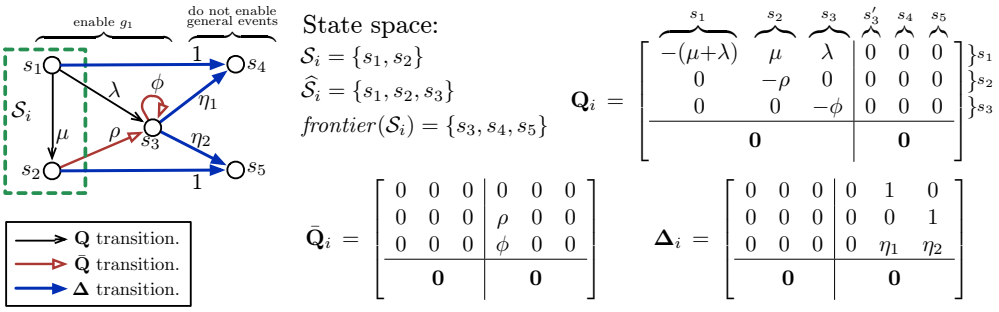


Figure 5: An example of an augmented set and frontier of a component.

are split appropriately: transition from s_1 with rate λ leads to s_3 in $\mathbf{Q}$, since it is non-preemptive, while transition from s_2 with rate ρ leads to s'_3 in $\bar{\mathbf{Q}}$, since it is preemptive. The self-loop over state s_3 reaches state s'_3 in $\bar{\mathbf{Q}}$, since it is preemptive (note that there is no self-loop anymore in the component matrices). The $\mathbf{0}$ term is the zero matrix.

We can then define an MRgP component as:

Definition 11 (MRgP component)**.** *Given a set of states $\mathcal{S}_i$ of an MRgP $\mathcal{R}$, the MRgP component $\mathcal{R}_i = (\mathcal{S}_i, G_i, \Gamma_i, \mathbf{Q}_i, \bar{\mathbf{Q}}_i, \mathbf{\Delta}_i)$ is defined as the projection of $\mathbf{Q}$, $\bar{\mathbf{Q}}$, and $\mathbf{\Delta}$ over the set of states in $\widehat{\mathcal{S}}_i \cup frontier(\mathcal{S}_i)$ and by setting to zero the rows of states in $frontier(\mathcal{S}_i)$ (thus making these states absorbing). G_i is the restriction of G to the general events enabled in $\mathcal{S}_i$ and $\Gamma_i(s) = \Gamma(s)$ if $s \notin frontier(\mathcal{S}_i)$, and E otherwise.*

Algorithm 1 defines the forward component-based model checking procedure that computes the probability of eventually reaching the success state $\top$. It corresponds to the method named $\text{Comp}^{\mathcal{A}}_{\text{fwd}}$ in Figure 1. The MRgP $\mathcal{M}\times\mathcal{A}$ is computed first, as well as a DAG of components $\mathcal{R}_i$. Components are taken one at a time, in *forward topological order*, that is to say: a component is considered in step k only if all components of the states in any path from the initial states to the component itself have already been taken into account in the previous $k-1$ steps. We assume that $\mathcal{R}_i$ components follow the same order, so that the index i of the component coincides with the index k of the step. For each component the probability $\boldsymbol{\mu}_i$ of the frontier states is computed, assuming an initial probability which is the result of the previous steps. At each step the probability $\boldsymbol{\mu}_i$ for the frontier states is added to the current probability vector (vector at step i, $\boldsymbol{\pi}^{(i)}$) and the initial probability of the component is subtracted from the current vector (as it has been "pushed down" to the frontier states by the computation at step i).

Algorithm 1 Pseudocode of the component-based forward model checking method.

function MODELCHECK-COMP$^{\mathcal{A}}_{\text{FWD}}$($s_0$: initial state)
  Construct the synchronized process $\mathcal{R} = \mathcal{M}\times\mathcal{A}$, starting from $\langle s_0, l_0, 0\rangle$.
  Let $\boldsymbol{\pi}^{(0)}$ be the vector with $\boldsymbol{\pi}^{(0)}[\langle s_0, l_0, 0\rangle] = 1$.
  Build a set of components $\{\mathcal{R}_i\}$ of $\mathcal{R}$ that form a DAG.
  for each component $\mathcal{R}_i$ in $\mathcal{R}$ taken in forward topological order **do**
    Let $\mathbf{I}_{\widehat{\mathcal{S}}_i}$ be the filtering matrix of $\widehat{\mathcal{S}}_i$
    Compute (with Equation(2)) the probability μ_i outgoing $\widehat{\mathcal{S}}_i$ and reaching $frontier(\mathcal{S}_i)$
    $\boldsymbol{\pi}^{(i)} \leftarrow (\mathbf{I} - \mathbf{I}_{\widehat{\mathcal{S}}_i}) \cdot \boldsymbol{\pi}^{(i-1)} + \boldsymbol{\mu}_i$
  end for
  return $\boldsymbol{\pi}^{(k)}[\top]$
end function

Example 7 (**Algorithm 1**)**.** *In the MRgP of Figure 3(D), the dotted lines identify 4 components (plus $\top$ and $\bot$), numbered in forward topological order. If s_1 is the initial state considered, at the first step Algorithm 1 considers component $\mathcal{S}_1$ and computes the probability of reaching the frontier states (either $\bot$ or the state in $\mathcal{S}_2$). At the next step $\mathcal{S}_2$ is considered, taking as initial probability that assigned to the frontier states in the previous step. The solution*

at time α of the component assigns a non-null probability to $\top$, as well as to the other frontier states of $\mathcal{S}_2$ (which include all the states of $\mathcal{S}_3$). The algorithm then considers $\mathcal{S}_3$ and $\mathcal{S}_4$ in sequence. These two components do not bring any probability to the $\top$ state, but the procedure defined in Algorithm 1 is not aware of it and therefore two (non-useful) component solutions are performed.

Algorithm 2 defines the backward Component Method $\text{Comp}^{\mathcal{A}}_{\text{bwd}}$ that computes the Sat set for the formula $P_{\bowtie}(\mathcal{A})$. Components are taken in *backward topological order*: a component is considered in step i only if all components on any path from the component itself to the terminal components (components with no outgoing transitions) have already been taken into account. Again we assume that the index of each component indicates the correct topological order, and we therefore use a single index i. $\boldsymbol{\xi}_i$ is the state reward vector for component i as defined by Equation (3), and the reward vector is updated by adding them to the full reward vector $\mathbf{r}$.

Algorithm 2 Pseudocode of the component-based backward model checking method.

function MODELCHECK-COMP$^{\mathcal{A}}_{\text{BWD}}$()
  Construct $\mathcal{R} = \mathcal{M} \times \mathcal{A}$, starting from the set of initial states $\{\langle s, l, 0\rangle \mid s \in S, l \in L_0, s \models \Lambda(l)\}$.
  Build a set of components $\{\mathcal{R}_i\}$ of $\mathcal{R}$ that form a DAG.
  $\mathbf{r}^{(k)} : \mathcal{S} \to \mathbb{R}$ ▷ sparse vector of per-state acceptance probabilities
  $\mathbf{r}^{(k)}[\top] \leftarrow 1$
  for each $\mathcal{R}_i$ in $\mathcal{R}$, taken in backward topological order **do**
    Compute (with Equation (3)), the reward vector $\boldsymbol{\xi}^{(i)}$ for $\widehat{\mathcal{S}}_i$ states, starting from the rewards $\mathbf{r}^{(i-1)}$
    $\mathbf{r}^{(i-1)} \leftarrow \mathbf{r}^{(i)} + \boldsymbol{\xi}^{(i)}$
  end for
  Each $\mathbf{r}^{(0)}[s]$ is the probability of eventually reaching $\top$ from s, for all $s \in \mathcal{S}$.
  return $\mathbf{r}^{(0)}$
end function

Example 8 (Algorithm 2.). *With reference to the MRgP of Figure 3(D), the backward algorithm considers the same components as the forward one, but in the opposite order: $\mathcal{S}_4$, $\mathcal{S}_3$, $\mathcal{S}_2$, $\mathcal{S}_1$. It starts with a reward of 1 for the $\top$ state, and then computes the reward for the component states, based on the reward of the component frontier states. Since for $\mathcal{S}_4$ the reward on the frontier states ($\bot$) is 0, the computation can be skipped. Same for $\mathcal{S}_3$. The method proceeds by solving component $\mathcal{S}_2$ and component $\mathcal{S}_1$, to assign a reward to the two possible initial states s_1 and s_2.*

4. Model checking based on region graph

The $\mathcal{M} \times \mathcal{A}$ construction takes into account, at the same time, time constraints expressed by the DTA and the acceptance of CTMC moves by the DTA. The works in [18] and [19] propose to build first the region graph of the automaton and then to build the MRgP as cross-product of the CTMC with that region graph. The region graph construction accounts for timed reachability and the successive product accounts for DTA acceptance of CTMC moves. This construction allows to devise a procedure that avoids the construction of non useful states (state that do not contribute to the probability of the $\top$ state) and it will the starting point for the on-the-fly model checking algorithm of Section 5.

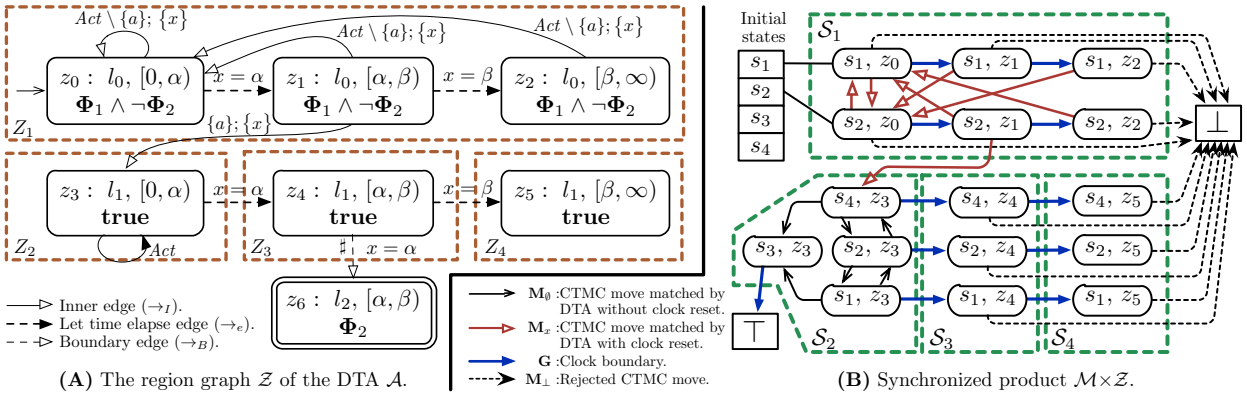
The region graph $\mathcal{G}(\mathcal{A})$ of $\mathcal{A}$ is constructed by combining the set of locations L with the set of regions $\mathcal{C}$ of Definition 7. This construction is the classical one for timed automata, simplified by the presence of a single clock. The set of *z-states*[1] Z is then subset of $L \times \mathcal{C}$ states reachable from an initial location l_0 with clock $x = 0$. The transition relation among z-states features three types of edges: *inner* edges that account for an inner edge in $\mathcal{A}$; *Time elapse* edges that account for the passage of time between two successive clock values; And *Reach next boundary* edges that account for a boundary edge in $\mathcal{A}$.

[1] In region graphs literature the term "state" is usually adopted, but since we use region graph states and CTMC states to compute MRgP states, we use the term "z-states" to avoid confusion.

Definition 12 (Region graph). *Given a DTA* $\mathcal{A} = \langle L, \Lambda_{\mathcal{A}}, L_0, L_F, AP, Inner, Boundary\rangle$ *of region set* $\mathcal{C}$ *built on the constants* $C = \{\delta_0 = 0, \delta_1, \ldots, \delta_m\}$*, the* Region Graph $\mathcal{G}(\mathcal{A})$ *is defined by the tuple* $\mathcal{Z} = \mathcal{G}(\mathcal{A}) = \langle Z, \Lambda, Z_0, Z_F, AP, \rightarrow_I, \rightarrow_e, \rightarrow_B\rangle$ *where* $Z \subseteq L \times \mathcal{C}$ *is a finite set of* z-states*,* $\Lambda : Z \rightarrow \mathcal{B}_{AP}$ *assigns to each z-state a boolean expression over the set* AP *of atomic propositions,* $Z_0 \subseteq L_0 \times [0, \delta_1)$ *is the set of* initial z-states*,* $Z_F \subseteq L_F \times \mathcal{C}$ *is the set of* final z-states*; the transition relation between regions is defined by the set of* inner edges $\rightarrow_I \subseteq Z \times 2^{Act} \times \{\emptyset, x\} \times Z$*, the set of* time-elapse edges $\rightarrow_e \subseteq Z \times Z$*, and the set of* reach next boundary edges $\rightarrow_B \subseteq Z \times Z$*, where:*

- $\Lambda(\langle l, [\delta_k, \delta_{k+1}\rangle) = \Lambda_{\mathcal{A}}(l)$*;*
- $Z_0 = \{\langle l_0, [0, \delta_1)\rangle \mid l_0 \in L_0\}$*;*
- *(*inner edge*) Given* $z = \langle l, [\delta_k, \delta_{k+1})\rangle \in Z$ *and the DTA inner edge* $l \xrightarrow{\hat{c}, A, r} l'$*, where* $[\delta_k, \delta_{k+1}) \in \hat{c}$*, then* $z' = \langle l', [\delta_k, \delta_{k+1})[r := 0]\rangle$[2] $\in Z$ *and* $(z, A, r, z') \in \rightarrow_I$*;*
- *(*time-elapse edge*) Given* $z = \langle l, [\delta_{k-1}, \delta_k)\rangle \in Z$*,* $k \leq m$*, then* $z' = \langle l, [\delta_k, \delta_{k+1})\}\rangle \in Z$ *and* $(z, z') \in \rightarrow_e$*;*
- *(*reach next boundary edge*) Given* $z = \langle l, [\delta_k, \delta_{k+1})\rangle \in Z$*, and the DTA boundary edge* $l \xrightarrow{x=\delta_k, \sharp, r} l'$*, then* $z' = \langle l', [\delta_k, \delta_{k+1})[r := 0]\rangle$*,* $z' \in Z$ *and* $(z, z') \in \rightarrow_B$*;*

This definition is actually an extension of the region graph for CSL$^{\text{TA}}$ defined in [19] to include boundary edges, and of that in [18] to eliminate "point regions" (regions $[\delta_k, \delta_k]$), to avoid the creation of states of the MRgP that are entered and exited in zero time. Note that the region graph here defined is semantically close to the classical construction for timed automata, but its representation is slightly different, to ease the subsequent $\mathcal{M}\times\mathcal{Z}$ construction. *Inner* edges of the region graph contain the explicit indication of the reset, to distinguish whether an inner edge that starts and ends in the first region includes a reset or not; *Time elapse* edges are distinct from boundary edges; *Boundary* edges can only be triggered at the beginning of the region interval, while time elapse edges are taken at the end of the region interval.



(A) The region graph $\mathcal{Z}$ of the DTA $\mathcal{A}$. **(B)** Synchronized product $\mathcal{M}\times\mathcal{Z}$.

Figure 6: The region graph $Z = \mathcal{G}(\mathcal{A})$ and the MRgP $\mathcal{M}\times\mathcal{Z}$ for the DTA $\mathcal{A}$ and the CTMC $\mathcal{M}$ of Figure 3.

Example 9 (Region graph). *Figure 6(A) shows the region graph of the DTA of Figure 3(B). z-states are organized into rows, each row corresponds to a location of the DTA, and transitions from left to right correspond to the elapsing of time. Final z-states are graphically identified by a double border. Transition back to* z_0 *are due to the self-loop over location* l_0*, transition from* z_1 *to* z_3 *is due the DTA edge from* l_0 *to* l_1*. The transition from* z_4 *to* z_6 *is an example of how boundary edges are taken as soon as the region is entered: the* $\rightarrow_B$ *transition leading to* z_6 *can be taken if* $x = \alpha$*, while the* $\rightarrow_e$ *transition leading to* z_5 *is taken when the clock reaches* β*, the end of the region.*

The synchronized product $\mathcal{M}\times\mathcal{Z}$ is a MRgP whose states are pairs $\langle s, z\rangle$ of a CTMC state s and a z-state z, plus two special absorbing states $\top$ and $\bot$. The construction follows the same structure as the $\mathcal{M}\times\mathcal{A}$ construction of definition 8. In the definition if $z = \langle l, c\rangle$, then $region(z) = c$.

[2] Remember that $[\delta_k, \delta_{k+1})[r := 0]$ is $[\delta_k, \delta_{k+1})$ if $r = \varnothing$ and $[0, \delta_1)$ if $r = \{x\}$. Moreover, for uniformity, $\delta_{k+1} = \infty$)

Definition 13 (Synchronized product $\mathcal{M}\times\mathcal{Z}$). *The synchronized product of a CTMC $\mathcal{M}$ and a region graph $\mathcal{Z} = \mathcal{G}(\mathcal{A})$ is the smallest MRgP $\mathcal{R} = \langle \mathcal{S}, G, \Gamma, \mathbf{Q}, \bar{\mathbf{Q}}, \mathbf{\Delta}\rangle$ defined by:*

- *$\mathcal{S} \subseteq (S \times Z) \cup \{\top, \bot\}$. We indicate with $\langle s, z\rangle$ a generic state in $S \times Z$.*
- *The set G of general events has one deterministic event g_k for each region $[\delta_{k-1}, \delta_k) \in \mathcal{C}$ of finite duration, i.e. $1 \le k \le m$ with $m = |\mathcal{C}|$.*
- *Let $\Gamma(s, z)$ be defined as g_k if $region(z) = [\delta_{k-1}, \delta_k)$ and $k \le m$ and E otherwise.*
- *Let $fin : (S \times Z) \times (S \times Z) \cup \{\top\}$ be a function defined as $fin(s, z) = \top$ if $z \in Z_F$, and $fin(s, z) = \langle s, z\rangle$ otherwise.*
- *Let $closure : (S \times Z) \times (S \times Z) \cup \{\top\}$ be defined recursively over boundary edges as $closure(s, z) = closure(s, z')$ iff exists $(z, z') \in \to_B$ with $s \models \Lambda(z')$; and as $closure(s, z) = fin(s, z)$ otherwise.*
- *The tuples $\langle s, z\rangle \in \mathcal{S}$ and the matrices $\mathbf{Q}, \bar{\mathbf{Q}}, \mathbf{\Delta}$ are defined through the following rules:*
  1. *For each $s \in S$, if there is an initial location $z_0 \in Z$ with $s \models \Lambda(z_0)$, then $closure(s, z_0) \in \mathcal{S}$.*
  2. ***(G:** time elapse) Given $\langle s, z\rangle$ and $\langle z, z'\rangle \in \to_e$, then $closure(s, z') \in \mathcal{S}$ and $\mathbf{\Delta}[\langle s, z\rangle, closure(s, z')]=1$*
  3. ***(M:** CTMC transition. Given $\langle s, z\rangle \in \mathcal{S}$ and the CTMC transition $s \xrightarrow{a,\lambda} s'$, then:*
     - ***($\boldsymbol{M}_\emptyset$)** if $\exists(z, A, \varnothing, z') \in \to_I$ with $a \in A \wedge s' \models \Lambda(z')$, then $fin(s', z') \in \mathcal{S}$ and $\mathbf{Q}$ and $\bar{\mathbf{Q}}$ are defined as follows:* $\begin{cases} \mathbf{Q}[\langle s, z\rangle, fin(s', z')] = \lambda & \text{if } \Gamma(\langle s, z\rangle) = E \text{ or } fin(s', z') \neq \top \\ \bar{\mathbf{Q}}[\langle s, z\rangle, \top] = \lambda & \text{if } \Gamma(\langle s, z\rangle) \neq E \text{ and } fin(s', z) = \top \end{cases}$
     - ***($\boldsymbol{M}_x$)** if $\exists(z, A, \{x\}, z') \in \to_I$ with $a \in A \wedge s' \models \Lambda(z')$, then $closure(s', z') \in \mathcal{S}$ and $\mathbf{Q}$ and $\bar{\mathbf{Q}}$ are defined as follows:* $\begin{cases} \bar{\mathbf{Q}}[\langle s, z\rangle, closure(s', z_0)] = \lambda & \text{if } region(z) \text{ has finite duration.} \\ \mathbf{Q}[\langle s, z\rangle, closure(s', z_0)] = \lambda & \text{otherwise.} \end{cases}$
     - ***($\boldsymbol{M}_\bot$)** if $\nexists(z, A, r, z') \in \to_I$ with $a \in A \wedge s' \models \Lambda(z')$ (no edge in $\mathcal{Z}$ matches the CTMC transition), and $\mathbf{Q}$ and $\bar{\mathbf{Q}}$ are defined as follows:* $\begin{cases} \bar{\mathbf{Q}}[\langle s, z\rangle, \bot] = \lambda & \text{if } region(z) = [\delta_{k-1}, \delta_k) \text{ and } k \le m \\ \mathbf{Q}[\langle s, z\rangle, \bot] = \lambda & \text{otherwise.} \end{cases}$

Example 10 (Example of a synchronized product $\mathcal{M}\times\mathcal{Z}$). *Figure 6(B) shows the synchronized product $\mathcal{M}\times\mathcal{Z}$ for the example of Figure 3. The graphical notation for the edges is the same as in Figure 3(D) and the definition and naming of the z-states is given in Figure 6(A). Note that only s_1 and s_2 give rise to initial states, since $\Lambda(z_0) = (\mathbf{\Phi}_1 \wedge \neg\mathbf{\Phi}_2)$ and only s_1 and s_2 satisfy that condition. Let us consider an example of application of each one of the rules above. The transition from $\langle s_1, z_0\rangle$ to $\langle s_1, z_1\rangle$ is produced by rule **G**: the Markov chain state is the same, the location is the same (l_0) but the clock region of z_1 is the next region of that in z_0. The transition from $\langle s_2, z_1\rangle$ to $\langle s_4, z_3\rangle$ is produced by rule $\boldsymbol{M}_x$: the Markov chain transition of label a from s_2 to s_4 is accepted by the DTA edge from l_0 to l_1, which is also labelled a, moreover z_1 is in the $[\alpha, \beta)$ region, so that the clock constraint on the DTA edge is satisfied; since the edge has an associated clock reset, the MRgP moves from $region(z_1) = [\alpha, \beta)$ to $region(z_3) = [0, \alpha)$. The transition from $\langle s_4, z_3\rangle$ to $\langle s_2, z_3\rangle$ is produced by rule $\boldsymbol{M}_\emptyset$: the Markov chain transition from s_4 to s_2 is accepted by the self loop on z_3. The transition from $\langle s_1, z_0\rangle$ to $\bot$ is produced by rule $\boldsymbol{M}_\bot$ when the CTMC moves from s_1 to s_3, since s_3 is a $\mathbf{\Phi}_2$-state and neither the self loop over z_0 can accept this transition, nor the edge from z_0 to z_1 can. The $\top$ state can be reached through the application of the closure and fin functions: in the MRgP example there is a time-elapse transition from $\langle s_3, z_3\rangle$ when $x = \alpha$ that expands as:*

$$\langle s_3, z_3\rangle \xrightarrow{\mathbf{G}} closure(s_3, z_4) \to closure(s_3, z_6) \to fin(s_3, z_6) \to \top \tag{5}$$

since $z_6 \in Z_F$, and $s_3 \models \Lambda(z_6)$. Hence, the time-elapse transition from $\langle s_3, z_3\rangle$ goes directly to $\top$.

The MRgP $\mathcal{M}\times\mathcal{Z}$ of Figure 6(B) and the MRgP $\mathcal{M}\times\mathcal{A}$ of Figure 3(D) identify the same MRgP process, up to the obvious mapping $\langle s, l, c\rangle \to \langle s, z\rangle$, as indeed stated by the next theorem. Since the two MRgPs are identical, the model checking of a CSL$^{\text{TA}}$ formula can be based on any of the two.

Theorem 1. *The MRgPs represented by $\mathcal{M}\times\mathcal{A}$ and $\mathcal{M}\times\mathcal{Z}$ identify the same process.*

Proof. The proof is reported in Appendix B, since it is rather mechanical, moreover the possibility of using a region graph (instead of the DTA) to compute the MRgP was already proven in [19], based on the proofs in [29] (although the DTA used in [19] has some differences, the structure of the proof is the same). □

The full MRgP solution of $\mathcal{M}\times\mathcal{Z}$ are called $\text{Full}^{\mathcal{Z}}_{\text{fwd}}$ and $\text{Full}^{\mathcal{Z}}_{\text{bwd}}$ (see Figure 1). The Component Method can obviously be applied also to the MRgP $\mathcal{M}\times\mathcal{Z}$, leading to a variation of Algorithm 1 and 2 that considers as basic MRgP $\mathcal{M}\times\mathcal{Z}$ instead of $\mathcal{M}\times\mathcal{A}$: they are named $\text{Comp}^{\mathcal{Z}}_{\text{fwd}}$ and $\text{Comp}^{\mathcal{Z}}_{\text{bwd}}$, respectively.

4.1. Reduction based on non-useful z-states.

Consider the synchronized product of Figure 6(B): no states in partitions $\mathcal{S}_3$ and $\mathcal{S}_4$ are visited by any path leading to $\top$. Nevertheless they are computed, stored and are included in the MRgP matrices used in the solution process. These states can be removed and all the ingoing transitions are redirected to $\bot$. This elimination can be done after the full MRgP has been built, thus removing all states that, by combination of the conditions on the DTA and by the structure of the Markov chain, are not in any path to $\top$. With reference to the example in Figure 6(B), it corresponds to removing the states in $\mathcal{S}_3$ and $\mathcal{S}_4$.

Reducing the number of states to be considered by the solution process may save a significant amount of computation time for the numerical solutions. For example, in the MRgP of Figure 6(B), we save the cost of solving the MRgP components built on $\mathcal{S}_3$ and $\mathcal{S}_4$. Nevertheless, the amount of memory required to generate the state space is the same. Since the structure of the MRgP is strongly influenced by the structure of the region graph $\mathcal{Z}$, we can exploit this dependency to devise a modification of the $\mathcal{M}\times\mathcal{Z}$ construction that *only generates useful MRgP states*. With reference to the same example, we can observe that a MRgP state generated from the z-state z_5 will never lead to $\top$. We could therefore envision to simply discard z_5 before the $\mathcal{M}\times\mathcal{Z}$ construction, but this solution is too simplistic. Consider the z-state z_4: from Figure 6(B) it is immediate to observe that there is no MRgP state $\langle s, z_4\rangle$ that can lead to $\top$, for any choice of s, but the z-state z_4 of Figure 6(A) is nevertheless used in the $\mathcal{M}\times\mathcal{Z}$ construction, as part of the computation of the *closure* function for $\langle s_3, z_3\rangle$, as illustrated by the computation in Eq. 5. This example suggests that the $\mathcal{M}\times\mathcal{Z}$ construction should considered z_4 only for the computation of *closure*, and not for the generation of states of the form $\langle s, z_4\rangle$. The $\mathcal{M}\times\mathcal{Z}$ construction is therefore modified so as to consider a *tagging* of the z-states: tag NK (not keep) is assigned to z if no MRgP state $\langle s, z_\rangle$ can lead to $\top$, all other z-states are tagged K (keep). The tagging is then used to avoid the construction of all MRgP states with a NK z-state. This is achieved by modifying appropriately the *fin* function of the $\mathcal{M}\times\mathcal{Z}$ construction of Definition 13, so as to discard the state while maintaining the transitions required for the correct computation of the probability of reaching $\top$.

Definition 14 (z-states tagging). *Given a region graph $\mathcal{Z}$, we define the function $tag(z) : Z \to \{K, NK\}$ as*

- $tag(z) = K$ *(Keep state) if either $z \in Z_F$, or if there is at least one path from z to z', with $z' \in Z_F$ that includes at least an edge of type inner or time-elapse ($\to_I$ or $\to_e$).*
- $tag(z) = NK$ *(do Not Keep state) otherwise: either there is no path to an accepting z-state or the path consists only of boundary edges.*

The function *fin*, that is used in definition 13 to identify the MRgP transitions that go to $\top$, is changed so as to force a transition to $\bot$ when we get to a z-state tagged NK. The definition of $fin(s, z)$ as: $\top$ if $z \in Z_F$, or $\langle s, z\rangle$ otherwise, is therefore changed to:

$$fin(s,z) = \begin{cases} \top & \text{if } z \in Z_F. \\ \bot & \text{if } tag(z) = NK. \\ \langle s, z\rangle & \text{otherwise.} \end{cases} \tag{6}$$

With this change, $fin(s, z)$ may evaluate also to $\bot$, therefore also line (3) of the $\mathcal{M}\times\mathcal{Z}$ construction of definition 13 should change: whenever a CTMC transition leads to $\bot$, the contribution should go to $\bar{\mathbf{Q}}$, as it is the case when $fin(s, z)$ evaluates to $\top$. It is important to remark that non useful MRgP states are not deleted, but simply aggregated into the $\bot$ state, to ensure that the solution algorithms "sees" all transitions required to correctly compute the probability that the next states, at least for all next states that are on a path to $\top$.

Example 11 (Example of a reduced $\mathcal{M}\times\mathcal{Z}$). *Figure 7 shows the result of the modified $\mathcal{M}\times\mathcal{Z}$ construction for the same DTA and CTMC used for the example reported in Figure 6. The z-states z_4 and z_5 are tagged as NK, therefore edges departing from $\langle s_h, z_3\rangle$, $h \in \{1,2,4\}$, that in the $\mathcal{M}\times\mathcal{Z}$ of Figure 6 go into states $\langle s_h, z_4\rangle$ now reach directly $\bot$.*

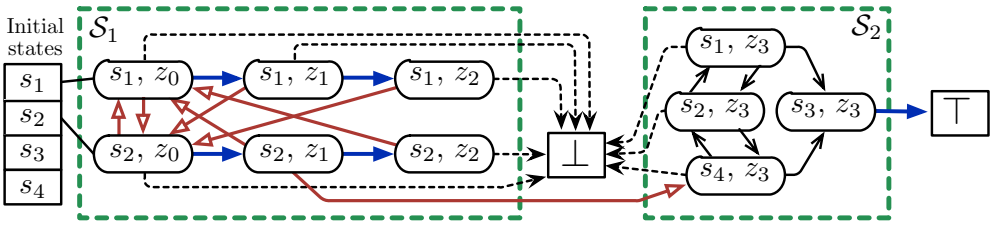


Figure 7: $\mathcal{M}\times\mathcal{Z}$ process of Figure 6(B) considering reachability of the final locations on the Region Graph.

Of course, z-states tagging does not guarantee that the resulting $\mathcal{M}\times\mathcal{Z}$ will have include states that lead to $\top$: a z-state could have an edge that reaches a final location, but the CTMC $\mathcal{M}$ may never activate that edge. We now prove that the modification in the *fin* function does not alter the probability of reaching state $\top$.

Theorem 2. *Given a MRgP $\mathcal{M}\times\mathcal{Z}$, let $\mathcal{M}\times\mathcal{Z}_{red}$ be the MRgP generated with the modified function fin of Equation 6. Given an initial state $\langle s, z\rangle$, the probability of reaching $\top$ from $\langle s, z\rangle$ in $\mathcal{M}\times\mathcal{Z}$ is equal to the probability of reaching $\top$ from the same state $\langle s, z\rangle$ in $\mathcal{M}\times\mathcal{Z}_{red}$.*

Proof. We need to prove that **(1)** $\mathcal{M}\times\mathcal{Z}_{red}$ includes all and only the states that in $\mathcal{M}\times\mathcal{Z}$ go to $\top$, and that **(2)** the transitions out of these states are maintained, including the ones that are part of paths that do not lead to $\top$, as they are required for the correct computation of the probability of reaching $\top$.

(1). If $\langle s, z\rangle$ is a state of $\mathcal{M}\times\mathcal{Z}$ that leads to $\top$ then $\langle s, z\rangle$ is a state of $\mathcal{M}\times\mathcal{Z}_{red}$. Assume the above is not true, then there exists $\langle s, z\rangle$ in $\mathcal{M}\times\mathcal{Z}$ that leads to $\top$ and $\langle s, z\rangle$ is not a state of $\mathcal{M}\times\mathcal{Z}_{red}$. If there is a path in $\mathcal{M}\times\mathcal{Z}$ from $\langle s, z\rangle$ to $\top$, then in the region graph $\mathcal{Z}$ there is a path from z to z', with $z' \in Z_F$. Moreover, due to the *closure* function (that does the transitive closure over boundary edges), and its use in Rules (2) and (3) of Definition 13, the path from $\langle s, z\rangle$ to $\top$ (1.a) is either made by boundary-edges only or (1.b) it contains at least an Inner or a time-elapse edge. Case (1.a) has no difference between $\mathcal{M}\times\mathcal{Z}$ and $\mathcal{M}\times\mathcal{Z}_{red}$, since it is just a recursive evaluation of *closure* up to a final location. Case (1.b) implies that $tag(z) = K$ since there is at least an inner or a time-elapse edge. Therefore, the same rule that adds $\langle s, z\rangle$ to $\mathcal{M}\times\mathcal{Z}$ would add $\langle s, z\rangle$ also to $\mathcal{M}\times\mathcal{Z}_{red}$ since the construction $\mathcal{M}\times\mathcal{Z}_{red}$ may replace states with $\bot$ only when $tag(z) = NK$. The "only-if" case is trivial since the modification of function *fin* may restrict the number of states considered, but does not add any state.

(2). If a state $\langle s, z\rangle$ is a state of $\mathcal{M}\times\mathcal{Z}$ that leads to $\top$ (and that, according to the previous point, is also part of $\mathcal{M}\times\mathcal{Z}_{red}$) then we consider the following situations:

(2.a). If $\langle s, z\rangle \xrightarrow{\mathbf{Q},\bar{\mathbf{Q}},\mathbf{\Delta}} \langle s', z'\rangle$ is a transition in $\mathcal{M}\times\mathcal{Z}$, and both $\langle s, z\rangle$ and $\langle s', z'\rangle$ are on a path to $\top$, then $\langle s, z\rangle \xrightarrow{\mathbf{Q},\bar{\mathbf{Q}},\mathbf{\Delta}} \langle s', z'\rangle$ is a transition in $\mathcal{M}\times\mathcal{Z}_{red}$. This statement is true since $tag(z) = tag(z') = K$ and therefore the algorithms for building $\mathcal{M}\times\mathcal{Z}$ and $\mathcal{M}\times\mathcal{Z}_{red}$ behave the same.

(2.b). If $\langle s, z\rangle \xrightarrow{\mathbf{Q},\bar{\mathbf{Q}},\mathbf{\Delta}} \langle s', z'\rangle$ is a transition in $\mathcal{M}\times\mathcal{Z}$, with $\langle s, z\rangle$ on a path to $\top$ and $\langle s', z'\rangle$ in no path to $\top$, with $tag(z') = NK$, then $\langle s, z\rangle \xrightarrow{\mathbf{Q},\bar{\mathbf{Q}},\mathbf{\Delta}} \bot$ is a transition in $\mathcal{M}\times\mathcal{Z}_{red}$. This statement is true since the $\mathcal{M}\times\mathcal{Z}$ construction evaluates $closure(\langle s', z'\rangle)$ as $fin(\langle s', z'\rangle)$. Since $tag(z') = NK$ the function evaluates to $\bot$.

(2.c). If $\langle s, z\rangle \xrightarrow{\mathbf{Q},\bar{\mathbf{Q}},\mathbf{\Delta}} \langle s', z'\rangle$ is a transition in $\mathcal{M}\times\mathcal{Z}$, with $\langle s, z\rangle$ on a path to $\top$ and $\langle s', z'\rangle$ in no path to $\top$, with $tag(z') = K$, then $\langle s, z\rangle \xrightarrow{\mathbf{Q},\bar{\mathbf{Q}},\mathbf{\Delta}} \langle s', z'\rangle$ is a transition in $\mathcal{M}\times\mathcal{Z}_{red}$. As in case (a), since $tag(z') = K$ the transition $\langle s, z\rangle \xrightarrow{\mathbf{Q},\bar{\mathbf{Q}},\mathbf{\Delta}} \langle s', z'\rangle$ is added also in the $\mathcal{M}\times\mathcal{Z}_{Red}$ construction.

Points 2.a to 2.c imply that all transitions out of a state $\langle s, z\rangle$ of $\mathcal{M}\times\mathcal{Z}$ are maintained in $\mathcal{M}\times\mathcal{Z}_{red}$ if $\langle s, z\rangle$ is a state that leads to $\top$. Note that a transition which is in $\mathbf{Q}$ of $\mathcal{M}\times\mathcal{Z}$ may end-up in $\bar{\mathbf{Q}}$ of $\mathcal{M}\times\mathcal{Z}_{red}$ (see the comment following Equation 6), but this does not alter probability distribution of successor states, for those states on paths

leading to $\top$. The combination of point **(1)** and **(2)** above allows us to conclude that the portions of $\mathbf{Q}$, $\bar{\mathbf{Q}}$, and $\mathbf{\Delta}$ that are used in the computation of probability of eventually reaching $\top$ in $\mathcal{M}\times\mathcal{Z}$ and $\mathcal{M}\times\mathcal{Z}_{Red}$ are the same. □

The idea of removing non-useful states based on the region graph was first presented in [18]: a region graph is built and it is then reduced to remove non-tangible z-states and z-states that, under no conditions, can reach $\top$. This reduction led to a new form of region graph in which conditions move from z-states to edges among z-states, thus requiring a new definition of the MRgP $\mathcal{M}\times\mathcal{Z}$. The construction based on Definition 14 and Equation 6 allows instead to use the same $\mathcal{M}\times\mathcal{Z}$ construction while producing a reduced MRgP. As we shall see in Section 7, this reduction is crucial to allow CSL$^{\text{TA}}$ model checking based on OTF to attain the same memory performances as CSL model-checkers on CSL formulas.

5. On-the-fly model checking of CSL$^{\text{TA}}$.

Starting from the observation that the structure of the MRgP is strongly influenced by the structure of the DTA, or better, by its region graph, we propose to build the MRgP components based on $\mathcal{Z}$ components, leading to an *on-the-fly construction* of the synchronized process $\mathcal{M}\times\mathcal{Z}$ that allows us to construct and solve one component at a time, thus reducing the memory consumption of the model checker. The method, identified as case OTF in Figure 1, works with a *valid partition* $\{Z_j\}_{1\le j\le J}$ of the states Z of the region graph. For each region graph component Z_j, the method computes the synchronized product $\mathcal{M}\times\mathcal{Z}_j$, and then performs the numerical computation. Unlike Algorithms 1 and 2, there is not a global view of the entire state space of the synchronized process $\mathcal{M}\times\mathcal{Z}$. As a consequence, the forward on-the-fly method is different from the backward on-the-fly method, as we shall discuss in Section 7.3.

The first step is to define the region graph components.

Definition 15. *A* region graph component Z_j *is a subset of the set of z-states Z of the region graph $\mathcal{Z}$.*

It is convenient to distinguish inner edges $\rightarrow_I$ that have an associated clock reset ($\rightarrow_{I_x}$) from those with no associated clock reset ($\rightarrow_{I_\emptyset}$). Same distinction for $\rightarrow_B$.

In the Component Method, for each component $\mathcal{S}_i$ it is necessary to compute its augmented set $\widehat{\mathcal{S}_i}$ and its frontier set $frontier(\mathcal{S}_i)$: analogous constructions are required for the region graph components. Recalling that the augmented set is built based on transitions that are in $\mathbf{Q}$ (accepted CTMC moves without a clock reset), and that these moves are accepted by $I_\emptyset$ edges of the DTA, we have the following definition.

Definition 16 (Augmented set of a region graph component). *Let $Z_j \subseteq Z$ be a set of z-states of the region graph $\mathcal{Z}$ and let us indicate with $z \overset{*}{\rightarrow}_{I_\emptyset} z'$ that there exists a path between z and z' made only of $\rightarrow_{I_\emptyset}$ edges. The* augmented set $\widehat{Z_j}$ *of Z_j is defined as the largest set such that:*

$$\widehat{Z_j} = Z_j \cup \left\{ z' \in Z \setminus Z_j \;\middle|\; \exists z \in Z_j \;:\; region(z) \neq [\delta_m, \infty) \;\wedge\; z \overset{*}{\rightarrow}_{I_\emptyset} z' \right\}$$

Note that the augmented set is the set itself if the component is entirely in the $[\delta_m, \infty)$ region. Recalling that the *frontier* function for an MRgP components $\mathcal{S}_i$ is defined (Definition 10) based on $\mathbf{Q}$ transitions when $\Gamma(\mathcal{S}_i) = E$ and on $\bar{\mathbf{Q}}$ and $\mathbf{\Delta}$ when $\Gamma(\mathcal{S}_i) \neq E$, and that $\bar{\mathbf{Q}}$ and $\mathbf{\Delta}$ transitions are generated by the presence of I_x and let time elapse edges in the region graph, we can define the *frontier* function of a region graph component as follows.

Definition 17 (Frontier of a region graph component). *Let $Z_j \subseteq Z$ be a set of z-states of a region graph $\mathcal{Z}$. The* forward frontier *of Z_j is defined as the largest set of z-states such that:*

$$\begin{aligned} frontier(Z_j) = &\left\{ z' \;\middle|\; z' \notin Z_j \;\wedge\; \exists\, z \in \widehat{Z_j} : \left(region(z) = [\delta_m, \infty) \wedge z \rightarrow_{I_\emptyset} z' \right) \right\} \cup \\ &\left\{ atbound(z') \;\middle|\; z' \notin Z_j \;\wedge\; \exists\, z \in \widehat{Z_j} : \left(region(z) \neq [\delta_m, \infty) \wedge (z \rightarrow_{I_x} z' \;\vee\; z \rightarrow_e z') \right) \right\} \end{aligned} \tag{7}$$

where $atbound(z')$ is defined recursively as:

$$atbound(z) = \{z\} \cup \left\{ atbound(z') \;\middle|\; \exists\, z' \notin Z_j :\; z \rightarrow_B z' \right\} \tag{8}$$

Example 12 (Frontier of a region graph component)**.** *In the region graph of Figure 6(A), the frontier of Z_1 is z_3, while the one of Z_2 includes both z_4 and z_6, with the latter region included by the $atbound$ function due to edge $z_4 \rightarrow_{B_\emptyset} z_6$.*

Similarly to Def. 10, the frontier of a region graph component is not necessarily disjoint from its augmented set, since the same z-state $z \in \widehat{Z_j}$ could be reached by both an inner edge and a time elapse edge (or an edge with a clock reset) from $\widehat{Z_j}$ states, making it both a member of the augmented set and of the frontier set. As for the MRgP case, if a state is part of both the augmented set and the frontier, then it is duplicated. Based on the above definition, we can now introduce the notion of the region graph $\mathcal{Z}_j$ generated by the z-states of a region graph component Z_j.

Definition 18 (**Region graph $\mathcal{Z}_j$ of component Z_j**)**.** *Given a set of z-states $Z_j \subseteq Z$, the component region graph $\mathcal{Z}_j$ is defined as the projection of $\mathcal{Z}$ over the z-states $\widehat{Z_j} \cup frontier(Z_j)$. Edges $\rightarrow_I, \rightarrow_e, \rightarrow_B$ are defined as the edges of $\mathcal{Z}$ that have a source z-state in the set $\widehat{Z_j}$. All edges that have a source in $frontier(Z_j)$ are removed, but for the boundary ones $\rightarrow_B$ (these are the edges used by the $atbound$ function to build the frontier itself). The set of final z-states is $Z_F \cap \big(\widehat{Z_j} \cup frontier(Z_j)\big)$, where Z_F is the the set of final z-states of $\mathcal{Z}$. The region set $\mathcal{C}$ of $\mathcal{Z}_j$ remains the same of $\mathcal{Z}$.*

Example 13 (Region graph $\mathcal{Z}_j$ of a component Z_j)**.** *As an example of region graph of a component we can take the region graph generated by Z_2 which includes z_3 and its self loop, the time-elapse edge from z_3 to z_4, the z-state z_4, the boundary edge from z_4 to z_6 and z_6 itself.*

Note that not all frontier z-states are made fully absorbing, as the $\rightarrow_B$ are retained, to ensure a proper evaluation of the *closure* function in the $\mathcal{M}\times\mathcal{Z}_j$ construction.

Definition 19 (**Synchronized product $\mathcal{M}\times\mathcal{Z}_j$** from a set S_0 of initial states)**.** *Since each $\mathcal{Z}_j$ is a region graph, the cross product $\mathcal{M}\times\mathcal{Z}_j(S_0)$ is defined following the rules of Definition 13 using S_0 as initial states.*

5.1. On the fly algorithm, forward

Algorithm 3 defines the on-the-fly forward model checking procedure that uses the Component Method with the $\mathcal{M}\times\mathcal{Z}_j$ components. Each $\mathcal{M}\times\mathcal{Z}_j$ component is generated only when it is required by the computation and it is then deleted afterwards. Each component needs to be generated only once. Components are considered, and generated, in *forward topological order*: a component is considered in step k only if all components of the states from any paths from the initial components to the component itself have already taken into account in the previous $k-1$ steps.

Algorithm 3 Pseudocode of the forward on-the-fly model checking method.

function MODELCHECK-OTF$_{\text{FWD}}$($\{Z_j\}$ components of $\mathcal{Z}$, s_0 : initial state)
  $\boldsymbol{\pi}^{(0)} : S \times Z \rightarrow \mathbb{R}$ ▷ sparse vector of state probabilities
  $\boldsymbol{\pi}^{(0)}[\langle s_0, l_0, 0\rangle] \leftarrow 1$
  Identify a set $\{Z_j\}$ of components of the states of $\mathcal{Z}$ and build the corresponding $\{\mathcal{Z}_j\}$ as per definition 18.
  for each $\mathcal{Z}_j$, taken in forward topological order **do**
    H_j = all the tuples $\langle s, z\rangle$ with $\boldsymbol{\pi}^{(j-1)}[\langle s, z\rangle] \neq 0 \wedge z \in Z_j$
    if $H_j \neq \varnothing$ **then**
      Let $\mathbf{I}_{H_j}$ be the filtering matrix of H_j
      Construct $\mathcal{M}\times\mathcal{Z}_j(H_j)$ as per definition 19. Let $\mathbf{I}_{H_j} \cdot \boldsymbol{\pi}^{(j-1)}$ be the initial distribution.
      Compute the probability $\boldsymbol{\mu}_i$ outgoing $\mathcal{M}\times\mathcal{Z}_j$ and reaching states $\langle s, z\rangle \in frontier(\mathcal{M}\times\mathcal{Z}_j)$.
      $\boldsymbol{\pi}^{(j)} \leftarrow (\mathbf{I} - \mathbf{I}_{H_j}) \cdot \boldsymbol{\pi}^{(j-1)} + \boldsymbol{\mu}_i$
    end if
  end for
  return $\boldsymbol{\pi}^{(K)}[\top]$
end function

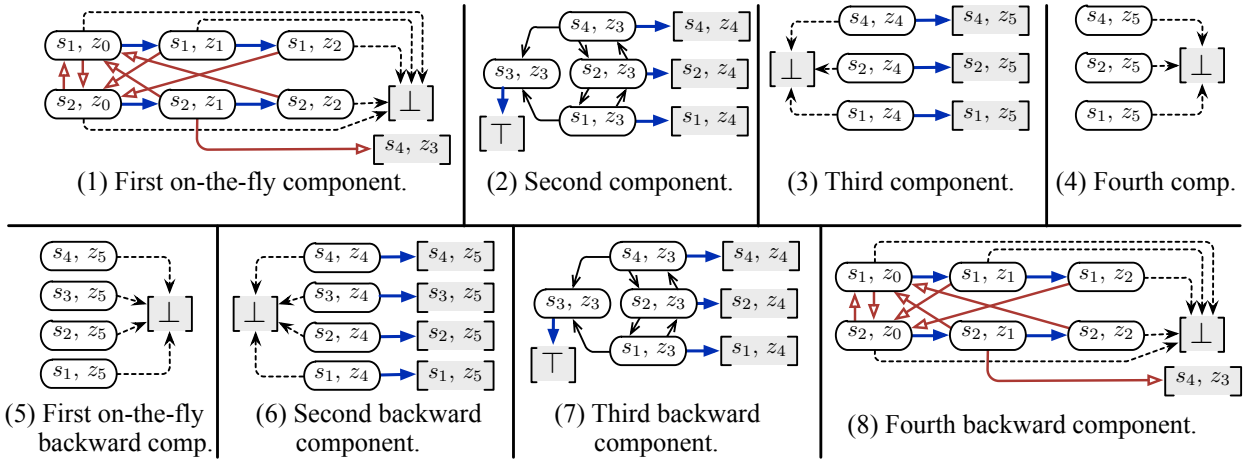


Figure 8: Components generated by the MODELCHECK-OTF$_{\text{FWD}}$ (above) and MODELCHECK-OTF$_{\text{BWD}}$ (below) algorithm on the example of Figure 6.

Example 14 (Execution of Algorithm 3)**.** *Figure 8, upper part, depicts, in the order, the four components generated and solved by Algorithm 3. The first component corresponds to $\mathcal{M}\times\mathcal{Z}_1$, where $\mathcal{Z}_1$ is the region graph generated from the component Z_1 in Figure 6 according to Definition 18. The augmented set of Z_1 is the set itself, the frontier is z_3, therefore the frontier of the MRgP component $\mathcal{M}\times\mathcal{Z}_1$ are the states $\langle s_4, z_3\rangle$ and $\perp$ that receive, according to the steady-state solution of the component, the probability initially accumulated in the initial state $\langle s_1, z_0\rangle$. The second component is $\mathcal{M}\times\mathcal{Z}_2$, whose frontier states are built starting from $frontier(Z_2) = \{z_4, z_6\}$. The component is depicted in Figure 8(2), and the gray states are the frontier ones. The initial probability of the component, that derives from the solution of the component in Figure 8(1), is all concentrated in $\langle s_4, z_3\rangle$ and it gets distributed to $\top$ and to the other frontier states. Note that $\top$ is generated from the z-state z_6. The algorithm then proceeds in building and solving the third and fourth component (Figure 8(3) and (4)), but none of the two solutions adds probability to $\top$.*

To prove the correctness of Algorithm 3 we first need to show that $\{\mathcal{M}\times\mathcal{Z}_j\}_{1\le j\le J}$ is an acyclic set of components.

Theorem 3. *If $\{Z_j\}_{\{1\le J\}}$ is an acyclic set of components of $\mathcal{Z}$, then the set $\{\mathcal{M}\times\mathcal{Z}_j\}_{\{1\le J\}}$ is an acyclic set of components of the MRgP $\mathcal{M}\times\mathcal{Z}$.*

Proof. By contradiction, assume that there exists $\mathcal{M}\times\mathcal{Z}_i$ and $\mathcal{M}\times\mathcal{Z}_k$ ($i \neq k$) with a cyclic path in the MRgP from $\mathcal{M}\times\mathcal{Z}_i$ to $\mathcal{M}\times\mathcal{Z}_k$ and vice-versa. Since $\perp$ and $\top$ are absorbing states, they cannot be part of a cyclic path so let's assume, without loss of generality, that the cyclic path does not include $\perp$ and $\top$ and it is determined by two transitions: a transition e_{ik} from a state $\langle s_i, z_i\rangle$ of $\mathcal{M}\times\mathcal{Z}_i$ to state $\langle s_k, z_k\rangle$ of $\mathcal{M}\times\mathcal{Z}_k$, and a transition e_{ki} from $\langle s'_k, z'_k\rangle$ of $\mathcal{M}\times\mathcal{Z}_k$ to $\langle s'_i, z'_i\rangle$ of $\mathcal{M}\times\mathcal{Z}_i$. Since z_i (and z'_i) are in a different components than z_k (and z'_k) then $z_i \neq z_k$ (and $z'_i \neq z'_k$). Therefore the transitions e_{ik} and e_{ki} could be present in the MRgP only if in the region graph there were at least an edge from z_i to z_k and from z'_k to z'_i, but this will imply that $\{Z_j\}$ is not an acyclic set of components, which violates the hypothesis. □

Using Theorem 3 we can then prove the correctness of Algorithm 3.

Theorem 4. *Algorithm 3 correctly computes the probability of eventually reaching the $\top$ state of a MRgP $\mathcal{M}\times\mathcal{Z}$*

Proof. Since the Component Method was already shown to be correct [17], we only need to show that 1) we are using a set of components which is acyclic and that it is a partition of the state space, and that 2) the input and output states of the components are correctly built and used by the on-the-fly technique.

Theorem 3 proves point 1, since we know that the set of components is acyclic and that the components are a partition of the state space, since $\{Z_j\}$ is a partition of the region graph $\mathcal{Z}$. Note that the $\{Z_j\}$ z-states constitute a partition, while the $\{\widehat{Z_j}\}$ do not, exactly as for the MRgP components.

To prove point 2 we only need to show that, for each component, the correct set of states in the augmented set and in the frontier is identified. The MRgP component $\mathcal{M}\times\mathcal{Z}_j$ considered in each iteration of the algorithm is built as the synchronized product of $\mathcal{M}$ and $\mathcal{Z}_j$, where, according to Def. 18, the construction of the region graph $\mathcal{Z}_j$ is based on the Z_j states, *and* on the states of $\widehat{Z_j}$ and $\mathit{frontier}(Z_j)$.

Correct Augmented set: Def. 9 states that the augmented set of a MRgP component includes all states reachable from the component through a path of one or more exponential events that do not preempt a general event. $\widehat{Z_j}$ includes all z-states reachable through a $\rightarrow_{I_\emptyset}$ edge from a z-state in Z_j, which are exactly the edges that accepts Markovian moves that do not preempt a general event. Since the construction of $\mathcal{M}\times\mathcal{Z}_j$ is based on the region graph $\mathcal{Z}_j$, which includes the augmented set and the frontier of Z_j, then $\mathcal{M}\times\mathcal{Z}_j$ will include all the states of the MRgP component augmented set.

Correct Frontier Set: according to Def. 10, the frontier in a MRgP component is the set of states reachable in one step from the augmented set of the component (through a $\mathbf{Q}$, $\bar{\mathbf{Q}}$, or $\mathbf{\Delta}$ entry). If the whole MRgP is built, determining the frontier is a trivial task, but since in Algorithm 3 the full MRgP is not available, we need to be sure that $\mathcal{Z}_j$ includes all the z-states that, in the synchronized product $\mathcal{M}\times\mathcal{Z}_j$, will allow to correctly reach all frontier states, which can be done by inspecting all the rules of the $\mathcal{M}\times\mathcal{Z}$ construction in Def. 13. The only non trivial part is the *closure* construction, in which the end state of a transition that preempts the clock or that lets the time elapse is determined by following the *longest path through boundary edges*, which may pass through different regions. This is mimicked, in the definition of $\mathit{frontier}(Z_j)$, by including in the frontier the set of z' z-states identified by the $\mathit{atbound}$ function, which are *all the z-states potentially reachable through a path of boundary edges*. The reason while this is required is better explained through an example. Assume that in the region graph there is a path $z \rightarrow_e z' \rightarrow_B \cdots \rightarrow_B z''$, and that $z \in \widehat{Z_j}$ and $z', \ldots, z'' \notin \widehat{Z_j}$. From a state $\langle s, z\rangle$ in the synchronized product we may end up, through the *closure* computation, to either $\langle s, z\rangle$ or any of $\langle s, z'\rangle, \ldots, \langle s, z''\rangle$, depending on the state propositions associated to s and the state proposition expressions on $z, z', \ldots, z''$. Before the actual construction of $\mathcal{M}\times\mathcal{Z}_j$ takes place we do not know in advance the properties of any possible state s combined with z. Therefore all the z-states $\{z', \ldots, z''\}$ have to be considered as potentially reachable z-states in the component frontier from the time-elapse edge $z \rightarrow_e z'$. □

Note that in the proof we do not assume that the Component Method and the on-the-fly method work with the same set of components, as indeed this is not true as shown in Section 6.

5.2. On the fly algorithm, backward

Pseudocode of Algorithm 4 describes the backward on-the-fly model checking method. The structure of the procedure is similar to Algorithm 2 and the component construction is similar to that of Algorithm 3, with three main distinctions: 1) subsets are evaluated in the opposite order of the forward case, from the $\top$ state to the initial components; 2) computation follows the backward formula of Equation (3); and 3) obviously the construction of the synchronized process $\mathcal{M}\times\mathcal{Z}_j$ for a component $\mathcal{Z}_j$ is forward, but, for the backward on-the-fly algorithm, this construction does not know the set of initial states (the H_j sets of states with non-null probability of Algorithm 3) since, in backward, they are not known. The construction is therefore based on a set of potential initial states: the set $\mathcal{S}_0$ of all states $\langle s, z\rangle$, in which $s \models \Lambda(z)$. Working with *potential states* may have performance implications, as will be discussed in Sections 7.3 and 7. In any case, the number of MRgP states is bound by $|S| \cdot |\widehat{Z_j}|$, for every subset Z_j.

Example 15 (Execution of Algorithm 4)**.** *Figure 8, bottom part, depicts, in the order, the components built and solved by the backward on-the-fly algorithm. The first component is built from $\mathcal{Z}_4$, but since the reward of the frontier states ($\bot$) is null the component is built but the solution is skipped. The second component is built from $\mathcal{Z}_3$, and again no solution is computed since all states have a null reward. The third component is built from $\mathcal{Z}_2$: $\top$ is initially assigned a reward of 1, all the other states get 0. The computation then assigns a non-null reward to all states, and leaves unchanged the rewards of the frontier states ($\top$ and $\bot$). For the fourth component (the one built from $\mathcal{Z}_1$), the frontier has only $\langle s_4, z_3\rangle$, and therefore its reward is used in the backward computation for the fourth component. At the end the computed reward vector contains, for each state, the probability of eventually reaching $\top$. Note that OTF backward builds larger components than OTF forward (compare components in Figure 8(6) and (7) with those in 8(4) and (3)), but it may solve a smaller number of components, since when the input reward is null the solution is skipped.*

The correctness of the backward on-the-fly algorithm is proved by the following theorem:

Algorithm 4 Pseudocode of the backward on-the-fly model checking method.

function MODELCHECK-OTF$_{\text{BWD}}$($\{Z_j\}$ components of $\mathcal{Z}$)
  $\mathbf{r}^{(j)} : S \times Z \to \mathbb{R}$ ▷ sparse vector of per-state acceptance probabilities
  $\mathbf{r}^{(j)}[\top] \leftarrow 1$
  Identify a DAG of $\{Z_j\}$ components of the states of $\mathcal{Z}$ and build the corresponding $\{\mathcal{Z}_j\}$ as per definition 18.
  for each $\{\mathcal{Z}_j\}$ of $\mathcal{Z}$ in backward topological order **do**
    Construct, as per definition 19, $\mathcal{M}\times\mathcal{Z}_j(S_0)$, where $S_0 = \langle s, z\rangle : z \in Z_j \wedge s \models \Lambda(z)$.
    Compute the reward vector $\boldsymbol{\xi}^{(j)}$ for $\mathcal{M}\times\mathcal{Z}_j$ states, starting from the frontier rewards $\mathbf{r}^{(j)}$, if $\mathbf{r}^{(j)} \neq 0$.
    $\mathbf{r}^{(j+1)} \leftarrow \boldsymbol{\xi}^{(j)} + \mathbf{r}^{(j)}$
  end for
  $\mathbf{r}^{(K)}[s]$ contains $Prob\{s \text{ reaches } \top\}$, for all $s \in \mathcal{S}$.
  return $\mathbf{r}^{(K)}$
end function

Theorem 5. *Algorithm 4 correctly computes, for all states in $\mathcal{M}\times\mathcal{Z}$, the probability of reaching the $\top$ state.*

Proof. The proof takes advantage of what has already been proven for the forward case. Indeed it was already shown that the $\mathcal{M}\times\mathcal{Z}_j$ construction correctly computes the augmented set and the frontier states of the component. The only issue is whether the use of a potential state space may impair the computation. Assume the full state space $\mathcal{S}$ has been computed a-priori, and let $\mathcal{S}_j$ be the set of states $\langle s, z\rangle$ such that $z \in \mathcal{Z}_j$. Since $s \models \Lambda(z)$ is only a necessary condition for $\langle s, z\rangle$ to be a reachable state, clearly the algorithm considers a set of initial states which is a superset of all the states in $\mathcal{S}_j$ that have a transition incoming from a $\langle s', z'\rangle$ state with $z' \notin \mathcal{Z}_j$. Consequently, the states of $\mathcal{M}\times\mathcal{Z}_j$ are a superset of $\mathcal{S}_j$. To prove that the presence in $\mathcal{M}\times\mathcal{Z}_j$ of a state $\langle s, z\rangle \notin \mathcal{S}_j$ does not alter the computation, we can observe that the algorithm, when computing the reward vector $\boldsymbol{\xi}^{(j)}$ for $\mathcal{M}\times\mathcal{Z}_j$ states (starting from the frontier rewards $\mathbf{r}^{(j)}$), can indeed assign a reward also to states that are not reachable from any initial state of the MRgP. This reward may contribute to the reward of other non-reachable states, but clearly the reward of an unreachable state $\langle s, z\rangle$ cannot contribute to the backward computation of the reward of a reachable state (which will impair the algorithm correctness), since this will imply that $\langle s, z\rangle$ is reachable. □

5.3. Reduction based on non-useful z-states

The same optimization proposed in Section 4.1 for the $\mathcal{M}\times\mathcal{Z}$ construction can be applied to each $\mathcal{M}\times\mathcal{Z}_j$ process used by the OTF technique. For the forward case the $\mathcal{M}\times\mathcal{Z}_j$ construction is modified to a $(\mathcal{M}\times\mathcal{Z}_j)_{Red}$ one, as was done for the $\mathcal{M}\times\mathcal{Z}$ construction, to account for tagged z-states as per Equation 6. This implies that all MRgP states $\langle s, z\rangle$ in which $tag(z) = NK$ are mapped to $\perp$ through the *closure* and *fin* functions.

Example 16 (Modified execution of Algorithm 3.)**.** *When the optimization is in place only the first two components of Figure 8, upper part, are built and solved. Indeed the second component starts with a non-null probability in $\langle s_4, z_3\rangle$ and this probability ends-up in $\top$ and in $\perp$, since $tag(z_4) = NK$. As a consequence the vector H_3 of the initial states of component $\mathcal{Z}_3$ is empty and the $\mathcal{M}\times\mathcal{Z}_3(H_3)$ component is not built. The same situation holds for H_4 and $\mathcal{M}\times\mathcal{Z}_4(H_4)$.*

In addition, since the initial set S_0 is defined with the *fin* function, its content may be different in Algorithm 4 when considering $(\mathcal{M}\times\mathcal{Z}_j(S_0))_{Red}$ instead of $(\mathcal{M}\times\mathcal{Z}_j(S_0))$.

Example 17 (Modified execution of Algorithm 4.)**.** *When the optimization is in place the backward OTF solution for our running example (Figure 8, bottom part) does not build $\mathcal{M}\times\mathcal{Z}_4$ since the set S_0 only includes states of z-state z_4, and $tag(z_4) = NK$, therefore S_0 reduces to $\varnothing$. Same for $\mathcal{M}\times\mathcal{Z}_3$.*

5.4. Number of generated states: forward vs backward, component vs OTF

There is one intrinsic difference between the forward and backward solution: backward provides the Sat-set of the CSL$^{\text{TA}}$ formula, while forward concentrate on satisfaction of the initial CTMC state. The MRgP construction

starts therefore from a single state for the forward approach, and from a set of potential initial states in the backward one. For the component method the solution approach, whether forward or backward, has no impact on the number of generated states, as the state space construction is always forward and, indeed, when the set of potential initial states reduces to the initial state of the forward case, the two Algorithms 1 and 2 generate the same number of states. A similar consideration applies when the two algorithms are modified to account for non useful z-states based on the $\mathcal{M}\times\mathcal{Z}_{Red}$ construction.

OTF forward also builds the state space forward (as for the forward and backward component method): the fact that components are generated one at a time reduces the amount of total required memory, but does not reduce the number of visited states. OTF backward is instead very different: components are generated backward, based, for each component, on a set of potential initial states that are identified as all states that satisfy the Λ condition. If some of these potential states reach $\top$ but are not reachable from any initial state of the MRgP, then the difference in terms of number of generated states may be significant, as we shall see in one example of Section 7.

6. Reasoning about components: a modified Component Method and a modified OTF technique

Although any acyclic set of components ensures that $\mathbf{P}$ is in RNF as per Equation (1), the work in [17] provides evidence that the choice of the components can heavily influence the Component Method performance. Following [30] we consider three component classes, based on the cost of computing the component outgoing probability (component solution for short). A components is of class C_E if no state of the component enables a general event: the component is a CTMC and the computation of the outgoing probabilities amounts to computing the steady-state solution of the frontier states. A component is of class C_{g_k} if the only general event is g_k and the firing of the general event or its preemption leads to a state of another component: the computation of the outgoing probabilities was shown to be reducible to a transient solution at time δ_k of a CTMC generated from the component. In all other cases the component is of class C_M, and the computation of the outgoing probabilities requires the (matrix-free) solution of a MRgP.

Example 18 (**Component classification and solution costs**). *The MRgP of Figure 3(D) has 4 components. Component $\mathcal{S}_1$ is of class C_M and the probability of reaching the frontier states (either $\perp$ or the state in $\mathcal{S}_2$) is computed through a (matrix-free) steady-state solution of MRgP $\mathcal{R}_1$. $\mathcal{S}_2$ is of class C_{g_1}, since all blue and red arcs (clock boundaries and CTMC moves matched by a DTA edge with a clock reset) lead to states out of $\mathcal{S}_2$. The outgoing probability computation corresponds to the solution at time α of the component (which is a CTMC transient solution). $\mathcal{S}_3$ is of class C_{g_2} and the outgoing probability computation corresponds to the solution at time $\beta-\alpha$ of the component (which is again a CTMC transient solution). $\mathcal{S}_4$ is of class C_E, since there is no clock boundary, and its outgoing probability computation requires a CTMC steady state solution.*

The experiments in [17] suggest that working with the smallest components or with very large components may lead to inefficiency, and therefore the paper defines an optimality criteria for component aggregation, recalled in the following definitions. The paper identifies an heuristic for the construction of an optimal partition, while an optimal solution based on linear integer programming is given in [30].

Definition 20 (Uniform component). *A MRgP component $\mathcal{S}_i$ of class C (C being either C_E, C_M or C_{g_k}, $1 \leq k < m$) is* uniform *iff the set $\mathcal{S}_i$ is not decomposable into an acyclic group of sub-components of classes different from C.*

Definition 21 (MRgP valid partition). *A set of components of a MRgP is a* valid partition *iff (1) the set of components forms a partition of the state space; (2) the components are in acyclic relation; and (3) each component is uniform.*

Acyclicity ensures that the partition leads to matrices in RNF form, that can be used for the Component Method. Component uniformity ensures convenience, i.e. aggregation does not change the complexity of the required solution.

Definition 22 (MRgP component optimization problem). *The MRgP component optimization problem consists in finding a valid partition of the MRgP with the smallest cardinality.*

Example 19 (**Optimal partitioning**). *Figure 9 illustrates four examples of component choice for the same MRgP. The enabled general events are indicated on top, and the three different transition types are depicted with different arrow*

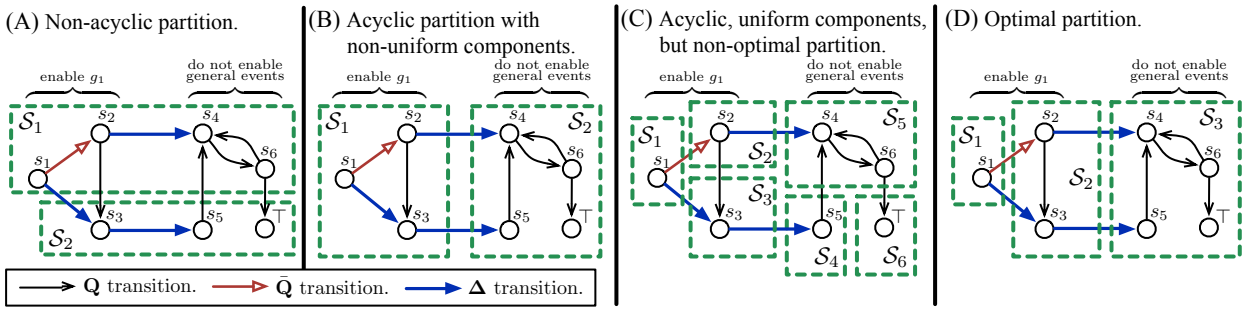


Figure 9: Different component identification for the same MRgP.

styles. Components are shown as dotted rectangles of states, and have a $\mathcal{S}_i$ label. Case (A) shows a MRgP partition that is not acyclic, since there is a transition from component $\mathcal{S}_1$ to component $\mathcal{S}_2$ ($s_2 \to s_3$), and a transition from $\mathcal{S}_2$ to $\mathcal{S}_1$ ($s_5 \to s_4$). This partition is not suitable for the Component Method. Case (B) shows an acyclic MRgP partition, suitable for the Component Method. $\mathcal{S}_1$ is of class C_M, since preemption of g_1 ($\bar{\mathbf{Q}}$ transition) and firing of g_1 ($\mathbf{\Delta}$ transition) lead to states inside the partition. $\mathcal{S}_1$ could be further decomposed into an acyclic set of components, (made by states $\{s_1\}$ and $\{s_2, s_3\}$) of class C_{g_1}, meaning that $\mathcal{S}_1$ is non-uniform. In case (C) the considered components are the strongly-connected-components of the MRgP graph. The partition is acyclic and uniform: $\mathcal{S}_1, \mathcal{S}_2$ and $\mathcal{S}_3$ are of type C_{g_1}, while $\mathcal{S}_4$, $\mathcal{S}_5$ and $\mathcal{S}_6$ are C_E. In this case the components form a DAG, they can be solved with the cheapest solution technique, but the Component Method may require the solution of many small components, with the associated overhead. Indeed the partition is non-optimal according to Definition 22. Finally, case (D) has three components: $\mathcal{S}_1$ and $\mathcal{S}_2$ of type C_{g_1} and $\mathcal{S}_3$ of type C_E. Note that if $\mathcal{S}_1$ and $\mathcal{S}_2$ are merged in a single component, this component will be of C_M type (due to the transitions out of s_1). Partition (D) depicts an optimal MRgP partition.

The original definition of the three component classes given in [30](Sec.2), can be simplified when the MRgP is a $\mathcal{M}\times\mathcal{A}$ or $\mathcal{M}\times\mathcal{Z}$, based on the observation that clock resets always lead to states of $\mathcal{S}_{g_1}$, and that the firing of g_k without clock reset enables g_{k+1}.

Definition 23. *A component $\mathcal{S}_j$ is classified in exactly one of the following $m+2$ classes:*

[Class C_E] *If* $\Gamma(\mathcal{S}_j) = \{E\}$.
[Class C_{g_k}] $1 < k \leq m$, *if* $\Gamma(\mathcal{S}_j) = \{g_k\}$,
[Class C_{g_1}] *If* $\Gamma(\mathcal{S}_j) = \{g_1\} \wedge \bar{\mathbf{Q}}_i \cdot \mathbf{I}_i = \mathbf{0}$. *(the single general event g_1 is enabled and $\bar{\mathbf{Q}}$ transitions exit from $\mathcal{S}_j$).*
[Class C_M] *Otherwise*

Algorithms 1 and 2 can be modified to work with an optimal set of components, that is to say components computed by the MRgP component optimization problem of Def. 21. Computing an optimal partition is not possible in OTF, since the full state space is not available, but, again, we can observe that the structure of the $\mathcal{M}\times\mathcal{Z}$ MRP is strongly influenced by the structure of $\mathcal{Z}$ and we can therefore envision to define a classification for the components of the region graph, based on what could be the complexity of the resulting MRgP components $\mathcal{R}_j = \mathcal{M}\times\mathcal{Z}_j$.

Definition 24. *A component Z_j of the region graph $\mathcal{Z}$ is classified in exactly one of the following $m+2$ classes:*

[Class D_E] *iff,* $\forall\langle l, c\rangle \in Z_j$, $c = [\delta_m, \infty)$.
[Class D_{g_k}] *with* $1 < k \leq m$, *iff,* $\forall\langle l, c\rangle \in Z_j, c = [\delta_{k-1}, \delta_k)$.
[Class D_{g_1}] *iff* $\forall\langle l, c\rangle \in Z_j : c = [0, \delta_1)$ *and does not exists an edge* $(l, c) \to_{I_x} (l', c)$ *and does not exists a path* $(l, c) \to_e (l, next(c)) \overset{*}{\to}_{B_\emptyset} (l', next(c)) \to_{B_x} (l'', c)$
[Class D_M] *otherwise.*

The definition of D_{g_1} is slightly complex (as for the C_{g_1} case), since it is necessary to exclude the presence in Z_j of a reset edge in the component, either by an inner edge or by a time-elapse edge followed by a path of boundary edges that goes back to the same Z_j component.

Example 20 (Components of $\mathcal{Z}$)**.** *The dotted rectangles in Figure 6(A) identify an acyclic set of components of $\mathcal{Z}$. Component $Z_1 = \{z_0, z_1, z_2\}$ is of class D_M, $Z_2 = \{z_3\}$ is of class D_{g_1}, $Z_3 = \{z_4\}$ is of class D_{g_2} and $Z_4 = \{z_5\}$ is of class D_E. Final z-states (like z_6) can be considered as a separate component since they are absorbing. Note that $\{z_0, z_1, z_2\}$ have to be in the same component to ensure acyclicity.*

The definition of uniform, valid, and optimal partition can be extended to the $\mathcal{Z}$ components.

Definition 25 (Uniform region graph component)**.** *A region graph component Z_j of class D (with D being either D_E, D_M, D_{g_1} or D_{g_k}, $k > 1$) is* uniform *iff it is not decomposable into an acyclic group of sub-components of classes different from D.*

Definition 26 (Region graph valid partition)**.** *A partition $\{Z_j\}_{1 \le j \le J}$ of $\mathcal{Z}$ is a* valid partition *iff (1) the J components are in an acyclic relation; and (2) each component Z_j is uniform.*

Definition 27 (Region graph component optimization problem)**.** *The region graph component optimization problem consists in finding a valid partition $\{Z_j\}$ of the set of z-states of $\mathcal{Z}$ with the smallest cardinality.*

The above problem is solved, using the technique presented in [30], as in the case of the MRgP component optimization problem of Definition 22.

Example 21 (Optimal components of $\mathcal{Z}$)**.** *The components of $\mathcal{Z}$ identified by the dotted rectangles in Figure 6(A) constitute an optimal set of components of $\mathcal{Z}$. As observed before $\{z_0, z_1, z_2\}$ have to be in the same component to ensure acyclicity, and all the other z-states go into separate partition elements to ensure uniformity.*

Algorithms 3 and 4 can then be modified to work with an optimal set of components of $\mathcal{Z}$, according to the definition above. Note that the size of $\mathcal{Z}$ is typically much smaller than that of $\mathcal{M}\times\mathcal{Z}$ and therefore the computation of an optimal set of components is usually feasible (for example by reducing the optimization to the solution of the ILP problem presented in [30] for MRgP), while for $\mathcal{M}\times\mathcal{A}$ in most cases only a sub-optimal partition can be found, based on the heuristic for MRgP defined in [17]. A natural question then arises: the components $\{\mathcal{M}\times\mathcal{Z}_j\}_{1\le j\le J}$, generated from an optimal set of components $\{\mathcal{Z}_j\}_{1\le j\le J}$ are an optimal set of component for the MRgP $\mathcal{M}\times\mathcal{Z}$? As we shall see this is not true, and therefore we propose next a modified OTF technique.

6.1. A modified OTF solution

Let us consider the three examples of Figure 10: in the first column there is a region graph $\mathcal{Z}$, with the optimal set of Z_j components identified by dotted lines, while the CTMC $\mathcal{M}$ is depicted at the top of the figure. In the second and third columns there are the MRgP $\mathcal{M}\times\mathcal{Z}$ built by the OTF and by the Component Method, respectively. We have chosen the example so that the $\mathcal{M}\times\mathcal{Z}$ states are the same for the two techniques, but the components considered, identified again by the dotted line, are not the same, either in number or in type. The three region graphs have two z-states z_0 and z_1, both in the first clock region $[0, \alpha)$, and they differ only in the edges connecting z_0 and z_1.
Case A: the region graph has a single component Z_1 of type D_M, since there is a reset edge from z_1 to z_0. Consequently the OTF algorithm builds a single MRgP component, as shown in the second column, which is of type C_{g_1}, since the reset edge $z_1 \rightarrow_{I_x} z_0$, accepting action b, is never triggered by the CTMC. Same construction is performed by the Component Method. Case A shows that the region graph component Z_1 of class D_M can produce a MRgP component which is not necessarily of class C_M, which suggest that, after a $\mathcal{M}\times\mathcal{Z}_j$ component generation, it is necessary to assess its class to choose the simplest applicable solution technique.
Case B: the region graph has two components Z_1 and Z_2, both of type D_{g_1}. Since there are two components, OTF builds first thecomponent $\mathcal{M}\times\mathcal{Z}_1$, solves it, and then it builds $\mathcal{M}\times\mathcal{Z}_2$ and solves it. Both components are of type C_{g_1}. The Component Method builds the components after having built the full state space, which allows to identify an optimal partition with a single component of type C_{g_1}, as illustrated in the third column. In this second case, OTF fails to identify the optimal aggregation, because, although the reset edge $z_0 \rightarrow_{I_x} z_1$ is never triggered by $\mathcal{M}$, its presence in $\mathcal{Z}$ separates Z_1 from Z_2 in the region graph partition. This shows that an optimal region graph partition does not necessarily result in a optimal $\mathcal{M}\times\mathcal{Z}$ partition.
Case C: the region graph has a single component Z_1 of type D_M. Since there is a single component, OTF builds a single MRgP component $\mathcal{M}\times\mathcal{Z}_1$ and, due to the clock reset associated to the a and b labelled edges, it classifies it

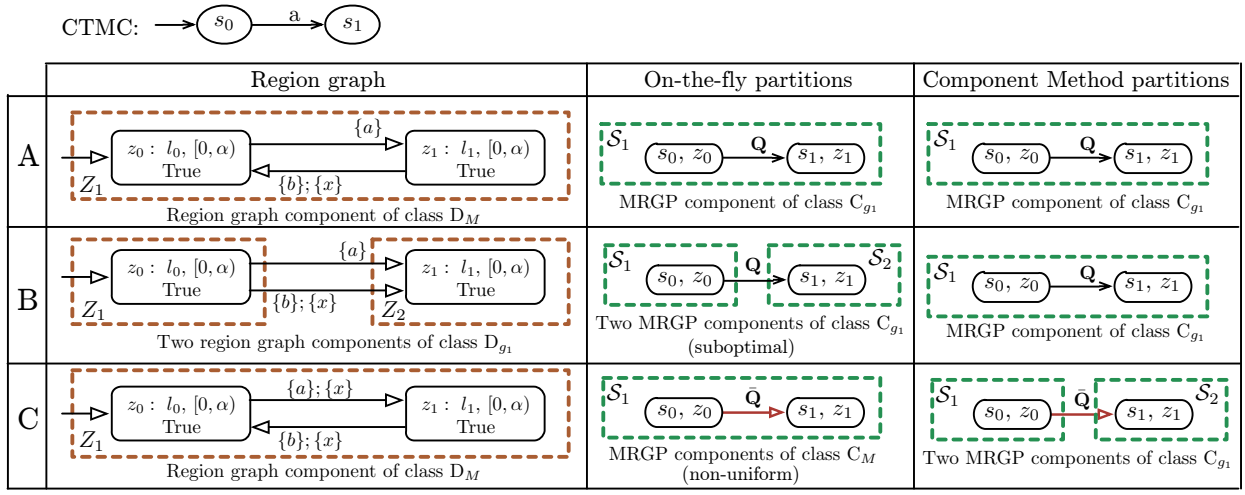


Figure 10: Relationships between $\{\mathcal{M}\times\mathcal{Z}_j\}_{1\le j\le J}$ and the MRgP components $\{\mathcal{R}_i\}_{1\le i\le I}$ of $\mathcal{R}=\mathcal{M}\times\mathcal{Z}$.

as C_M. The component based technique is instead able to recognize that in the MRgP there is no cycle, since the b labeled edge of $\mathcal{Z}$ is never used in the $\mathcal{M}\times\mathcal{Z}$ construction ($\mathcal{M}$ has no transitions labelled b) and identifies two components of class (C_{g_1}).

The three cases above represents rather different situations: case A tells us that OTF should classifies ex-novo each component, without relying on the classification of the region graph component class. Case B shows that, due to the local view of region graph partition, OTF algorithms may miss optimizations based on components aggregation. Case C shows that OTF again may miss optimizations based on components dis-aggregation, due to loops existing in $\mathcal{Z}$ but not in $\mathcal{M}\times\mathcal{Z}$. Clearly the inefficiency incurred in case B cannot be easily corrected while the inefficiency of case C can be avoided. This inefficiency stems from the presence of a *non-uniform* OTF component of class C_M, and, by applying the Component Method on each OTF component $\mathcal{M}\times\mathcal{Z}_j$ of class C_M allows us to derive a sub-partition of the component, which is uniform by definition.

Algorithm 5 shows the modified OTF for the forward case: in bold is the addition with respect to Algorithm 3. Note that the component is already built, therefore the Component Method is applicable.

Algorithm 5 Pseudocode of the refined forward on-the-fly model checking method.

function MODELCHECK-OTF$_{\text{FWD}}$+COMP($\{Z_j\}$ components of $\mathcal{Z}$, s_0 : initial state)
  $\boldsymbol{\pi}^{(0)} : S \times Z \to \mathbb{R}$ ▷ sparse vector of state probabilities
  $\boldsymbol{\pi}^{(0)}[\langle s_0, l_0, 0\rangle] \leftarrow 1$
  Identify the set $\{Z_j\}$ of optimal components of $\mathcal{Z}$, taken in topological order.
  for each subset Z_j in $\mathcal{Z}$ **do**
    H_j = all the tuples $\langle s, z\rangle$ with $\boldsymbol{\pi}^{(j-1)}[\langle s, z\rangle] \neq 0 \wedge z \in Z_j$
    Let $\mathbf{I}_{H_j}$ be the filtering matrix of H_j
    Construct $\mathcal{M}\times\mathcal{Z}_j$ considering H_j as initial states and $\mathbf{I}_{H_j}\cdot\boldsymbol{\pi}^{(j-1)}$ as initial distribution.
    Compute the probability $\boldsymbol{\mu}_i$ outgoing $\mathcal{M}\times\mathcal{Z}_j$ and
      reaching states $\langle s, z\rangle \in frontier(\mathcal{M}\times\mathcal{Z}_j)$, **using Algorithm 1 if $\mathcal{M}\times\mathcal{Z}_j$ is of type C_M**
    $\boldsymbol{\pi}^{(j)} \leftarrow (\mathbf{I} - \mathbf{I}_{H_j})\cdot\boldsymbol{\pi}^{(j-1)} + \boldsymbol{\mu}_i$
  end for
  return $\boldsymbol{\pi}^{(K)}[\top]$
end function

Algorithm 6 shows the modified OTF for the backward case, where, again, the new part is reported in bold. In this case the algorithm to use is the backward Component Method.

Algorithm 6 Pseudocode of the refined backward on-the-fly model checking method.

function MODELCHECK-OTF$_{\text{BWD}}$+COMP($\{Z_j\}$ components of $\mathcal{Z}$)
  $\mathbf{r}^{(j)} : S \times Z \to \mathbb{R}$ ▷ sparse vector of per-state acceptance probabilities
  $\mathbf{r}^{(j)}[\top] \leftarrow 1$
  for each $\{Z_j\}$ of Z in backward topological order **do**
    Construct the synchronized process $\mathcal{M}\times\mathcal{Z}_j$, assuming as initial states all $\langle s, z\rangle : s \models \Lambda(z)$.
    Compute the reward vector $\boldsymbol{\xi}^{(j)}$ for $\mathcal{M}\times\mathcal{Z}_j$ states, starting from
      the frontier rewards $\mathbf{r}^{(j)}$, **using Algorithm 2 if** $\mathcal{M}\times\mathcal{Z}_j$ **is of type** $\mathbf{C}_M$
    $\mathbf{r}^{(j+1)} \leftarrow \boldsymbol{\xi}^{(j)} + \mathbf{r}^{(j)}$
  end for
  $\mathbf{r}^{(K)}[s]$ contains $Prob\{s \text{ reaches } \top\}$, for all $s \in \mathcal{S}$.
  return $\mathbf{r}^{(K)}$
end function

7. Numerical results and model checking tool

To test the impact of the various model checking methods described in this paper, we have developed an extended version of the MC4CSL$^{\text{TA}}$ tool [14]. The tool is part of the GreatSPN framework [31], and is integrated in its graphical user interface. CTMCs are constructed from Petri net models (specifically from GSPN [32]), and DTAs are drawn directly in the GUI. DTAs are defined in a parametric manner: when the user requires to model check a GSPN $\mathcal{N}$ for a given DTA $\mathcal{A}$, the DTA is instantiated by associating a specific marking expression of $\mathcal{N}$ to each atomic proposition of $\mathcal{A}$, and a transition name of $\mathcal{N}$ to each action name of $\mathcal{A}$. From the GUI is it also possible to play a joint "token-game": starting from an initial marking the user interactively selects a transition of the GSPN to be fired and the interface displays the new marking (on the GSPN) and the edge that accept that firing (in the DTA), if any.

A virtual machine with the tool pre-installed and all the model data needed to reproduce the results can be found at `http://www.di.unito.it/~greatspn/VBox/GreatSPN-8.0.ova`, as a VirtualBox image. Instructions are found in the Desktop/CSLTA directory. The data presented in this paper have been computed on the Occam machine [33], having 128GB of free memory, fixing a time limit of 1 hour for each algorithm run. The tool implements the model checking algorithms of table of Figure 1.

This section is meant to experimentally answer the following questions:

Q1. *z-states tagging.* What is the impact of building the MRgP based on the tagged region graph $\mathcal{M}\times\mathcal{Z}_{Red}$, instead of the standard $\mathcal{M}\times\mathcal{A}$ construction initially proposed for CSL$^{\text{TA}}$?

Q2. *Forward vs Backward.* What differences may arise in performing forward or backward model checking of CSL$^{\text{TA}}$? Is the state space different? Are the considered components different?

Q3. *Component vs Full.* Is the Component Method (Comp) beneficial over the standard MRgP solution (Full) in all cases?

Q4. *OTF.* Does the On-The-Fly (OTF) method reduce the treated state space? Is it advantageous in general?

Q5. *OTF+Comp* Is the recursive refinement of the OTF method (OTF+Comp) of Section 6.1 useful?

Q6. *OTF on CSL.* Is the performance of a CSL$^{\text{TA}}$ model checker based on OTF comparable to that of a state-of-the-art CSL model checker on CSL formulas?

The tests consists of two models and various CSL$^{\text{TA}}$ DTAs to asses the first 5 questions, and a model and two different CSL Until queries for comparison with the CSL model checker of Prism [8] and Storm [11]. Each DTA corresponds to a CSL$^{\text{TA}}$ property and the results of its model checking are reported in the same format: an upper table that contains the info on the generated states, components, and their types and a middle and lower tables that report, respectively, the solution time and the memory occupation for all algorithms presented in this paper, using the MC4CSL$^{\text{TA}}$ model checker. All data is reported for both forward and backward. Solution times include the time to verify the CSL$^{\text{TA}}$ properties, excluding the time to load the CTMC and the DTA from the external files, and including the solution of the MRgPs. The reported memory is the whole memory allocated by the process, in MBytes. Partitions are computed using the method described in [17].

			Forward										Backward									
			$\mathcal{M}\times\mathcal{A}$				$\mathcal{M}\times\mathcal{Z}_{Red}$				OTF				$\mathcal{M}\times\mathcal{A}$				$\mathcal{M}\times\mathcal{Z}_{Red}$				OTF			
N	$\lvert\mathcal{M}\rvert$	$\lvert Z\rvert$	states	components			states	components			max size	components			states	components			states	components			max size	components		
				C_E	C_g	C_M		C_E	C_g	C_M		C_E	C_g	C_M		C_E	C_g	C_M		C_E	C_g	C_M		C_E	C_g	C_M
10	42636	15	222682	1	3	0	75384	0	3	0	55672	0	3	0	223122	1	3	0	75494	0	3	0	55794	0	3	0
20	519211	15	2685182	1	3	0	899734	0	3	0	671297	0	3	0	2686022	1	3	0	899944	0	3	0	671519	0	3	0
30	2388736	15	12303482	1	3	0	4107034	0	3	0	3075872	0	3	0	12304722	1	3	0	4107344	0	3	0	3076194	0	3	0

Solution time (in seconds) for the 10 methodologies:

N	$\text{Full}^{\mathcal{A}}_{\text{fwd}}$	$\text{Comp}^{\mathcal{A}}_{\text{fwd}}$	$\text{Full}^{\mathcal{Z}}_{\text{fwd}}$	$\text{Comp}^{\mathcal{Z}}_{\text{fwd}}$	OTF_{fwd}	$\text{Full}^{\mathcal{A}}_{\text{bwd}}$	$\text{Comp}^{\mathcal{A}}_{\text{bwd}}$	$\text{Full}^{\mathcal{Z}}_{\text{bwd}}$	$\text{Comp}^{\mathcal{Z}}_{\text{bwd}}$	OTF_{bwd}
10	8.13	3.69	3.08	1.57	0.94	8.99	4.18	4.15	1.82	1.11
20	119.22	49.10	47.27	24.63	13.55	70.70	50.70	48.91	24.32	15.48
30	468.71	188.64	159.43	78.76	46.94	276.48	195.29	120.03	83.17	46.63

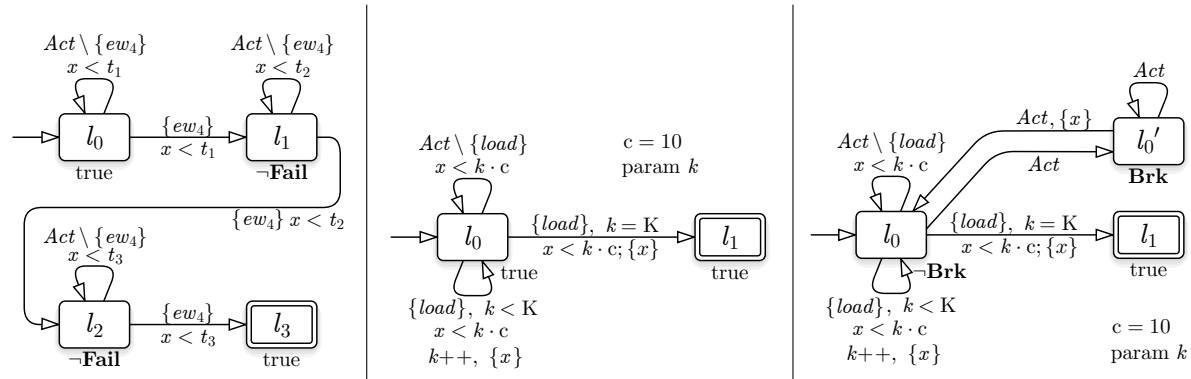
Memory occupation (in MBytes):

N	$\text{Full}^{\mathcal{A}}_{\text{fwd}}$	$\text{Comp}^{\mathcal{A}}_{\text{fwd}}$	$\text{Full}^{\mathcal{Z}}_{\text{fwd}}$	$\text{Comp}^{\mathcal{Z}}_{\text{fwd}}$	OTF_{fwd}	$\text{Full}^{\mathcal{A}}_{\text{bwd}}$	$\text{Comp}^{\mathcal{A}}_{\text{bwd}}$	$\text{Full}^{\mathcal{Z}}_{\text{bwd}}$	$\text{Comp}^{\mathcal{Z}}_{\text{bwd}}$	OTF_{bwd}
10	198.18	159.87	58.75	73.11	47.16	182.45	119.46	59.12	56.80	50.14
20	2353.62	1934.27	1040.09	930.75	563.28	2465.85	1600.22	872.36	837.68	601.64
30	10819.30	8841.74	4806.00	4333.68	2632.25	11357.54	7322.23	4974.15	4269.90	2794.29

Table 1: Performance result of the DTA 11(a) on the FMS model.

7.1. Flexible Manufacturing System model

The first test is on a *Flexible Manufacturing System* (FMS) model, taken from [34]. The model represents a system where N pallets are treated in a sequence of four machines, that can break down. The model is parametric in the number N of circulating pallets, and it is ergodic. The Petri net model can be found in Appendix A and on the provided virtual machine. Figure 11 shows three DTAs that define three CSL$^{\text{TA}}$ path properties for the FMS model.



(a) First test DTA that models three *ew4* time bounded events.

(b) Second test DTA

(c) Third test DTA

Figure 11: DTA used for the performance test on the FMS model.

Results for FMS, first DTA. DTA(a) accepts the set of paths where the event *ew4* (completion on the M_4 machine) happens three times in a row without encountering a failure, and each occurrence happens before a given time bound (t_1, t_2 and t_3) elapses. The clock is never reset. Its model checking will therefore compute the probability that Machine M_4 terminates three pieces in a row in within the three given time bounds.

Table 1 presents the model checking results for different values of N. The structure of this table is common for the whole set of results. The upper table reports, in the order: the parameter on which the test is performed, the number $|\mathcal{M}|$ of CTMC states, the number of clock regions $|Z|$, information for the $\mathcal{M}\times\mathcal{A}$ and $\mathcal{M}\times\mathcal{Z}_{Red}$ construction (state, number and types of components), and, for OTF, the size of the largest component and the number and types of components. All techniques are reported for the two cases: forward (columns on the left) and backward (columns on the right). The central and the lower tables report the solution time and the memory occupation for MC4CSL$^{\text{TA}}$ tool,

for the 5×2 (forward and backward) solution techniques summarized in Figure 1. For all algorithms based on the region graph (OTF and algorithms that use $\mathcal{M}\times\mathcal{Z}$) , the optimization based on the tagging of the z-states is in place. From the upper table in Table 1 we can observe that the region graph has 15 states, $|\mathcal{M}|$, the number of states in the CTMC being verified, goes from a few thousands to more than two millions, and that the MRgP built on the $\mathcal{M}\times\mathcal{A}$ is about 5 times as large as $|\mathcal{M}|$. From this table we can answer the 5 questions as follows.

Q1 - *z-states tagging* can be observed by comparing the states of the columns labelled $\mathcal{M}\times\mathcal{A}$ against those labelled $\mathcal{M}\times\mathcal{Z}$ and the solution times and memory occupation of the columns marked with a superscript $\mathcal{A}$ with the columns of equal name but superscript $\mathcal{Z}$. z-states tagging in this model has a visible impact: $|\mathcal{M}\times\mathcal{Z}|$ (the number of states in the MRgP generated from $\mathcal{M}\times\mathcal{Z}$) is one third of $|\mathcal{M}\times\mathcal{A}|$: in $\mathcal{M}\times\mathcal{A}$ there are many states that do not lead to $\top$, since it is possible to remain in locations l_0, l_1 and l_2 beyond the time limits t_1, t_2 and t_3 specified on the DTA edges. z-states tagging allows us to reduce also the number of components: $\mathcal{M}\times\mathcal{A}$ has 4 components (one C_E and three C_g, each one of size 3 075 872), while $\mathcal{M}\times\mathcal{Z}$ has only three components of size 3 075 872, 687 444 and 343 718. Not surprisingly, the reduction in the number of components and in the number of states per component is matched by a reduction in solution time both with forward (compare the column marked $\mathrm{Full}^{\mathcal{A}}_{\mathrm{fwd}}$ with that marked $\mathrm{Full}^{\mathcal{Z}}_{\mathrm{fwd}}$), and with backward techniques (columns $\mathrm{Full}^{\mathcal{A}}_{\mathrm{bwd}}$ and $\mathrm{Full}^{\mathcal{Z}}_{\mathrm{bwd}}$); similar reduction is obtained for the Comp methods.

Q2 - *Forward vs Backward.* The difference is rather limited: there is a small difference between the state space generated from the single, given initial state, as does the forward technique, and the state space generated by considering all possible feasible initial states (as does the backward technique). The difference is small for all three state space generation techniques ($\mathcal{M}\times\mathcal{A}$, $\mathcal{M}\times\mathcal{Z}$, and OTF) and it does not significantly influence the solution time and the memory occupation. Observe that $\mathrm{Full}^{\mathcal{A}}_{\mathrm{fwd}}$ has larger solution time than $\mathrm{Full}^{\mathcal{A}}_{\mathrm{bwd}}$ (almost the double for $N = 30$), despite the fact that a single iteration in backward is more expensive than a forward one. The better performance of backward is due to the fact that the backward MRgP solution requires less iterations than the forward case, for all techniques.

Q3 - *Comp vs Full.* Comp performs better than Full in both the forwards and backwards techniques, and independently of the state space construction (superscript $\mathcal{A}$ or $\mathcal{Z}$). This means that the time spent for building the components is negligible with respect to the advantage of solving four (or three) smaller components instead of a single large one as in Full. It may appear somehow counter-intuitive that Comp performs better than Full also in terms of memory occupation, but this due to the fact that working with the full MRgP requires larger solution vectors than working with components. Moreover the effect is amplified by the use of the GMRES [35] algorithm for the computation of the steady-state solution, for which we retain 30 vectors for the Krylov subspace.

Q4 - *OTF.* Both $\mathrm{OTF}_{\mathrm{fwd}}$ and $\mathrm{OTF}_{\mathrm{bwd}}$ build the same number of components as the corresponding Comp methods, and of the same type. As reported above a single component counts for 75% of the full state space and this is certainly not a very good condition for OTF, that works better when there are components of similar size: nevertheless OTF performs significantly better than Comp both in memory (as expected) and in time. Since Comp and OTF, for this model, solve the same components, this reduction in time is related to the component identification and construction: OTF computes the components of the region graph $\mathcal{Z}$, which has only 15 states, for any value of N, while Comp computes the components of the whole state space (which for $N = 30$ has more than 4 millions states).

Q5 - *OTF + Comp.* does not apply since there is no C_M component (there is no clock reset in the DTA).

Results for FMS, second DTA. The DTA of Figure 11(b) accepts all executions with K contiguous loads on machine M_1 that respect a given inter-event time constraint. Indeed DTA (b) counts K occurrences of the *load* event (a new pallet enters the M_1 machine) and at the k-th occurrence, the *load* event must happen before a time bound $10 \times k$, with the clock being reset at every event occurrence. When a load event occurs and the counter k is less than K, the DTA stays in location l_0 and increments the counter (k++). If the counter reaches K and a new *load* event is observed, the DTA moves to the final location l_1. The DTA is here represented in a compact manner: in the tool increasing K linearly increases the number of locations, and this example has been chosen to study the behaviour of the algorithms for an increasing number of z-states and components.

Table 2 reports the results for this DTA for different event counts K. The table follows the same structure as the previous example, except for the missing column of the CTMC states which is constant in all runs (180336 states). These results allows us to investigate the questions Q1, Q2, Q3, and Q4.

Q1 - *z-states tagging*. The impact in percentage is more limited than in the previous example, nevertheless the absolute difference $|\mathcal{M}\times\mathcal{A}| - |\mathcal{M}\times\mathcal{Z}|$ can grow rather big (more than 16 millions states for $K = 25$). Note the

Data for N=15, CTMC states =180336, build time: 12s

		Forward												**Backward**											
		$\mathcal{M}\times\mathcal{A}$				$\mathcal{M}\times\mathcal{Z}_{Red}$				OTF				$\mathcal{M}\times\mathcal{A}$				$\mathcal{M}\times\mathcal{Z}_{Red}$				OTF			
K	|Z|	states	components			states	components			max size	components			states	components			states	components			max size	components		
			C_E	C_g	C_M		C_E	C_g	C_M		C_E	C_g	C_M		C_E	C_g	C_M		C_E	C_g	C_M		C_E	C_g	C_M
5	30	21920	1	15	0	16316	0	15	0	2227	0	15	0	5410082	1	15	0	2705042	0	15	0	180338	0	15	0
10	110	857518	1	55	0	673455	0	55	0	29637	0	55	0	19836962	1	55	0	9918482	0	55	0	180338	0	55	0
15	240	8144546	1	120	0	6521294	0	120	0	140472	0	120	0	43280642	1	120	0	21640322	0	120	0	180338	0	120	0
20	420	29624786	1	210	0	22751374	0	210	0	180338	0	210	0	-	-	-	-	37870562	0	210	0	180338	0	210	0
25	650	60121986	1	325	0	43490014	0	325	0	180338	0	325	0	-	-	-	-	58609202	0	325	0	180338	0	325	0

Solution time (in seconds) for the 10 methodologies:

K	$\text{Full}^{\mathcal{A}}_{\text{fwd}}$	$\text{Comp}^{\mathcal{A}}_{\text{fwd}}$	$\text{Full}^{\mathcal{Z}}_{\text{fwd}}$	$\text{Comp}^{\mathcal{Z}}_{\text{fwd}}$	OTF_{fwd}	$\text{Full}^{\mathcal{A}}_{\text{bwd}}$	$\text{Comp}^{\mathcal{A}}_{\text{bwd}}$	$\text{Full}^{\mathcal{Z}}_{\text{bwd}}$	$\text{Comp}^{\mathcal{Z}}_{\text{bwd}}$	OTF_{bwd}
5	1.42	0.40	1.19	0.34	0.28	1616.45	103.00	734.60	62.15	49.11
10	388.98	20.26	400.95	16.52	10.03	-	572.70	-	351.39	268.88
15	-	173.10	-	136.06	81.98	-	1109.98	-	596.26	454.54
20	-	732.39	-	597.03	332.31	-	-	-	1161.76	784.28
25	-	1457.20	-	1169.44	565.82	-	-	-	1807.00	1120.04

Memory occupation (in MBytes):

K	$\text{Full}^{\mathcal{A}}_{\text{fwd}}$	$\text{Comp}^{\mathcal{A}}_{\text{fwd}}$	$\text{Full}^{\mathcal{Z}}_{\text{fwd}}$	$\text{Comp}^{\mathcal{Z}}_{\text{fwd}}$	OTF_{fwd}	$\text{Full}^{\mathcal{A}}_{\text{bwd}}$	$\text{Comp}^{\mathcal{A}}_{\text{bwd}}$	$\text{Full}^{\mathcal{Z}}_{\text{bwd}}$	$\text{Comp}^{\mathcal{Z}}_{\text{bwd}}$	OTF_{bwd}
5	35.71	29.09	35.16	28.38	24.50	5542.26	3482.73	3293.44	2357.66	355.88
10	937.61	648.30	786.16	574.22	39.90	-	12868.47	-	8621.42	806.00
15	-	6379.68	-	5704.39	150.43	-	28184.15	-	18796.57	1537.07
20	-	22723.17	-	19802.47	221.62	-	-	-	32895.46	2554.07
25	-	44993.45	-	37818.35	229.37	-	-	-	50909.42	3855.65

Table 2: Performance result of the DTA 11(b) on the FMS model.

impact on the size of the solvable models for the backward case: Comp based on $\mathcal{A}$ can solve up to $K = 5$, while Comp based on $\mathcal{Z}$ can solve up to $K = 25$.

Q2 - *Forward vs Backward.* The difference in this case is significant, especially for small values of K. For instance with $K = 5$, $|\mathcal{M}\times\mathcal{A}|$ built by the backward techniques is more than 200 times larger than that built by the forward techniques. When all CTMC states are considered as initial states (as in backward), the synchronized process may trigger many combinations of $\langle s, z\rangle$ pairs that are not reachable from s_0. Indeed the DTA observes sequences of *load* events, which are just a specific sequence when starting from a single state s_0, but it is close to the full cross-product of $\mathcal{S}$ and Z when considering all initial states, because most $\mathcal{S}\times Z$ combinations happen to be reachable.

Q3 - *Component vs Full.* These MRgPs have a large number of components. An analysis of the size of the components reveals that, in forward, the size of the biggest component increases with K, up to $|\mathcal{M}|$, in backward it is fixed, and equal to $|\mathcal{M}|$, while in Full it keeps growing with K. These are favourable conditions for Comp, that indeed outperforms Full both in the forwards and backwards techniques in time and space. In Comp the state space is partitioned into multiple components of C_g class (up to 325 components for the $K = 25$ case), which are more efficiently solved than the whole MRgP (as done by Full). The table shows that, for $K > 10$, only the methods that exploit components, like Comp and OTF, can model check the property in less than the imposed time limit of 1 hour.

Q4 - *OTF.* OTF is here clearly superior to both Full and Comp. In time there is a factor of about 2 between OTF and Comp (less in backward), which is possibly due to the difference between computing the components for the full state space or only for the z-states of the region graph. In memory OTF shows very good performance: in particular OTF forward has a memory consumption of less than 0.3 GB against the 37GB of Comp. In this example OTF_{fwd} is significantly more efficient in memory than OTF_{bwd}, despite the fact that they build, for large K, the same number of components, of the same size (although in reverse order). This happens because of the way the vectors $\boldsymbol{\pi}$ and $\mathbf{r}$ are managed in Algorithms 3 and 4. Vector $\boldsymbol{\pi}$ is multiplied by $(\mathbf{I} - \mathbf{H}_j)$, which removes entries at every iteration. Vector $\mathbf{r}$ instead accumulates the reward of every encountered state. This difference is visible only in this case, since each component represents a small percentage of the state space.

Results for FMS, third DTA. The DTA in Figure 11(c) is similar to that in (b), but when machine M_3 fails (condition **Brk** is $\#M_3ko \neq 0$) the DTA goes and stays in location l'_0, going back to location l_0 only when the repair is completed. After a repair the clock is reset. Table 3 reports the results of the model checking. Since the DTA may do a clock

Data for N=5, CTMC states=4361

		Forward													**Backward**										
		$\mathcal{M}\times\mathcal{A}$				$\mathcal{M}\times\mathcal{Z}_{Red}$				OTF				OTF+comp				$\mathcal{M}\times\mathcal{A}$				$\mathcal{M}\times\mathcal{Z}_{Red}$				OTF				OTF+comp			
K	|Z|	states	components			same as $\mathcal{M}\times\mathcal{A}$	components			max size	components			components			states	components			same as $\mathcal{M}\times\mathcal{A}$	components			max size	components			components		
			C_E	C_g	C_M		C_E	C_g	C_M		C_E	C_g	C_M	C_E	C_g	C_M		C_E	C_g	C_M		C_E	C_g	C_M		C_E	C_g	C_M	C_E	C_g	C_M
3	24	34894	2	6	2	34894	2	6	2	17446	0	0	3	8	9	2	52334	2	6	3	52334	2	6	3	17446	0	0	3	8	9	3
4	40	65422	2	10	3	65422	2	10	3	21807	0	0	4	11	16	3	87222	2	10	4	87222	2	10	4	21807	0	0	4	11	16	4
5	60	104672	2	15	4	104672	2	15	4	26168	0	0	5	14	25	4	130832	2	15	5	130832	2	15	5	26168	0	0	5	14	25	5

Solution time (in seconds) for the 10 methodologies:

K	$\text{Full}^{\mathcal{A}}_{\text{fwd}}$	$\text{Comp}^{\mathcal{A}}_{\text{fwd}}$	$\text{Full}^{\mathcal{Z}}_{\text{fwd}}$	$\text{Comp}^{\mathcal{Z}}_{\text{fwd}}$	OTF_{fwd}	OTF_{fwd} +comp	$\text{Full}^{\mathcal{A}}_{\text{bwd}}$	$\text{Comp}^{\mathcal{A}}_{\text{bwd}}$	$\text{Full}^{\mathcal{Z}}_{\text{bwd}}$	$\text{Comp}^{\mathcal{Z}}_{\text{bwd}}$	OTF_{bwd}	OTF_{bwd} +comp
3	14.76	17.71	14.76	17.66	18.16	17.86	82.24	60.70	79.38	60.62	57.43	56.33
4	59.65	37.42	70.90	37.45	38.16	38.77	240.54	118.45	240.27	119.58	116.62	116.17
5	145.47	66.61	147.23	68.91	72.08	71.99	455.44	197.67	408.20	213.63	212.70	197.98

Memory occupation (in MBytes):

K	$\text{Full}^{\mathcal{A}}_{\text{fwd}}$	$\text{Comp}^{\mathcal{A}}_{\text{fwd}}$	$\text{Full}^{\mathcal{Z}}_{\text{fwd}}$	$\text{Comp}^{\mathcal{Z}}_{\text{fwd}}$	OTF_{fwd}	OTF_{fwd} +comp	$\text{Full}^{\mathcal{A}}_{\text{bwd}}$	$\text{Comp}^{\mathcal{A}}_{\text{bwd}}$	$\text{Full}^{\mathcal{Z}}_{\text{bwd}}$	$\text{Comp}^{\mathcal{Z}}_{\text{bwd}}$	OTF_{bwd}	OTF_{bwd} +comp
3	38.28	28.06	38.32	28.10	29.08	32.20	62.73	38.42	59.52	38.38	27.95	32.58
4	72.72	47.59	72.79	47.59	34.26	40.43	104.24	62.24	105.69	62.24	36.66	42.52
5	113.75	74.51	113.84	74.51	40.94	47.07	165.86	91.80	168.00	91.87	43.90	50.89

Table 3: Performance result of the DTA 11(c) on the FMS model.

reset, there are components of class C_M.

Q1 is not significant, since both $\mathcal{M}\times\mathcal{A}$ and $\mathcal{M}\times\mathcal{Z}$ build the same MRgP and Q3 answer is consistent with what observed in the previous models (Comp performing better than Full).

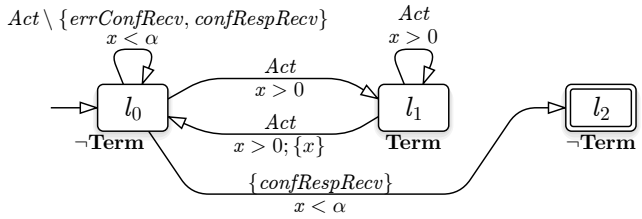
Q2 *Forward vs Backward.* The state spaces for the forwards and backwards techniques are different in size (as in the previous examples), but in this case also the number of identified components is different: $\mathcal{M}\times\mathcal{Z}$ backward has one component more that $\mathcal{M}\times\mathcal{Z}$ forward, while OTF builds the same number of components in the forwards and backwards techniques.

Q4 *OTF.* OTF builds K components of type C_M. Taking as reference $K = 5$ forward (backward is the same), OTF builds 5 C_M components, while Comp identifies 2 C_E, 15 C_g, and 4 C_M. The analysis of the structure of the components (not reported in the tables) reveals that OTF builds a single C_M component which corresponds to the union of the C_E and C_g components used by Comp: we are indeed in the situation illustrated by case C of Figure 10. Despite this poor construction of the components, the memory requirements of OTF are always better than that of Comp and Full, and the solution times are aligned with that of Comp.

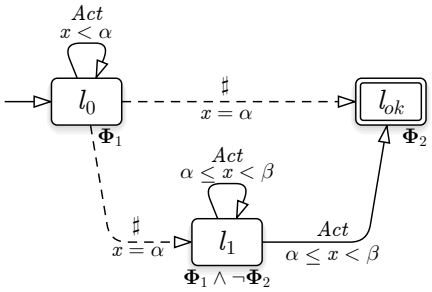
Q5 *OTF + Comp.* The presence of components of class C_M allows us to investigate whether OTF refined, in which Comp is applied to the solution of each C_M component, is able to build a better component set of components than OTF alone. Results are reported in the columns labelled "OTF + comp"). Taking as reference the forward case, for $K = 5$, OTF refined creates 14 C_E, 25 C_g, and 4 C_M components, while Comp on $\mathcal{M}\times\mathcal{Z}$ works with 2 C_E, 15 C_g and 4 C_M. This is an instance of the situation identified as Case B in Figure 10: OTF, having only a local knowledge, is not able to aggregate components that should be aggregated. Nevertheless the C_M components identified by OTF+Comp are exactly the same as those identified by $\text{Comp}^{\mathcal{Z}}_{\text{fwd}}$. The additional work to build the components of each $\mathcal{R}_i$ is balanced by the advantage of having to consider one C_M (expensive) component less, which leads to time and memory performances of OTF+Comp that are similar to those of plain OTF.

7.2. CLUE protocol model

The second test model is a Stochastic Petri net of the CLUE application protocol [36]. CLUE is an application protocol for the negotiation of a telepresence session between multiple participants. After a session establishment phase, the participants exchange messages upon a data channel in the form of XML descriptors. Each participant acts both as a Media Provider (MP) and as a Media Consumer (MC). Network errors may happen at any time, and the protocol allows at most R retry attempts, before terminating with a failure. The Petri net model, parametric in R, can be found in Appendix A and on the provided virtual machine. The parameter R is the number of retries the client (or server) is allowed to do before failing. Figure 12 shows the test DTAs used for the CLUE model. DTA(b) is actually a CSL Until property.



(a) Fourth test DTA with reset cycle.



(b) Fifth test DTA that models the CSL formula $\Phi_1 \, \mathcal{U}^{[\alpha,\beta]} \, \Phi_2$.

Figure 12: DTA used for the performance test on the CLUE protocol model.

			Forward												**Backward**											
			$\mathcal{M}\times\mathcal{A}$				$\mathcal{M}\times\mathcal{Z}_{Red}$				OTF				OTF+comp				$\mathcal{M}\times\mathcal{A}$				$\mathcal{M}\times\mathcal{Z}_{Red}$				OTF				OTF+comp		
R	$\lvert\mathcal{M}\rvert$	$\lvert Z\rvert$	states	components			same as $\mathcal{M}\times\mathcal{A}$	components			max size	components			components			states	components			same as $\mathcal{M}\times\mathcal{A}$	components			max size	components			components		
				C_E	C_g	C_M		C_E	C_g	C_M		C_E	C_g	C_M	C_E	C_g	C_M		C_E	C_g	C_M		C_E	C_g	C_M		C_E	C_g	C_M	C_E	C_g	C_M
5	58854	5	74250	1	0	1	74250	1	0	1	74250	0	0	1	1	0	1	117710	2	1	1	117710	2	1	1	117710	0	0	1	2	1	1
7	597553	5	751608	1	0	1	751608	1	0	1	751608	0	0	1	1	0	1	1195108	2	1	1	1195108	2	1	1	1195108	0	0	1	2	1	1
9	3651781	5	4629084	1	0	1	4629084	1	0	1	4629084	0	0	1	1	0	1	7303564	2	1	1	7303564	2	1	1	7303564	0	0	1	2	1	1

Solution time (in seconds) for the 10 methodologies:

R	$\text{Full}^{\mathcal{A}}_{\text{fwd}}$	$\text{Comp}^{\mathcal{A}}_{\text{fwd}}$	$\text{Full}^{\mathcal{Z}}_{\text{fwd}}$	$\text{Comp}^{\mathcal{Z}}_{\text{fwd}}$	OTF_{fwd}	OTF_{fwd} +comp	$\text{Full}^{\mathcal{A}}_{\text{bwd}}$	$\text{Comp}^{\mathcal{A}}_{\text{bwd}}$	$\text{Full}^{\mathcal{Z}}_{\text{bwd}}$	$\text{Comp}^{\mathcal{Z}}_{\text{bwd}}$	OTF_{bwd}	OTF_{bwd} +comp
5	4.78	3.94	4.84	4.46	4.50	4.00	9.15	6.38	9.47	6.41	9.42	6.37
7	86.43	73.07	88.01	66.35	82.72	60.92	186.49	102.18	188.11	91.35	181.16	86.10
9	558.09	473.82	563.96	471.08	546.18	466.51	1156.24	685.77	1150.02	693.42	1210.30	691.55

Memory occupation (in MBytes):

R	$\text{Full}^{\mathcal{A}}_{\text{fwd}}$	$\text{Comp}^{\mathcal{A}}_{\text{fwd}}$	$\text{Full}^{\mathcal{Z}}_{\text{fwd}}$	$\text{Comp}^{\mathcal{Z}}_{\text{fwd}}$	OTF_{fwd}	OTF_{fwd} +comp	$\text{Full}^{\mathcal{A}}_{\text{bwd}}$	$\text{Comp}^{\mathcal{A}}_{\text{bwd}}$	$\text{Full}^{\mathcal{Z}}_{\text{bwd}}$	$\text{Comp}^{\mathcal{Z}}_{\text{bwd}}$	OTF_{bwd}	OTF_{bwd} +comp
5	56.58	72.58	56.62	72.62	72.71	85.63	85.78	75.37	85.83	77.38	86.30	77.30
7	704.57	845.84	709.49	841.08	701.71	837.16	1070.62	994.05	1073.47	999.80	827.78	758.59
9	4363.45	5099.11	4351.79	5095.48	4312.89	5029.86	6582.77	6094.85	6577.33	6087.29	5006.41	4635.64

Table 4: Performance result of the DTA 12(a) on the CLUE model.

Results for CLUE, first DTA. DTA 12(a) accepts a path that stays in a $\neg\mathbf{Term}$ (not yet terminated) state and observes a *confRespRecv* event (a configuration response message received) before time α without observing the error event *errConfRecv* (unrecoverable protocol error). However, if the system moves to a $\mathbf{Term}$ state (which happens if more than R retry occurs), the automaton moves to location l_1 and waits the CTMC to reach a $\neg\mathbf{Term}$ state, before resetting the clock and restart. This property allows us to compute the probability of observing a configuration error, ignoring multiple retries.

Table 4 reports the performance results of DTA 12(a), varying the retry value R, to investigate how the Comp and OTF methods behave in presence of very few components. Since $\mathcal{M}\times\mathcal{A}$ and $\mathcal{M}\times\mathcal{Z}$ have the same size, Q1 is not relevant.

Q2 *Forward vs Backward.* The state spaces differ (backward case is 60% bigger than forward case). $\text{Comp}^{\mathcal{A}}_{\text{fwd}}$, $\text{Comp}^{\mathcal{Z}}_{\text{fwd}}$, and OTF-refined detect exactly the same component: one of class C_M with 4 629 084 states. In backward all algorithms detect a single large C_M component of 7 303 564 states. This difference is reflected also on the time and memory results, with backward requiring more memory and time than forward.

Q3 *Component vs Full.* Forward techniques uses only two components and the results show that there is still an advantage in time, that comes at the price of a limited increase of memory occupation, indicating that Comp can be used also in presence of only two components. In backward there are four components, and Comp performs better than Full.

Q4 *OTF.* The component construction for $\mathcal{Z}$ observes a loop with reset between l_0 and l_1 that results in the construction of a single D_M component that includes all z-states with location l_0 or l_1, and of the degenerate component

$\boldsymbol{\alpha}$=10, $\boldsymbol{\beta}$=20, $\boldsymbol{\Phi}_1$ = Terminated=0, $\boldsymbol{\Phi}_2$ = mcEstablished=1 $\wedge$ mpEstablished=1

			Forward										**Backward**									
			$\mathcal{M}\times\mathcal{A}$				$\mathcal{M}\times\mathcal{Z}_{Red}$				OTF			$\mathcal{M}\times\mathcal{A}$				$\mathcal{M}\times\mathcal{Z}_{Red}$				OTF	
R	$\lvert\mathcal{M}\rvert$	$\lvert Z\rvert$	states	components			states	components			max size	components			states	components			same as forward			max size	components		
				C_E	C_g	C_M		C_E	C_g	C_M		C_E	C_g	C_M		C_E	C_g	C_M					C_E	C_g	C_M
3	2209	6	6536	1	2	0	4373	0	2	0	2210	0	2	0	6536	1	2	0	4373	0	2	0	2210	0	2	0
5	58854	6	174015	1	2	0	116435	0	2	0	58855	0	2	0	174015	1	2	0	116435	0	2	0	58855	0	2	0
7	597553	6	1766430	1	2	0	1181992	0	2	0	597554	0	2	0	1766430	1	2	0	1181992	0	2	0	597554	0	2	0

Solution time (in seconds) for the 10 methodologies:

R	$\text{Full}^{\mathcal{A}}_{\text{fwd}}$	$\text{Comp}^{\mathcal{A}}_{\text{fwd}}$	$\text{Full}^{\mathcal{Z}}_{\text{fwd}}$	$\text{Comp}^{\mathcal{Z}}_{\text{fwd}}$	OTF_{fwd}	$\text{Full}^{\mathcal{A}}_{\text{bwd}}$	$\text{Comp}^{\mathcal{A}}_{\text{bwd}}$	$\text{Full}^{\mathcal{Z}}_{\text{bwd}}$	$\text{Comp}^{\mathcal{Z}}_{\text{bwd}}$	OTF_{bwd}
3	0.24	0.14	0.22	0.19	0.09	0.22	0.14	0.21	0.14	0.09
5	14.03	4.55	11.14	3.71	2.26	9.10	4.78	8.34	4.05	2.15
7	224.38	59.14	176.89	52.65	31.23	105.78	65.75	142.57	55.84	30.49

Memory occupation (in MBytes):

R	$\text{Full}^{\mathcal{A}}_{\text{fwd}}$	$\text{Comp}^{\mathcal{A}}_{\text{fwd}}$	$\text{Full}^{\mathcal{Z}}_{\text{fwd}}$	$\text{Comp}^{\mathcal{Z}}_{\text{fwd}}$	OTF_{fwd}	$\text{Full}^{\mathcal{A}}_{\text{bwd}}$	$\text{Comp}^{\mathcal{A}}_{\text{bwd}}$	$\text{Full}^{\mathcal{Z}}_{\text{bwd}}$	$\text{Comp}^{\mathcal{Z}}_{\text{bwd}}$	OTF_{bwd}
3	7.59	9.70	7.63	8.54	5.99	7.82	8.08	6.78	7.56	6.11
5	114.98	170.59	83.74	137.56	55.00	131.80	135.78	83.99	117.53	59.35
7	1662.38	1805.73	1274.24	1462.70	547.56	1644.29	1437.92	1075.88	1256.88	592.12

Table 5: Performance result of the DTA 12(b) on the CLUE model.

with the single $\top$ state (with l_2)[3]. Therefore, OTF generates one large (non-uniform) component, both in forward and backward, since it is based on the single D_M component of the region graph. However, in the product of l_1 with $\mathcal{M}$ the edge $l_1 \to l_0$ is never present for those states in which the total number of retries has been exceeded, and therefore Comp correctly identifies that these states constitute a separate C_E component. This allows Comp to perform better than OTF, although the differences are rather small. Note that the performances of OTF match those of Full, which is not surprising considering that OTF is working with a single component. In backward the situation is even worse: OTF works with a single C_M component, while Comp identifies 2 C_E, 1 C_g, and 1 C_M. The cost in time and memory is as for Full, but it is significantly worse than Comp, that takes advantage of the identification of 4 components.

Q5 *OTF + Comp.* The refinement step allows to build the same components as Comp, thus obtaining the same performances of Comp in time and memory, showing that, even in the extreme situation of very few components, OTF can be successfully applied, thanks to the refinement option that allows to correct situations in which there are very large D_M components in the region graph that do not lead to very large C_M components in the MRgP.

Results for CLUE, second DTA. DTA 12(b) implements the semantic of the *time-interval until* operator $\boldsymbol{\Phi}_1\, \mathcal{U}^{[\alpha,\beta]}\, \boldsymbol{\Phi}_2$ of CSL, as in [3]. The automaton stays in location l_0 up to time α, accepting any CTMC action. At time $x = \alpha$ the automaton moves with a boundary edge either to location l_{ok} (if the condition $\boldsymbol{\Phi}_2$ already holds), or to location l_1. In location l_1, the automaton may still accept a CTMC transition that goes to a $\boldsymbol{\Phi}_2$-state before time β, or a CTMC transition that stays in a Φ_1 state. Any other behavior is rejected. Table 5 reports the performance results of DTA 12(b) on the CLUE model, for different retry values R.The atomic propositions $\boldsymbol{\Phi}_1$ and $\boldsymbol{\Phi}_2$ appearing in DTA 12(b) are reported above the table.

DTA 12(b) represents a CSL time-interval Until that computes the probability of both parts in the protocol establishing the connection in the time interval $[10, 20]$ without any retry. Since DTA 12(b) represents a CSL time-interval Until, we know that the formula can be checked by a CSL model checker by computing the transient solution of two CTMCs of a size at most $|\mathcal{M}|$. We recall that solving in transient $\mathcal{M}$ is equivalent to solving a C_g component of size equal to $|\mathcal{M}|$. Since backward and forward see the same number of states and since no C_M component is present, despite the presence of reset edges, Q2 and Q5 are not relevant.

Q1 *z-states tagging.* Tagging reduces the number of states, from about $3 \times |\mathcal{M}|$ to $2 \times |\mathcal{M}|$ (from to 1 766 430 states to 1 181 992 for $R = 7$), as the tagging allows us to tag as NK the z-state $\langle l_1, (\beta, \infty)\rangle$ in which the DTA is in location l_1 and $x > \beta$, since there is no path that leads to the accepting location l_{ok}. This avoids the generation of

[3]Note that $\top$ component is never reported in the tables, as it is never treated as a component, but only as a frontier state

the MRgP states in which the DTA is in location l_1 and $x > \beta$, that are as many as the $\mathcal{M}$ states that satisfies the Φ_1 condition. Methods based on $\mathcal{M}\times\mathcal{A}$ see a "useless" C_E component, which is not generated with $\mathcal{M}\times\mathcal{Z}$ or OTF techniques.

Q3 *Component vs Full.* $\text{Full}^{\mathcal{A}}_{\text{fwd}}$ solves a single MRgP of size almost three times as large as $|\mathcal{M}|$, while $\text{Comp}^{\mathcal{A}}_{\text{fwd}}$ solves two C_g and one C_E components, each one of size almost $|\mathcal{M}|$, requiring two CTMCs transient solutions and a steady state one. Similar behaviour in backward, and when the computation is based on $\mathcal{M}\times\mathcal{Z}$. These differences positively reflects on the solution times.

Q4 *OTF.* This example clearly identifies the advantage of combining the location tagging with the on-the-fly generation of the components. OTF identifies exactly the same components as Comp, but since OTF builds them one at a time based on the partition of $\mathcal{Z}$, its memory occupancy is significantly smaller than Comp and Full. Moreover the solution time is reduced with respect to both Full and Comp. Hence, OTF is capable of reducing the treated state space without impairing the performance (and actually improving it).

Q6 - *OTF on CSL.* OTF builds and solves two C_g components. Each component, of size at most $|\mathcal{M}|$, is built, solved and then discarded. So the OTF complexity matches, in time and space, that of standard CSL model checking algorithms, which was one of the motivations for the development of the OTF technique.

7.3. Comparison with Prism and Storm CSL model checkers

While question Q6 has already been investigated in the previous example, showing that OTF builds the same CTMCs and applies the same solution algorithms as standard CSL model checkers, it is still to be determined whether the performance of the OTF implementation matches that of a model checker specifically developed for CSL. Table 6 shows a comparison of MC4CSL$^{\text{TA}}$ with two CSL model checking tools: Prism [8], which is well-known and very popular in research and industries, and Storm [11], a recently developed tool. We have considered a model from the Prism distribution, which can be uploaded[4] by both Storm (original Prism model) and GreatSPN (CTMC generated by Prism). Other CSL model checkers exist, like [12], which however do not share this specific format, moreover our goal is not to "benchmark" CSL model checkers, but only to better understand how the techniques presented in this paper behave on CSL formulas.

The model considered is the cell cycle control in eukaryotes [37], which represents a chemical reaction network of several proteins. The goal of the tested query is to determine if the quantity of the cyclin protein is sufficient after a certain amount of time. We test two bounded Until queries: (case A) Until(α, α) and (case B) a timed interval one, Until(α, β). The formulas are specified in CSL$^{\text{TA}}$ using DTAs: case B uses the DTA in Figure 12(b), while the DTA of case A is obtained by removing location l_1.

Table 6 reports the performance of the three tools for the two tested queries. Each of the two tables report on top the query in CSL and in CSL$^{\text{TA}}$ and on the first three columns the model parameter N and the CTMC size (states and transitions). Prism has been tested using both the sparse and the hybrid engine. Note that in Prism the Sat set is computed, using a backward technique. Storm has been tested using the default configuration. The MC4CSL$^{\text{TA}}$ tool is tested in $\text{Full}^{\mathcal{Z}}_{\text{bwd}}$, $\text{Comp}^{\mathcal{Z}}_{\text{bwd}}$ and OTF_{bwd} modes. The most comparable techniques are the sparse engine of Prism, the default Storm engine and the OTF_{bwd} method. Each table reports also the total number of components in the CSL$^{\text{TA}}$ region graphs, and the size of each individual component. Both the $\text{Comp}^{\mathcal{Z}}_{\text{bwd}}$ and OTF_{bwd} methods observe the same components.

OTF_{bwd} and the sparse engine of Prism have similar time performance, which is not surprising, since they execute similar tasks on the same CTMC. For the first query, Prism performs transient analysis on one CTMC, as MC4CSL$^{\text{TA}}$. Storm shows remarkably good numerical solution times, which are probably doe to the use of highly optimized linear algebra libraries that take advantage of the CPU SIMD instructions. MC4CSL$^{\text{TA}}$ determines that the region graph of the interval Until DTA has one component of type D_{g_1}, which includes a single z-state, and one of type D_E. The D_E component includes only states tagged NK and therefore they are not part of the $\mathcal{M}\times\mathcal{Z}$ product (neither of the $\mathcal{M}\times\mathcal{Z}_i$ product built by OTF). This results in MC4CSL$^{\text{TA}}$ identifying a single component of type C_{g_1}, which is solved for time 10 using transient analysis.

[4] We thank the Storm team for support in importing the Prism model into Storm

(A) CSL query (Prism/Storm): P=? [true U[10,10] *cyclin_bound*=N].
CSL^{TA} query (MC4CSL^{TA}): PROB=? until_AA (10 | | True, (*#cyclin_bound*=N))

			Prism				Storm		MC4CSL^{TA}									
			Sparse		Hybrid		Sparse		$\text{Full}^{\mathcal{Z}}_{\text{bwd}}$		$\text{Comp}^{\mathcal{Z}}_{\text{bwd}}$		OTF_{bwd}		Components			Size
N	States	Trns	Time	Memory	Time	Memory	Time	Memory	Time	Memory	Time	Memory	Time	Memory	C_E	C_g	C_M	$\lvert\mathcal{S}_1\rvert$
2	4666	18342	0.0	114.9	0.0	114.8	0.0	221.0	0.1	10.5	0.1	9.4	0.1	9.3	0	1	0	4666
3	57667	305502	0.2	119.3	0.5	122.3	0.1	230.2	1.7	59.6	0.6	47.0	0.6	47.6	0	1	0	57667
4	431101	2742012	3.3	173.9	6.7	172.5	1.2	295.1	21.1	424.8	7.3	350.2	6.7	352.8	0	1	0	431101
5	2326666	16778785	30.0	336.4	63.3	285.7	9.7	999.4	170.1	2312.6	56.0	2033.7	52.1	2082.3	0	1	0	2326666

(B) CSL query (Prism/Storm): P=? [true U[10,20] *cyclin_bound*=N].
CSL^{TA} query (MC4CSL^{TA}): PROB=? until_AB (10, 20 | | True, (*#cyclin_bound*=N))

			Prism				Storm		MC4CSL^{TA}										
			Sparse		Hybrid		Sparse		$\text{Full}^{\mathcal{Z}}_{\text{bwd}}$		$\text{Comp}^{\mathcal{Z}}_{\text{bwd}}$		OTF_{bwd}		Components			Size	
N	States	Trns	Time	Memory	Time	Memory	Time	Memory	Time	Memory	Time	Memory	Time	Memory	C_E	C_g	C_M	$\lvert\mathcal{S}_1\rvert$	$\lvert\mathcal{S}_2\rvert$
2	4666	18342	0.0	114.3	0.0	110.9	0.0	221.1	0.2	11.8	0.1	11.5	0.1	10.2	0	2	0	4666	3855
3	57667	305502	0.5	123.4	0.9	120.5	0.2	231.5	4.7	85.6	1.3	81.8	1.0	59.7	0	2	0	57667	51479
4	431101	2742012	6.5	177.4	12.8	169.1	2.1	295.4	50.5	803.3	15.1	654.4	10.7	462.0	0	2	0	431101	399016
5	2326666	16778785	58.9	352.2	119.8	300.3	17.3	1063.5	427.4	4569.2	115.2	4123.6	84.2	2589.4	0	2	0	2326666	2198758

Table 6: Comparison with Prism and Storm on CSL Until queries.

For the second query Prism and Storm perform the transient analysis of two CTMCs. Referring to the DTA of Figure 12(b) the region graph of the time-interval Until has 6 z-states that are partitioned into three components: two of class D_g with, respectively, the single z-states $\langle l_0, [0, \alpha)\rangle$ and $\langle l_1, [\alpha, \beta)\rangle$ and one of class D_E with the remaining four z-states, that are all tagged as NK. OTF therefore computes and solves two C_g components, resulting in two transient analysis. Therefore, in both cases MC4CSL^{TA} performs the same solution steps as Prism, but these steps depend on the input DTA and do not use any a-priori knowledge on the formula: indeed OTF *adapts its behaviour to the formula being checked*. The relative value of the solution times for the three tools follows the same pattern as for the previous formula.

Memory occupation of the CSL^{TA} model checker is significantly higher than CSL ones: about 7 times more than Prism and 2.5 times more Storm in the worst case. This difference in memory can be due to a combination of factors: CSL model checkers do not need to compute frontiers, thus avoiding a number of intermediate data structures, and the prototype code of MC4CSL^{TA} makes use of standard data structures that may not be the best choice (i.e. using C++ STL containers like map for the state space/spare vectors instead of more compact data structures).

7.4. Results summary

Q1 - *z-states tagging.* Results show that it is always worth applying this technique. Since time and memory are always lower when there is a reduction in the MRgP size, but even when there is no reduction (as for the third FMS case and the first CLUE one) the additional cost is negligible.

Q2 - *Forward vs Backward.* Differences can be rather large here, which is not surprising considering that backward computes the set of states that satisfies a CSL^{TA} formula, while forward only checks if the initial state satisfies the formula. Forward is not an option when formulas are nested, that is to say a location is labelled with another CSL^{TA} formula: a case that is not part of our test set, since formulas tend to be rather artificial, but nested CSL^{TA} is supported by the MC4CSL^{TA} tool. Obviously when the objective is to test a given single state, forward should be preferred.

Q3 - *Comp vs Full.* Comp is always quicker than Full, both forward and backward. In memory we have observed for Comp at most a 10% increase in memory over Full, but there are also cases, like the second and the third FMS case, in which memory of Comp is significantly better than Full, due to the reduced size of the iteration vectors. The gain is particularly significant when the number of components is large and components are balanced in size, but Comp can be successfully used also in presence of very few components, as for the CLUE models.

Q4 and Q5 - *OTF and OTF+Comp.* OTF is always superior to the other techniques when the number of components is not too small and their size is well balanced. When this is not the case, in particular when it is not the case with respect to Comp, we have always observed that it is worth to try to use OTF+Comp. One source of efficiency of

OTF w.r.t. Comp is the computation of the components, that OTF builds on the region graph while Comp builds them on the whole MRgP. The region graphs of our examples have at most 60 states, while MRgP size goes into millions.

Q6 - *CSL*. It was already shown in [17] that the component method builds the same CTMCs as do model checkers of CSL. Our experiments show that also OTF is able to build and solve the same CTMCs as standard CSL model checkers. In particular we have observed that MC4CSL^{TA} is able to have performance in time comparable to that of PRISM (no more than two times slower), while there is a significantly worse performance in terms of memory (up to 10 times bigger).

8. Related work

This paper builds on previous works for CSL and CSL^{TA} model checking and MRgP solution, that we shall here review. CSL was first defined in [3], where CSL model checking was shown to be decidable. In the original definition of CSL, paths could be specified as a sequence of timed Until formulae (nested LTL Until), a feature that has been limited in the definition of CSL given in [38] where timed Until formulae cannot be nested, and steady-state operators have been added. This is the most widely used form of CSL, and is the one we have used as reference in this paper. This limitation allows one to model check a CSL formula using well-known CTMC solution algorithms (either in transient or in steady-state). In particular the work in [38] shows that the computation of the probability of the paths of a CTMC that satisfy a timed Until $\Phi_1\, U^{[t,t']}\, \Phi_2$ for a CTMC $\mathcal{M}$ reduces to solving two CTMCs derived from $\mathcal{M}$, the first one is solved at time t and its solution is the initial distribution for the second one, solved at time $t' - t$.

Algorithms for forward and backward model checking of CTMCs have been discussed in [39] and [25]. The backward solution is actually a rephrasing in the CSL context of the computation of absorbing probabilities in non-ergodic Markov chains [20]. An extension for backward solution of non-ergodic MRgPs has been provided in [26], which is what we have used in this paper.

In CSL paths are specified in terms of state propositions of the CTMC. The logic asCSL [40] and [4] specifies path properties as regular expressions of state propositions *and actions names* of the so-called "continuous-time Markov chain with actions and state labels (ASMC)". A path expression is translated into an automaton (untimed), and the model checking is based on the analysis of the CTMC that results from the cross-product of the ASMC with the automaton, to reduce the asCSL model checking problem to the CSL one.

CSL^{TA} [13] also specifies path properties through automata but, unlike asCSL where the time requirement is associated to the whole expression, in CSL^{TA} the time is specified in the automaton itself and can thus specify time requirements also for sub-paths. This allows the analyser more freedom, as shown in the examples of Section 7, but the cross-product generated by a CSL^{TA} formula requires a MRgP solution instead of a CTMC one.

In this paper we generically refer to CSL^{TA}, meaning the original definition in [13]. Unfortunately the successive extension to multiple-clocks of CSL^{TA} in [15] does not immediately reduce to the original definition of CSL^{TA} when a single clock is considered, so does the work in [19] that considers only single clock DTAs and infinite CTMCs and that takes as starting point the CSL^{TA} definition of [15]. The difference is that in the DTA definition in [13] it is possible to associate an expression over CTMC state propositions to the DTA locations, while in [15] the expression over CTMC state propositions are associated to the DTA edges. This lead to a slightly different acceptance criteria for timed paths. So, while it is proved that CSL^{TA} is more powerful than CSL and asCSL [13], we do not know of a proof for the relationship between the original definition of CSL^{TA} and that in [15] when limited to a single clock. This is indeed an interesting research question, but it is out of the scope for this paper, that concentrates on model checking algorithms for single clock CSL^{TA} and their relationship to CSL model checking, and not on language comparison.

Other works on CSL^{TA} model-checking consider a two-steps approach in the construction of the stochastic process. The work in [15] uses an intermediate structure (called DMTA) which is a cross product of the CTMC with the locations of the timed automaton, to get rid of CTMC states and DTA locations that do not match. DMTA are a sort of timed automaton with rate extensions, and from DMTA the underlying stochastic process is built based on the region graph of the DMTA. In the 1-clock case, the stochastic process is then specified by a set of linear equations that identify the MRgP. When the DTA allows multiple-clocks the stochastic process is not a MRgP any longer, but a Piece-wise Deterministic Markov Process and the solution technique is more complicated and expensive. Although the equations are built by partitioning the region graph, the solution considers the full system of equations of all the region graph partitions, and cannot be computed by taking each region in isolation.

The works in [18] and [19] also use a two-steps approach, but they generate first the region graph of the DTA and then use it in a cross-product with $\mathcal{M}$, similarly to what is done in this paper for algorithms with $\mathcal{Z}$ superscript. In [18] this choice is meant to avoid the construction of non-useful parts of the MRgP. A comparison with [18] has been provided at the end of Section 4. The work in [19] introduces a technique for the model-checking of CSL$^{\text{TA}}$ for DTAs that do not include boundary edges and for CTMCs with an unlimited number of states. There is no backward approach proposed in [19], presumably because of the requirement to work also with infinite state spaces. The method uses the region graph to partition the infinite MRgP into subsets of states (called "columns" in the paper), one per clock region. We can therefore see the work in [19] as similar to a component method in which the components are identified by the clock regions. As already discussed with respect to the MRgP structure of Figure 4 this structure of components does not form a DAG and indeed the technique only computes an approximate solution, but this is certainly adequate considering that the goal is the model-checking of infinite CTMCs.

The model-checking algorithm in [19] proceeds iteratevely over all regions. Since the CTMC is infinite, the MRgP states associated to a given clock region are built while "moving" the probability from a state to its successors: if a new state with a probability above a threshold is encountered, that state is retained and used for successive generation of the state space. The computation of the probability for a region is basically a modified uniformization (to deal with the on-the-fly generation of states) that computes the probability that should go to the states in the next region or that should go back to the states of the initial region (due to clock resets). Therefore also the iteration over all regions should be iterated. The iteration terminates when the probability of transient states goes below a certain level. Note that the same technique is applied to all regions, also to the last one that typically requires a steady state solution (and therefore the uniformization may be very expensive). Again, this is caused by the need to deal with infinite CTMCs. Moreover the solution remains approximate even on finite CTMCs, since clock resets are treated by repeating the solution multiple times until the remaining probability is below a threshold. This work shares with the OTF technique of Algorithm 3 the idea of generating state spaces only when needed, but the technique is iterative and therefore, in presence of clock resets, each component can be visited more than once, and the components cannot be discarded once used in a computation, which are instead two peculiar aspects of OTF. Another difference is that in OTF the structure of the components is determined by a specific algorithm and it is not pre-defined as the set of the states in the same clock region. As it was shown in the experimental part (for example for the case reported in Table 2), the number of components can be much higher than the number of regions, as inside a region we can identify a DAG of components, which may save solution time (components are smaller) and memory (components are discarded once used).

Finally, no numerical comparison has been provided in the previous section with the technique in [19], since it is an approximate solution, nor with the technique of [15], since, obviously, multiple clock CSL$^{\text{TA}}$ model checking has a much higher complexity than that of plain CSL$^{\text{TA}}$, and comparison will be unfair.

9. Conclusion and future work

This paper presents a new model-checking algorithm called OTF for the stochastic logic CSL$^{\text{TA}}$ that exploits the region graph construction of the automaton to devise a solution in which the MRgP is built and solved "on the fly": the MRgP is built component by component, and each component is built, solved with the cheapest possible technique, and then discarded, with a clear advantage in both time and space. OTF is defined to work both in forward and backward, and it exploit a matrix-free approach to avoid the expensive construction of the embedded Markov chain. OTF builds on the Component Method for MRgP, that has been reviewed and re-defined in some parts to make its definition suitable for the proof of correctness of OTF. The region graph is also exploited to limit the number of non-useful states that are constructed and solved thanks to a new definition of the MRgP construction based on the region graph.

OTF, algorithms that build the whole MRgP based on the region graphs, and algorithm that build directly the MRgP, as in the original model checking algorithm of CSL$^{\text{TA}}$, for a total of 12 algorithms, have been implemented in the MC4CSL$^{\text{TA}}$ tool, to ease the experimental comparison. MC4CSL$^{\text{TA}}$ is part of the GreatSPN framework. In GreatSPN the CTMC is derived from a graphical specification of a GSPN, and the tool has been extended to provide a graphical support for DTAs specification, and the possibility of playing a joint "token game" to experiment with path acceptance. All the nets and the DTAs drawings of this paper have been made with GreatSPN.

Theory and experiments suggests that OTF (possibly with additional component refinement) could be a good default choice for the model checking of the CSL^{TA} logic, as it may save a significant amount of memory. It may also save solution time in the identification of the components: computing an optimal set of components can be very expensive, as it requires the solution of an ILP problem in the size of the number of states, therefore it is usually much cheaper to build the components of the (usually) small region graph instead of the (usually) large MRgP. We have also observed that the overhead of OTF is small enough even in those unfavourable cases in which the number of components is very small (even only one or two).

The paper also shows that CSL^{TA} model checking based on OTF has the capability of adapting to the actual complexity of the property expressed by the DTA: this allows to show that OTF computes and solves exactly the same CTMCs as do standard CSL model checking algorithms when the formula is a CSL. With respect to a CSL model checker like Prism or Storm, the use of a OTF in $MC4CSL^{TA}$ allows to deal also with CSL formulas that include both state properties and action names (like in asCSL).

As part of future work we plan to improve OTF along two ways: to devise a modified version that stops as soon as the property is verified and to perform a better analysis of memory occupation to understand if there are inefficiency in memory allocation that can be removed. We also plan to exploit Kronecker-based approaches. Kronecker-based matrix-free solution of MRgP has been introduced in [41]. It allows to compute the steady-state probability of an ergodic MRgP without ever building and storing the $\mathbf{Q}$, $\bar{\mathbf{Q}}$, and $\mathbf{\Delta}$ matrices, but only a (much more compact) Kronecker expression of them. We have to face two problems here: MRgPs stemming from model checking are typically non-ergodic and the computation of the outgoing probability vectors require the computation of transient CTMC solutions. At the moment, an efficient way for computing a Kronecker based transient solutions of a non-ergodic CTMC is not available.

Another interesting line of research to pursue is to investigate whether the approach in [19] and OTF can be combined. In particular we can envision to use the approximate solution technique of [19] to solve large MRgP components of C_M type. In the opposite direction we can investigate whether the definition of components at the region graph level can be exploited to provide better performance or better precision to the approximate technique, that, in its current status, uses the clock regions to identify components.

Acknowledgements

The authors would like to thank the anonymous reviewers for their exceptional service and Jeremy Sproston for proofreading the manuscript.

References

[1] E. Emerson, E. M. Clarke, Using branching time temporal logic to synthesize synchronization skeletons, Science of Computer Programming 2 (3) (1982) 241 – 266.

[2] A. Pnueli, The temporal logic of programs, in: Proceedings of the 18th Annual Symposium on Foundations of Computer Science, SFCS '77, IEEE Computer Society, Washington, DC, USA, 1977, pp. 46–57.

[3] A. Aziz, K. Sanwal, V. Singhal, R. Brayton, Model-checking continuous-time Markov chains, ACM Transactions on Computational Logic 1 (1) (2000) 162–170.

[4] C. Baier, L. Cloth, B. R. Haverkort, M. Kuntz, M. Siegle, Model Checking Markov Chains with Actions and State Labels, IEEE Transactions on Software Engineering 33 (2007) 209–224.

[5] C. Baier, B. Haverkort, H. Hermanns, J.-P. Katoen, On the Logical Characterisation of Performability Properties, Springer Berlin Heidelberg, Berlin, Heidelberg, 2000, pp. 780–792.

[6] M. Kuntz, B. R. Haverkort, GCSRL-a logic for stochastic reward models with timed and untimed behaviour, in: Proceedings of The Eighth International Workshop on Performability Modeling of Computer and Communication Systems, Centre for Telematics and Information Technology, University of Twente, 2007, pp. 50–56.

[7] M. Ajmone-Marsan, G. Balbo, G. Conte, S. Donatelli, G. Franceschinis, Modelling with Generalized Stochastic Petri Nets, Wiley & Sons, 1995.

[8] M. Kwiatkowska, G. Norman, D. Parker, PRISM: Probabilistic Model Checking for Performance and Reliability Analysis, Performance Evaluation 36 (4) (2009) 40–45.

[9] J.-P. Katoen, I. S. Zapreev, E. M. Hahn, H. Hermanns, D. N. Jansen, The ins and outs of the probabilistic model checker MRMC, Performance Evaluation 68 (2011) 90–104.

[10] H. Hermanns, J.-P. Katoen, J. Meyer-Kayser, M. Siegle, A tool for model-checking Markov chains, International Journal on Software Tools for Technology Transfer 4 (2) (2003) 153–172.

[11] C. Dehnert, S. Junges, J.-P. Katoen, M. Volk, A storm is coming: A modern probabilistic model checker, arXiv preprint arXiv:1702.04311. URL http://www.stormchecker.org/
[12] M. Schwarick, M. Heiner, C. Rohr, Marcie - model checking and reachability analysis done efficiently, in: Int. conf. on Quantit. Eval. Systems, 2011, pp. 91–100.
[13] S. Donatelli, S. Haddad, J. Sproston, Model checking timed and stochastic properties with CSL^{TA}, IEEE Transactions on Software Engineering 35 (2) (2009) 224–240.
[14] E. G. Amparore, S. Donatelli, MC4CSL^{TA}: an efficient model checking tool for CSL^{TA}, in: International Conference on Quantitative Evaluation of Systems, IEEE Computer Society, Los Alamitos, CA, USA, 2010, pp. 153–154.
[15] B. Barbot, T. Chen, T. Han, J.-P. Katoen, A. Mereacre, Efficient ctmc model checking of linear real-time objectives, in: Proceedings of 17th Int. Conf. on Tools and algorithms for the construction and analysis of systems (TACAS), Springer, 2011, pp. 128–142.
[16] P. Ballarini, H. Djafri, M. Duflot, S. Haddad, N. Pekergin, COSMOS: a statistical model checker for the hybrid automata stochastic logic, in: Proceedings of the 8th International Conference on Quantitative Evaluation of Systems (QEST'11), IEEE Computer Society Press, Aachen, Germany, 2011, pp. 143–144.
[17] E. G. Amparore, S. Donatelli, A component-based solution for reducible Markov regenerative processes, Performance Evaluation 70 (6) (2013) 400 – 422.
[18] E. G. Amparore, S. Donatelli, Improving and assessing the efficiency of the MC4CSL^{TA} model checker, in: Computer Performance Engineering, Vol. 8168 of LNCS, Springer Berlin Heidelberg, Venice, 2013, pp. 206–220.
[19] L. Mikeev, M. R. Neuhäußer, D. Spieler, V. Wolf, On-the-fly verification and optimization of DTA-properties for large Markov chains, Formal Methods in System Design 43 (2) (2013) 313–337.
[20] V. G. Kulkarni, Modeling and analysis of stochastic systems, Chapman & Hall Ltd., London, UK, 1995.
[21] M. Ajmone Marsan, G. Chiola, On Petri nets with deterministic and exponentially distributed firing times, in: Advances in Petri Nets, Vol. 266/1987 of LNCS, Springer, 1987, pp. 132–145.
[22] H. Choi, V. G. Kulkarni, K. S. Trivedi, Markov regenerative stochastic Petri nets, Performance Evaluation 20 (1-3) (1994) 337–357.
[23] C. G. Cassandras, S. Lafortune, Introduction to Discrete Event Systems, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.
[24] R. German, Iterative analysis of Markov regenerative models, Performance Evaluation 44 (2001) 51–72.
[25] E. G. Amparore, S. Donatelli, Backward solution of Markov chains and Markov renewal processes: formalization and applications, in: Sixth International Workshop on Practical Applications of Stochastic Modelling (PASM12), Vol. 296, Elsevier, 2012, pp. 7–26.
[26] E. G. Amparore, S. Donatelli, Revisiting the matrix-free solution of Markov regenerative processes, Numerical Linear Algebra with Applications, Special Issue on Numerical Solutions of Markov Chains 18 (2011) 1067–1083.
[27] K. S. Trivedi, A. Bobbio, G. Ciardo, R. German, A. Puliafito, M. Telek, Non-Markovian Petri nets, in: ACM SIGMETRICS Performance Evaluation Review, Vol. 23, ACM, 1995, pp. 263–264.
[28] J. Bengtsson, W. Yi, Timed automata: Semantics, algorithms and tools, in: Lectures on Concurrency and Petri Nets: Advances in Petri Nets, Springer Berlin Heidelberg, Berlin, Heidelberg, 2004, pp. 87–124.
[29] T. Chen, T. Han, J.-P. Katoen, A. Mereacre, Model checking of continuous-time Markov chains against timed automata specifications, Logical Methods in Computer Science (2011) 7 (1).
[30] E. G. Amparore, S. Donatelli, Optimal aggregation of components for the solution of Markov regenerative processes, in: Quantitative Evaluation of Systems: 13th International Conference, QEST 2016, Quebec City, QC, Canada, August 23-25, Proceedings, Springer International Publishing, Cham, 2016, pp. 19–34.
[31] E. G. Amparore, G. Balbo, M. Beccuti, S. Donatelli, G. Franceschinis, 30 Years of GreatSPN, Springer, Cham, 2016, Ch. In: Principles of Performance and Reliability Modeling and Evaluation: Essays in Honor of Kishor Trivedi, pp. 227–254.
[32] M. Ajmone Marsan, G. Conte, G. Balbo, A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems, ACM Transactions on Computer Systems 2 (1984) 93–122.
[33] M. Aldinucci, S. Bagnasco, S. Lusso, P. Pasteris, S. Vallero, S. Rabellino, The Open Computing Cluster for Advanced data Manipulation (OCCAM), in: 22nd Int. Conf. on Computing in High Energy and Nuclear Physics, San Francisco, 2016.
[34] G. Balbo, M. Beccuti, M. De Pierro, G. Franceschinis, First Passage Time Computation in Tagged GSPNs with Queue Places, The Computer Journal 54 (2010) 653–673, first published online July 22, 2010.
[35] Y. Saad, M. H. Schultz, GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems, SIAM Journal on Scientific and Statistical Computing 7 (3) (1986) 856–869.
[36] R. Presta, S. Romano, 2nd draft of the clue internet protocol, ietf, http://tools.ietf.org/pdf/draft-ietf-clue-protocol-02.pdf (2013).
[37] Lecca, Priami, Cell Cycle Control in Eukaryotes - Prism case studies, http://www.prismmodelchecker.org/casestudies/cyclin.php (2011).
[38] C. Baier, B. Haverkort, H. Hermanns, J.-P. Katoen, Model-Checking Algorithms for Continuous-Time Markov Chains, IEEE Transactions on Software Engineering 29 (6) (2003) 524–541.
[39] J.-P. Katoen, M. Z. Kwiatkowska, G. Norman, D. Parker, Faster and symbolic CTMC model checking, in: Proceedings of the Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification, PAPM-PROBMIV '01, Springer-Verlag, London, UK, 2001, pp. 23–38.
[40] C. Baier, L. Cloth, B. Haverkort, M. Kuntz, M. Siegle, Model checking action- and state-labelled Markov chains, in: International Conference on Dependable Systems and Networks, 2004, 2004, pp. 701–710.
[41] E. G. Amparore, P. Buchholz, S. Donatelli, A structured solution approach for Markov Regenerative Processes, in: G. Norman, W. Sanders (Eds.), Quantitative Evaluation of Systems: 11th International Conference, QEST 2014, Florence, Italy, September 8-10, 2014. Proceedings, Springer International Publishing, Cham, 2014, pp. 9–24.

Appendix A. Petri nets of the test models

The FMS model used for the first set of tests in Section 7 is depicted in Figure A.13, taken from [34]. The model represents a system where N pallets are treated in a sequence of four machines, $M_1 \dots M_4$. Each machine can treat one pallet at a time. Machine 2 and 3 are subject to breakages, and a repairman continuously checks the machine for repairs. Machine 2 has a set of spare parts that can be used to replace the broken parts, without losing work time. Machine 3 instead always requires a stop to do the repair. The model is parametric in the number N of circulating pallets, and is ergodic.

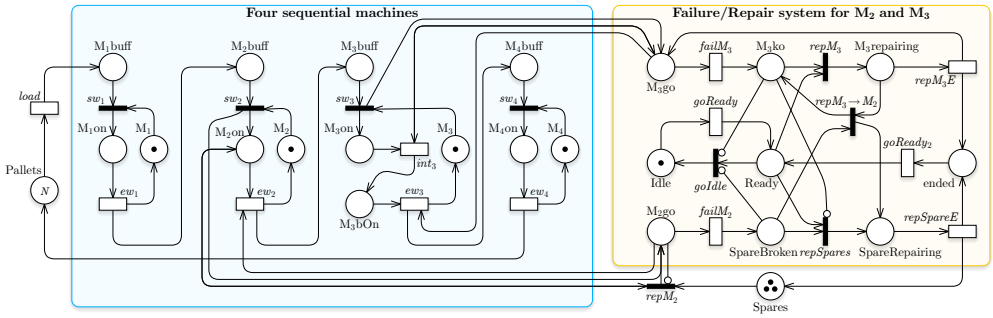


Figure A.13: Petri net of the *Flexible Manufacturing System* with four machines and the repairing net.

Figure A.14 depicts the Petri net used for the second set of tests in Section 7. It is a model of the CLUE protocol [36], a communication protocol of the application layer designed by the CLUE working group at IETF. CLUE is an application protocol for the negotiation of a telepresence session between multiple participants. After a session establishment phase, the participants exchange messages on a data channel in the form of XML descriptors. In particular, parties have to exchange a configuration (CONF) descriptor to adapt their media streams. Each participant acts both as a Media Provider (MP) and as a Media Consumer (MC). Network errors may happen at any time, and the protocol allows at most R retry attempts, before terminating with failure. The communication is completed when, after having exchanged the CONF message with success, both the MP and MC are in the established states (i.e. telepresence sessions may start).

Appendix B. Proof of theorem 1

We now prove Theorem 1 that asserts that $\mathcal{M}\times\mathcal{A}$ and $\mathcal{M}\times\mathcal{Z}$ represent the same MRgP process.

Proof. The proof consists in showing that the set of initial states is the same, and that for each transition in $\mathcal{M}\times\mathcal{A}$ there is one of the same type in $\mathcal{M}\times\mathcal{Z}$ and viceversa. Moreover the set of final states is the same.
We start by defining the obvious correspondence between $\mathcal{M}\times\mathcal{A}$ and $\mathcal{M}\times\mathcal{Z}$ states: $\langle s, l, c\rangle$, with $c = (\delta_k, \delta_{k+1})$ in $\mathcal{M}\times\mathcal{A}$ corresponds to a state $\langle s, z\rangle$ with $z = (l, [\delta_k, \delta_{k+1}))$ and viceversa. As a consequence two corresponding states have the same value of the Γ function.

The *fin* function assigns $\top$ to a $\langle s, l, c\rangle$ of $\mathcal{M}\times\mathcal{A}$ if $l \in L_F$, while in $\mathcal{M}\times\mathcal{Z}$ this is conditioned to $z \in Z_F$, but since Z_F is defined based on L_F then *fin* assigns $\top$ to corresponding states.

We also need to show that $\mathit{closure}(s, l, c) = \langle s', l', c'\rangle$ iff $\mathit{closure}(s, z) = \langle s', z'\rangle$ with $z = (l, c)$ and $z' = (l', c')$. Moreover $\mathit{closure}(s, l, c) = \mathit{fin}(s, l, c)$ iff $\mathit{closure}(s, z) = \mathit{fin}(s, z)$. Let's consider the right implication. Assume $c = [\delta_k, \delta_{k+1})$. According to its definition, if there is a Boundary edge $l \xrightarrow{\delta_k, \sharp, r} l'$, with $s \models \Lambda(l')$, then $\mathit{closure}(s, l, c) = \mathit{closure}(s, l', c[r := 0])$. But if there is such an edge then (for the last rule of the region graph construction), there is an edge from z to z', with $z' = \langle l', c[r := 0]\rangle$. The viceversa follows the same reasoning. If such an edge does not exists in $\mathcal{A}$, then it does not exist the corresponding one in $\mathcal{Z}$, and viceversa, and closures end-up in $\mathit{fin}(s, l, c)$ and $\mathit{fin}(s, z)$, that have been already shown to be corresponding states.

Since they have an isomorphic representation they correspond to the same Markov Regenerative Process, and therefore the probability of reaching the $\top$ state is the same □

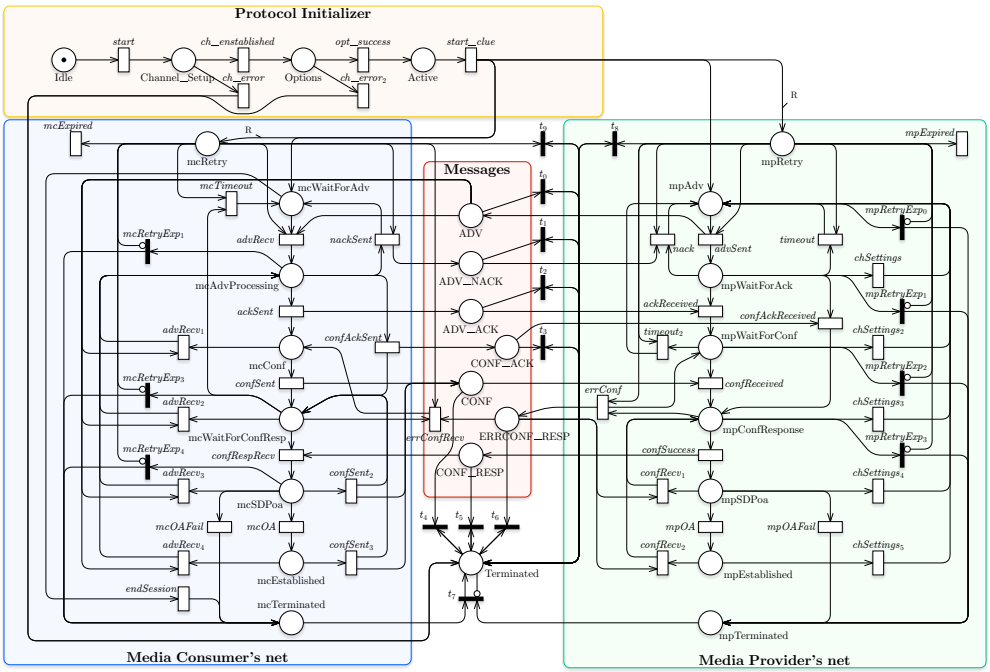


Figure A.14: Petri net of the *CLUE* IETF protocol.