IRIS AperTO

UNIVERSITÀ DEGLI STUDI DI TORINO

AperTO - Archivio Istituzionale Open Access dell'Università di Torino

Improving and Assessing the Efficiency of the MC4CSLTA Model Checker

This is the author's manuscript

Original Citation:

Availability:

This version is available http://hdl.handle.net/2318/1624731 since 2017-02-14T12:31:35Z

Publisher:

Springer Berlin Heidelberg

Published version:

DOI:10.1007/978-3-642-40725-3_16

Terms of use:

Open Access

Anyone can freely access the full text of works made available as "Open Access". Works made available under a Creative Commons license can be used according to the terms and conditions of said license. Use of all other works requires consent of the right holder (author or publisher) if not exempted from copyright protection by the applicable law.

(Article begins on next page)

08 September 2026

Improving and assessing the efficiency of the MC4CSL^{TA} model checker

Elvio G. Amparore, Susanna Donatelli

University of Torino, Corso Svizzera 187, Torino, Italy,
amparore@di.unito.it and susi@di.unito.it

Abstract. CSL^{TA} is a stochastic logic which is able to express properties on the behavior of a CTMC, in particular in terms of the possible executions of the CTMC (like the probability that the set of paths that exhibits a certain behavior is above/below a certain threshold). This paper presents the new version of the the stochastic model checker MC4CSL^{TA}, which verifies CSL^{TA} formulas against a Continuous Time Markov Chain, possibly expressed as a Generalized Stochastic Petri Net. With respect to the first version of the model checker presented in [1], version 2 features a totally new solution algorithm, which is able to verify complex, nested formulas based on the timed automaton, while, at the same time, is capable of reaching a time and space complexity similar to that of the CSL model checkers when the automaton specifies a neXt or an Until formulas. In particular, the goal of this paper is to present a new way of generating the MRP, which, together with the new MRP solution method presented in [2] provides the two cornerstone results which are at the basis of the current version. The model checker has been evaluated and validated against PRISM [3] (for whose CSL^{TA} formulas which can be expressed in CSL) and against the statistical model checker Cosmos[4] (for all types of formulas).

1 Introduction

System verification is a topic whose relevance increases with the increase of the dependency of everyday life from software systems. The more our society relies on computer-based systems, more critical is the demand for system reliability. Model checking of temporal logics has represented an important milestone in the computer science approach to verification, allowing the exhaustive check of systems with billions of states and more. Temporal logics allows to express invariant properties, like "in all states variable x is positive", as well as path-dependent properties, like "it exists a system execution (a path) in which variable x is always incremented after a decrement of variable y". When the system at hand includes timing aspects, temporal logic can be extended to include constraints over time intervals, like "on all paths the lift door will open within 2 seconds after the lift reaches the target floor". Finally, when the system description includes also probabilistic aspects, its evaluation and verification can be based on stochastic logics that allows properties like: "with probability greater than α the lift door will open within 2 seconds after the lift reaches the target floor".

The most well known stochastic logic is CSL [5], for which various verification engines (model-checkers) exists, like Prism [3], MRMC [6] and Marcie [7], to verify behaviour of DTMC or CTMC, possibly generated from higher level languages, like the guarded command language of Prism or the stochastic Petri nets of Marcie. CSL has a predefined set of operators to specify the paths of interest (called neXt and Until), and this might constitute a limitation. This restriction is nevertheless well motivated by the fact that the verification of these formulas only requires transient and steady-state solution of the chain.

CSL^{TA} [8] represents a step forward, as it allows to specify the paths of interest as the set of paths accepted by a single clock timed automaton [9], where the restriction to single clock is the key to allow the verification of CSL^{TA} formulas in terms of the steady-state solution of a Markov Regenerative Process (MRP). A multi clock CSL^{TA} has been defined in [10], but the gain in formula expressiveness is paid in solution terms, as the underlying process becomes a piece-wise deterministic process (PDP). In terms of model checkers, single clock CSL^{TA} formulas can be verified using MC4CSL^{TA} [1], and single and multiple clock formulas using Codemoc [11]. More recently the statistical model checker Cosmos [4] has been delivered. The tool uses simulation to verify formulas specified through timed (and even hybrid) automata [12], for a stochastic model that is a Stochastic Petri Net extended to arbitrary distributions for the transition firing.

This paper concentrates on CSL^{TA} and presents the new version of the model checker MC4CSL^{TA}. Version 2 solves a few inefficiencies of the first version, as will be explained in the next section, and introduces a totally new solution engine. As we shall see, the translation into Deterministic Stochastic Petri Net (DSPN) has been removed; to allow the evaluation of nested formulas the implementation now includes forward and backward solution; different MRP solution methods can now be employed (in particular the efficient component-based approach described in [2]), and non useful computations are removed thanks to a pre-analysis of the timed automata of the formula. This theory behind the pre-analysis and an assessment of the correctness and efficiency of MC4CSL^{TA} version 2 are the main contributions of this paper. Most of the work presented in this paper is the result of the PhD work of the first author [13].

2 Background and motivations

In this section we review the basic literature in CSL and CSL^{TA} and related tools and discuss the status of the MC4CSL^{TA} model checker, version 1 and the limits that the tool has shown in verifying "large" systems. The focus will be on the solution algorithms employed. It is important to recall that "large" in the stochastic context is never as large as in the qualitative model checking, since, unless approximate techniques are applied, the limit in size is given by the size and the number of vectors required by the solution process. In this paper we have considered systems with up to 10 millions states.

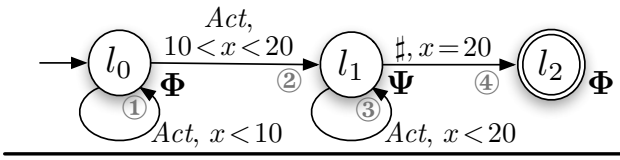
CSL and Prism. Prism allows to model check CSL formula in a very efficient way. The most complex operator, apart from steady state, is the evaluation of the probability of the set of paths that verify an Until formula of the type $\mathcal{P}_{\bowtie\lambda}(\Phi\ \mathcal{U}^{[t,t']}\Psi)$ for a CTMC $\mathcal{M}$, where Φ and Ψ are boolean functions over the set AP of atomic propositions associated to the states of $\mathcal{M}$, $\bowtie \in \{<, \leq, \geq, >\}$ is a comparison operator, $\lambda \in \mathbb{R}_{[0,1]}$ is interpreted as a probability, and $0 \leq t \leq t'$ is a time interval. The probability of the paths that satisfy the Until formula can be computed by the (transient) solution of one or two CTMCs (depending on the time interval $[t, t']$). These CTMCs are derived from the original model $\mathcal{M}$ by making certain states absorbing, and we shall term $\mathcal{M}[\Phi]$ the CTMC obtained from $\mathcal{M}$ by making all the states that satisfy Φ in $\mathcal{M}$ absorbing. When $t \neq t'$ the model checking algorithm requires the transient solution of two modified CTMCs: the chain $\boldsymbol{\pi}^{\mathcal{M}[\neg\Phi]}$ is solved for time t, assuming we start in s at time 0, the resulting probability vector is then used as initial probability for the solution at time $t' - t$ of the chain $\boldsymbol{\pi}^{\mathcal{M}[\neg\Phi\vee\Psi]}$, where the result of the first computation are filtered out to put to zero the probability of all states which are not Φ states. The elements of the second transient solution vector that satisfy Ψ are then summed-up to obtain the probability of the set of paths starting from the initial state s and that satisfy the Until. A comparison with λ allows to define whether s satisfies the formula or not.

Prism allows also an hybrid solution engine, in which the CTMC, and the modified CTMCs required in the computation, are stored efficiently using decision diagrams, while the solution vector is stored in full. Moreover, although the above description is fully forward, from time 0 to time t', the model checking works backward (as explained later) from the states that satisfy Ψ to the set of states that satisfies the full formula. This allows to compute, through two transient solutions only, the full set of states that satisfy the formula. Other model checkers like Marcie and MRMC apply the same solution approach.

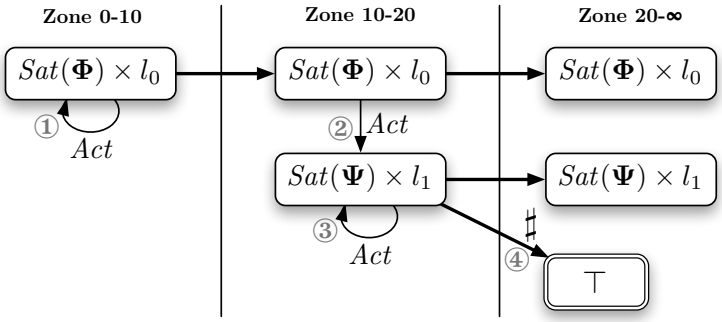
$\mathrm{CSL}^{\mathrm{TA}}$ *and MC4*$\mathrm{CSL}^{\mathrm{TA}}$*, version 1* $\mathrm{CSL}^{\mathrm{TA}}$ (single clock) uses timed automata (TA) to specify (timed) accepted path and the model checker goal is to compute the probability of the set of accepted paths. To avoid the introduction of nondeterminism the TA is required to be "deterministic" (DTA): for each path in the automaton there is at most one path in the TA that accepts it. A DTA $\mathcal{A}$ is made of a set of *locations* and a set of *edges*. Each DTA is equipped with a *clock*, usually named x, that runs constantly and whose value increases linearly over time. Edges describe the transition relation and can be labeled with a *clock constraint*. The DTA of Figure 1(A) has three locations l_0, l_1, l_2. Location l_0 is initial, and l_2 is final. An edge with a constraint in the form $x = c_1$ is a *Boundary* edge (marked with a ♯), and is triggered by the elapse of time. An edge with a constraint $c_1 < x < c_2$ is an *Inner* edge (as the l_0, l_1) edge) and is triggered by a transition firing in the GSPN (or by a transition in the CTMC). Each edge can have an associated reset of the clock x. *Inner* edges can have an associated set of actions (transition names of a GSPN or action names of a decorated CTMC), and locations can have an associated boolean formula (the atomic propositions Φ and Ψ in the example). With reference to GSPN, we can

Fig. 1. An example of DTA.

(A) A simple DTA that describes which CTMC paths are accepted.



(B) State space of the cross product of any CTMC with the DTA (A).



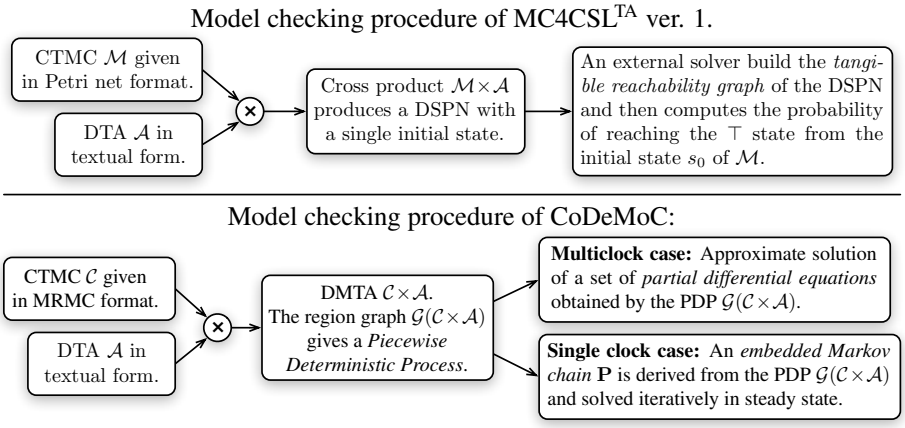
say that a transition in the underlying CTMC from marking m to marking m' due to the firing of transition a is accepted by the DTA in location l through the edge (l, l') if, assuming that m satisfies the boolean condition associated to l, the transition a is in the set of actions associated to the (l, l') edge, the current value of x satisfies the edge constraint, and m' satisfies the boolean condition of l'. The DTA of the example accepts CTMC paths that stay in a Φ state for at least 10 unit time, then moves to a Ψ state at any time between 10 and 20, with any CTMC move in the *Act* set, and at time 20 is found in a state that satisfies Φ. Note that all edges in the DTA are *Inner* edges, but the one between l_1 and l_2. For each DTA is possible to define $C = \{c_i\}$, the ordered set of clock values that label $\mathcal{A}$ clock constraints, with the addition of 0 and ∞. For the example in Figure 1(A), $C = \{0, 10, 20, \infty\}$ A state of a (D)TA is then given by a pair (l, c) where l is a location and c is a clock value in C.

CSL$^{\text{TA}}$ is a variation of CSL in which the $\mathcal{P}_{\bowtie\lambda}(\varphi)$ operator (with φ being either a timed neXt or a a timed Until operator) is substituted by a $\mathcal{P}_{\bowtie\lambda}(\mathcal{A})$. CSL$^{\text{TA}}$ is more expressive than CSL [8], and this comes at the price of a more complex model checking algorithm: verifying a formula requires the steady state solution of an (absorbing) Markov Regenerative Process (MRP) obtained as the cross-product of the Markov chain with the DTA. If s is the state of a CTMC and (l, c) is the state of the DTA, a state in the cross-product is the triple (s, l, c), or one of the two states $\top$ or $\bot$ The cross-product is built in such a way that all and only the paths of the CTMC that take the DTA to a final location end up in a $\top$ state. A state s of a CTMC $\mathcal{M}$ satisfies the formula $\mathcal{P}_{\bowtie\lambda}(A)$ if in the cross-product MRP $\mathcal{M}\times\mathcal{A}$ the probability of reaching $\top$ from (s, l_0, c_0) is

$\bowtie \lambda$. Fig. 1(B) shows the general cross-product induced by the DTA (A) on any CTMC, where the rectangles are set of CTMC states that satisfies the DTA's state propositions.

As depicted in the upper part of Figure 2, model checking of CSL$^{\text{TA}}$ requires two steps: building the MRP $\mathcal{M}\times\mathcal{A}$ and then solving it. In the first version of MC4CSL$^{\text{TA}}$ the cross product algorithm produces a DSPN whose underlying process is isomorphic to the $\mathcal{M}\times\mathcal{A}$, so that the solution step can be left to existing DSPN tools. This approach is inspired by software reuse, but it is highly inefficient, since even the starting CTMC has to be translated into a DSPN, moreover the use of existing tools, not specifically designed for model-checking, allows to use only the much less efficient forward approach.

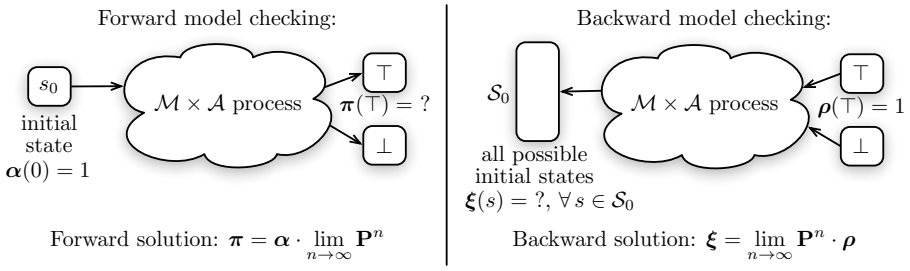
Fig. 2. Working structures of the MC4CSL$^{\text{TA}}$ and Codemoc model checkers.



MRP solution methods There is a large degree of variation in the solution approaches for MRP. When applied to CSL$^{\text{TA}}$ the classical approach builds the embedded Markov chain **P** and solves it to compute the probability of the renewal state $\top$. This approach suffers from the *fill-in* approach: P is usually a very dense matrix and only small states spaces can be solved. A matrix-free approach has been defined in [14], which computes the probability of renewal states without ever building and storing P. This approach has been extended in [15] to deal with non ergodic MRPs, as required by CSL$^{\text{TA}}$. More recently a component-based approach [2] has been defined for non-ergodic MRPs, which can significantly reduce the space and time complexity of model-checking CSL$^{\text{TA}}$. In particular it was shown in the same paper that the algorithm, when applied for DTAs that are Until formulas, reduces to the computation of the transient solution of two CTMC, although the space complexity is not the same since the $\mathcal{M}\times\mathcal{A}$ includes both the two CTMCs that have to be solved, while a CSL model checker can build and solve the one at a time.

CSL$^{\text{TA}}$ *and Codemoc* The work in [10] considers the model checking of paths specified by DTAs with multiple clocks. It actually changes also the semantics of how the DTA reads a path in the CTMC, so, even for the single clock case it might not be trivial to specify a CSL$^{\text{TA}}$ properties using the DTAs in [10]. The lower part of Figure 2 shows the algorithm used in the Codemoc tool [11] to model check CSL$^{\text{TA}}$ with multiple clocks: the cross product is built and then a *region graph* (a classical construction in multi-clock timed automata) is computed, which identifies a *Piece-wise Deterministic process*, that is then solved through the numerical solution of a set of differential partial equations. Codemoc has a specific procedure for the case of single clock DTAs (since in this case the stochastic process reduces to an MRP) , which builds and solves the embedded DTMC of the MRP. This last solution does not work very well and we could not use it in our comparison part.

Fig. 3. Forward and backward model checking.



Forward vs, backward approaches. Forward and backward model checking refers to the two different ways of formulating the system of linear equations to compute the $\mathcal{P}()$ operator. Figure 3 shows the different approaches and the solution equations. The forward method starts with the probability vector α at time 0, and computes the limiting probability $\boldsymbol{\pi}$ of reaching the $\top$ state. Backward probability instead starts with a reward $\boldsymbol{\rho}$ of 1 in the $\top$ state in the long run, and computes, for each state, the probability of reaching the $\top$ state, at the same cost as the computation of the forward probability from a *single* initial state. Note that, despite its name, even in the backward approach the $\mathcal{M}\times\mathcal{A}$ state space is built forward, starting from one or more initial states, and it is only the numerical solution that works backward.

Overdimensioning of the state space In some cases, the $\mathcal{M}\times\mathcal{A}$ process contains more states than it is needed. This is very clearly indicated by considering the cross-product between a generic CTMC $\mathcal{M}$ and the DTA of Figure 1(A) depicted in (B). The $\mathcal{M}\times\mathcal{A}$ is represented in compact form (putting together all states with the same (l, c) pair. It is clear from the picture that the two sets of states in the rightmost zone $(20, \infty)$ are useless, since the objective is to compute the probability of reaching $\top$, a computation that can be correctly performed even if the two sets are substituted by a single $\perp$ state. Since any of the two sets can

be as big as the whole state space, the substitution with a single state can be particularly interesting.

3 The zoned-DTA technique

To avoid the construction of non useful states in the cross-product we propose to expand the DTA automaton $\mathcal{A}$ into its *zoned transition system* (ZDTA) $\mathcal{Z}(\mathcal{A})$, where each state is a pair (location, clock zone). This new structure is then analyzed to collapse into a single $\perp$ state each pair for which there is no path that leads to an accepting location, before building the cross-product $\mathcal{M}\times\mathcal{Z}(\mathcal{A})$.

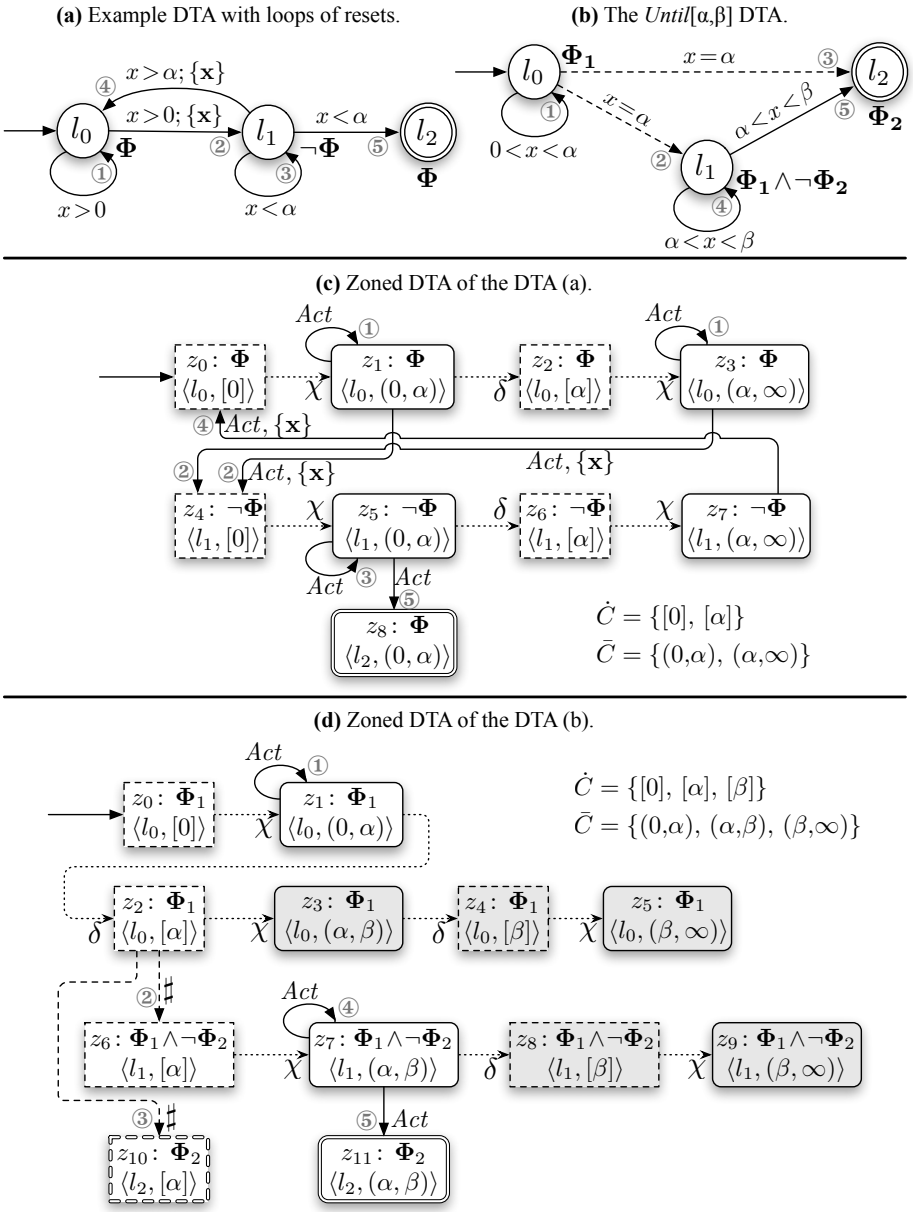
Zoned DTA. Let us recall that C is the ordered set of clock values that label the $\mathcal{A}$ clock constraints, with the addition of 0 and ∞, and we write $C = \{c_0, c_1, \ldots, c_m\}$, with $c_0 = 0$, $c_{i+1} > c_i \ \forall i \in [0, m)$ and $c_m = \infty$. Then two clock values $a, b \in \mathcal{R}_{\geq 0}$ are in the same *equivalence class* if, for all edges e, the evaluation of the clock constraint of e is unchanged. A zone automaton $\mathcal{R}(\mathcal{A})$ records the smallest set of *equivalence classes* of clock values, denoted as *zones*. Since $\mathcal{A}$ has a single clock x, classes in $\mathcal{R}(\mathcal{A})$ have form $[x = c]$ or $(c < x < c')$, for all the values $c \in C$. Therefore, the construction of $\mathcal{R}(\mathcal{A})$ is a straightforward partitioning of $\mathbb{R}_{\geq 0}$, as in [9]. From the above we can build a Zoned DTA $\mathcal{Z}(\mathcal{A})$ for any DTA $\mathcal{A}$, in which the locations of $\mathcal{A}$ are paired with the clock zones. We first define the set of *immediate zones* $\dot{C}$ and the set of *timed zones* $\overline{C}$.

$$\dot{C} \stackrel{\text{def}}{=} \Big\{[c] \mid c \in C\Big\} \qquad \text{and} \qquad \overline{C} \stackrel{\text{def}}{=} \Big\{\big(c, \mathsf{next}(c)\big) \mid c \in C\Big\}$$

Starting from the initial location $(l_0, [c_0])$ we can generate all possible reachable pairs $(l_0, [c])$, or $\big(l, (c, \mathsf{next}(c))\big)$ through a set of rules that can be found in [13].

Figure 4 illustrates the zoned DTAs of two sample DTAs. Each location in (c) and (d) reports the location $z \in Z$, the state proposition of l (that holds also in each $z = \langle l, c\rangle$), and, on the second line, the DTA location and the clock zone. Immediate and timed locations are drawn with a dotted and a solid border, respectively, while final locations have a double border. The set of locations that cannot reach a final location are colored in gray. Edges are marked as χ if they are generated from a *Boundary* edge of the DTA, δ (let time elapse) otherwise. The timed reachability of some locations (for instance z_8 and z_9 in (d)) represents an information that is not directly available in the DTA $\mathcal{A}$. These locations are irrelevant for the computation of the path probability, and can be discarded, since they will never reach a final location. Observe also that the construction of (d) could be modified to avoid the construction of the edge $z_2 \xhookrightarrow{\delta} z_3$: indeed the *Boundary* edges ② and ③ in the DTA have priority over the ① edge and the process will take for sure one of the first two edges, since the logic condition for remaining in l_0 in $[\alpha]$ is: $\Phi_1 \wedge \neg\big(\Phi_2 \vee (\Phi_1 \wedge \neg\Phi_2)\big)$ which always evaluates to false, for any CTMC. If z_3 is unreachable, also z_4 and z_5 are so they could be removed. This condition can be evaluated for any χ edge, to remove those locations that

Fig. 4. Two sample DTAs with their associated zoned DTAs.

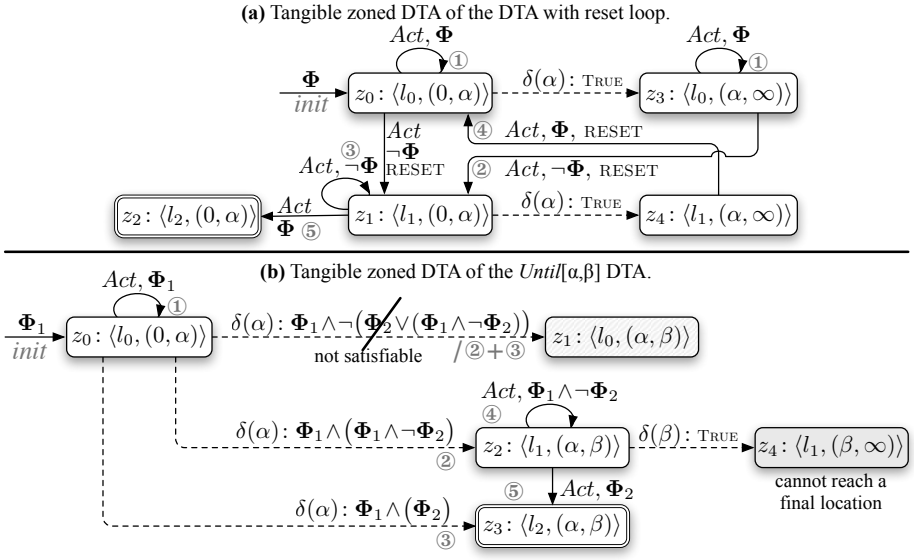


are logically unreachable. Each location $\langle l, c\rangle$ of Fig. 4(c,d) is labeled with the state proposition expressions of l. The presence of immediate zones can make the construction of the $\mathcal{M}\times\mathcal{Z}(\mathcal{A})$ process more complex and we prefer to define the concept of *tangible zoned DTA*, where only timed locations are kept, and bound-

ary locations are collapsed with a transitive closure. The firing of a sequence of DTA *Boundary* edges $l_0 \xrightarrow{\gamma_1, \sharp, r_1} l_1 \xrightarrow{\gamma_2, \sharp, r_2} \dots \xrightarrow{\gamma_n, \sharp, r_n} l_n$ may happen only if all the state proposition expressions $\Lambda(l_0), \Lambda(l_1), \dots, \Lambda(l_n)$ are satisfied by the destination CTMC state s'. A transitive closure of *Boundary* firings is more easily expressed by moving the state proposition onto the edge, which give rise to the Tangible Zoned DTA $\mathcal{T}(\mathcal{A})$ of $\mathcal{A}$.

A TZDTA edge $(\mathrm{z}, \mathrm{z}')$ has a logical condition λ which is the logical *and* of satisfying the destination location condition $\Lambda(\mathrm{z}')$, as well as all the intermediate location conditions $\Lambda(\dot{z}_i), 1 \leq i \leq n$, and in the last immediate location every other *Boundary* edge must not be satisfied. Given $\mathcal{Z}(\mathcal{A})$, the corresponding $\mathcal{T}(\mathcal{A})$ is constructed by taking all the timed locations and *Inner* edges, and by applying the closure rule on all *Boundary* edges. The ZDTA edges are not marked as either δ or χ since all edges are from tangible to tangible locations. Figure 5

Fig. 5. Tangible zoned DTA of the two DTAs of Fig. 4.



shows the *tangible* closure of the two ZDTA of Fig. 4. Boundary edges are all collapsed into $\boldsymbol{\delta}$ edges, which are labeled with a state proposition expression that is the transitive closure of all the s.p.e. that must be satisfied to follow that edge. In the tangible ZDTA of the *Until*$[t, t']$, location z_1 is unreachable because the condition associated to the edge is false. Location z_4 is irrelevant for the computation of the probability of reaching a final state, since any path that reaches this location will certainly be rejected. The advantage of collapsing the state proposition expression of the closure of *Boundary* edges is that it becomes clear whether an edge has an unsatisfiable condition. Each edge is also labelled with the sequence of DTA edges that represents (with circled numbers), and

which DTA edges are not satified by the transitive closure (written after a '/'). The structure of Fig. 5(b) shows that there are at most three tangible zones for an $Until[\alpha, \beta]$, while the other two zones can be discarded. This allows to optimize the $\mathcal{M}\times\mathcal{A}$ cross product, by removing irrelevant states in advance.

4 The MC4CSL$^{\mathrm{TA}}$ tool, version 2: features and assessment

The MC4CSL$^{\mathrm{TA}}$ tool, version 1, presented in [1], based on the theoretical results defined in [16], was meant as a prototype implementation to show the feasibility of model-checking CSL$^{\mathrm{TA}}$, but it had many drawbacks that make it an unpractical tool to use even on small to medium size examples (around ten thousand states). The main problem was the use of DSPN, as explained above, and the limited set of numerical methods available for matrix-free solution of DSPN solver (as the explicit MRP solution method is never a realistic option). The dependency from DSPN has been solved by implementing directly the $\mathcal{M}\times\mathcal{A}$ construction, which leads to an MRP for which several solution techniques can then be applied, techniques that implement the theoretical advancements in [15] and [2]. The backward solution approach has been implemented for both the matrix-free approach (which is rather straightforward despite the fact that the embedded DTMC $\mathbf{P}$ is never built or stored) and the component-based method (which can be less intuitive). A full discussion of the topic and the precise formulation of the backward solution process can be found in [13], and it is implemented in version 2. Another issue that has been solved in version 2 is the presence of significant number of states in the $\mathcal{M}\times\mathcal{A}$ process that never lead to the $\top$ state, since the implementation now is based on a cross product of the Markov chain with the tangible zoned DTA.

Fig. 6. Structure of MC4CSL$^{\mathrm{TA}}$ version 2.

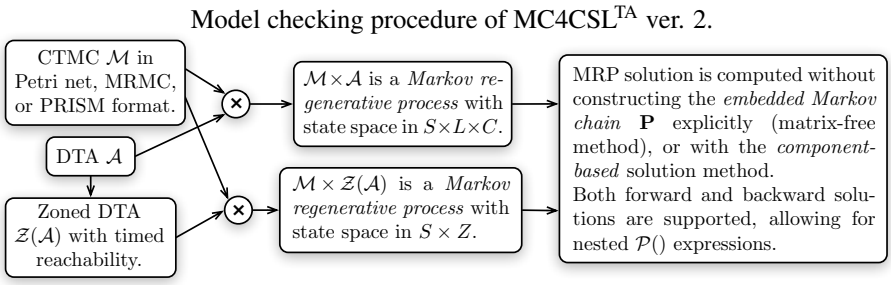


Figure 6 shows the structure of MC4CSL$^{\mathrm{TA}}$ version 2, available through [17]. The tool takes in input a model, which can be either a Generalized Stochastic Petri Net in GreatSPN [18] format, or a CTMC in MRMC/Prism format, and a formula specification, which consists of a single clock DTA in textual form.

When the input language is a Petri net, the atomic proposition associated with the locations of the timed automaton are expressions over the Petri net marking, while the actions associated to the edges are sets of transitions' names. There are two ways of generating the underlying MRP, according to the two different ways of computing the cross-product: either as $\mathcal{M}\times\mathcal{A}$ or as $\mathcal{M}\times\mathcal{Z}(\mathcal{A})$. We now evaluate the tool correctness and efficiency against Prism and Cosmos. No comparison with version 1 is reported since it would hardly solve the first instances of the proposed models.

Cell cycle control. This first test considers a probabilistic model of the cell replication control in eukaryotes. This biological model is taken from [19], and originally specified in [20]. This model describes the molecular machinery used by eukaryotic cells in order to control their replication. The control mechanism is made by an antagonistic interaction between two proteins, CDK and APC, the first extinguishing the activity of the second and viceversa. The cell replication cycle is controlled by the binding of CDK with its activator cyclin. The state of the model is described by the quantities of the proteins involved in the biochemical interaction, and transitions represent the reactions. The tool directly imports the CTMC produced by Prism.

Table 1 shows three CSL queries asking for the probability of having all the CDK proteins bound by their cyclin activator in a given time window - where N is the quantity of CDK proteins in the system. For the first and second queries, the probability is set at time 10 and in the time interval (10-20). In the third case, the time interval is (0-5), with the condition that the initial state must have a probability of having all the CDK molecules bounded within 1 second.

The table shows the overall model checking time of both tools. For Prism, both the hybrid (default) engine and the sparse engine are used. For MC4CSL$^{\mathrm{TA}}$ the timings for the explicit, matrix-free and component-based (SCC) methods are shown. The data reflect the theoretical result of [2], which ensures that the sparse engine of Prism and the SCC method have the same asymptotical cost. The Table also reports, for the (A) and (B) cases, the state space of the MRP produced using the DTA $\mathcal{A}$ or the tangible zoned DTA $\mathcal{T}(\mathcal{A})$ introduced in this paper, which shows the advantage of the method. The time reported are for the tangible ZDTA case. In all the tests, Prism performs better than MC4CSL$^{\mathrm{TA}}$, which is not surprising since the CSL model checking algorithm works with a predefined structure of the formulas and requires fewer steps than that of CSL$^{\mathrm{TA}}$. All tests were run on a Xeon 2.13 GHz single-core of a multicore machine with $128G$ bytes of available memory.

Workflow model. In this second sample we compare the MC4CSL$^{\mathrm{TA}}$ tool against the simulator Cosmos [4], which has an input modeling language that is a superset of CSL$^{\mathrm{TA}}$ DTAs [12]. Figure 7 shows the (Generalized Stochastic) Petri net of the model [21] which describes an order-handling company. The net illustrates the flow of an order, which involves two separate tasks: preparing and sending the bill to the client, and to ship the requested goods. The company reckons on three types of employees: those who manage accounting (F), logistics (L) and generic employees (E). Different tasks are carried out by different

(A) CSL Until with a single time interval. Durations are expressed in seconds.

			Prism 4.1		MC4CSL^{TA}				
N	**States**	**Trns.**	**hybrid**	**sparse**	**explicit** (1 smc)	**matrix-free**	**SCC** (1 comp)	$\mathcal{M}\times\mathcal{A}$ (no zdta)	$\mathcal{M}\times\mathcal{T}(\mathcal{A})$ (zdta)
2	4666	18342	0.1	0.1	0.1	0.2	0.1	8524	4668
3	57667	305502	1.5	0.7	3.2	11.6	2.8	109148	57667
4	431101	2742012	37.3	14.0	39.7	157.9	38.0	830119	431103
5	2326666	16778785	277.8	144.6	306.3	1277.3	307.7	4525426	2326668
6	9960861	78768799	nc	nc	2267.5	9108.0	2050.9	19495025	9960863

CSL: P=? [true U[10,10] *cyclin_bound*=N]

CSL^{TA}: PROB=? until_AA (10 | | True, (#*cyclin_bound*=N))

(B) CSL Until with (t, t') time interval.

			Prism 4.1		MC4CSL^{TA}					
N	**States**	**Trns.**	**hybrid**	**sparse**	**explicit**	**smc**	**matrix-free**	**SCC** (2 comp)	$\mathcal{M}\times\mathcal{A}$ (no zdta)	$\mathcal{M}\times\mathcal{T}(\mathcal{A})$ (zdta)
2	4666	18342	0.5	0.1	78.61	3827	0.6	0.1	12380	8524
3	57667	305502	2.8	1.5	~29 hours	51394	26.0	5.7	160627	109148
4	431101	2742012	54.3	26.0	-	-	354.3	73.0	1229135	830119
5	2326666	16778785	502.2	234.5	-	-	3814.4	690.6	6724184	4525426
6	9960861	78768799	nc	nc	-	-	21004.2	4646.8	29029187	19495025

CSL: P=? [true U[10,20] *cyclin_bound*=N]

CSL^{TA}: PROB=? until_AB (10, 20 | | True, (#*cyclin_bound*=N))

(A) Nested CSL query.

			Prism 4.1		MC4CSL^{TA}	
N	**States**	**Trns.**	**hybrid**	**sparse**	**matrix-free**	**SCC**
2	4666	18342	0.1	0.1	0.1	0.1
3	57667	305502	0.7	0.5	5.7	2.0
4	431101	2742012	17.2	9.6	79.6	25.1
5	2326666	16778785	159.1	78.3	608.9	192.0
6	9960861	78768799	nc	nc	3356.6	1123.6

CSL: P=? [P>0.5 [true U<1 *cdk_cat*=2] U<5 *cyclin_bound*=2]

CSL^{TA}: PROB=? until_0B(5 | | PROB>0.5 until_0B (1 | | True, #*cdk_cat*=2), #*cyclin_bound*=2)

Table 1. Performance comparison of Prism 4.1 and MC4CSL^{TA}.

employees. The Petri net is made of some subnets consisting of an immediate transition (thin bar), a place and an exponentially distributed timed transition (white box). Such subnets first allocate one of these staff resources, execute the specified task and then release the resource. The staff is represented by three places *finance*, *logistics* and *employees*. Arrows from and to these three places are drawn only for the case of the activity represented by the *register_E* transition, and omitted in the picture for the other subnets whose transitions have labels with suffixes "_E", "_F" and "_L".

Fig. 7. Petri net of the workflow model.

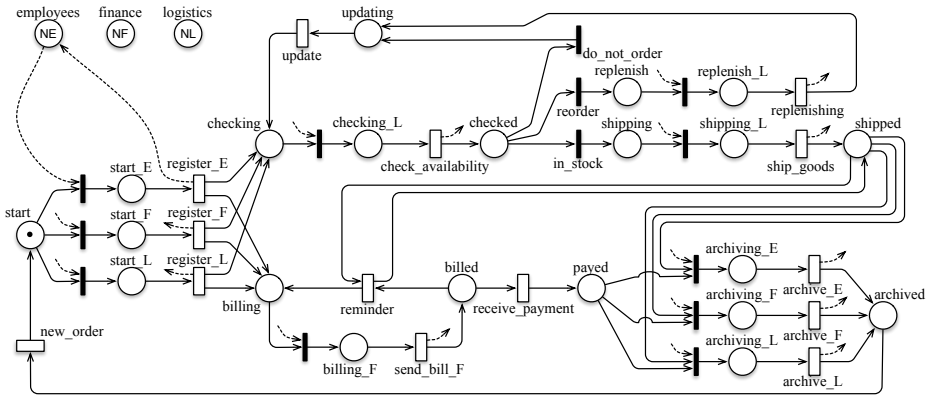
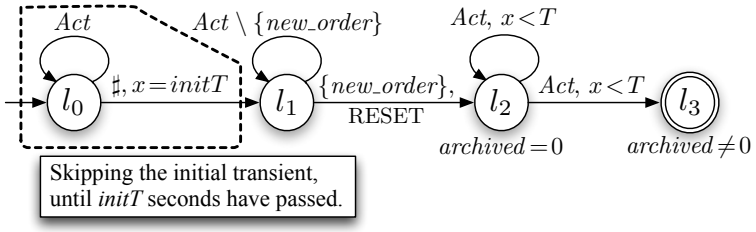


Fig. 8. Property tested in the workflow model with Cosmos and MC4CSL$^{\mathrm{TA}}$.



The DTA of the measured property is depicted in figure 8. A path starts in the initial state and skips $initT$ time units, as an initial transient. Then the DTA waits for the arrival of a *new_order* event, which signals the beginning of the ordering cycle. The path is accepted if the order is *archived* in less than T time units. This DTA can be converted in the input language of Cosmos, allowing for a cross validation of the MC4CSL$^{\mathrm{TA}}$ tool for non-CSL queries.

			Cosmos				**MC4CSL$^{\mathrm{TA}}$**	
			width=0.001		**width=0.0001**		**matrix-free**	**SCC**
N	**States**	**Trns**	**paths**	**Time**	**paths**	**Time**	**MC**	**MC**
1	44	81	7000	19.95	63000	58.68	0.75	0.02
2	1811	6408	7000	26.25	62000	177.58	330.64	2.12
3	68942	349980	7000	45.64	67000	413.19	-	210.23
4	2440192	15827904	8000	91.03	75000	669.82	-	~5 hours
5	81M	633M	9000	108.53	85000	1072.15	-	-

Table 2. Performance comparison of Cosmos and MC4CSL$^{\mathrm{TA}}$.

The probabilities computed with MC4CSL^{TA} are in accordance with that computed with Cosmos available in [22]. The comparison with Cosmos, on this and on other models, proved to be very useful in detecting errors in MC4CSL^{TA}. Table 2 shows a performance comparison of the simulator Cosmos with the numerical solution of MC4CSL^{TA}, with the timings set to $initT = 100$ and $T = 50$. Simulations were run at a 99% of precision with the confidence interval width reported in the Table. As expected, simulator scales better for large state spaces. The chosen timings of the GSPN transitions and of the DTA queries have been chosen so as to require very long uniformization sequences, thus putting MC4CSL^{TA} in its worst possible conditions, a case in which the advantage of the component-based solution over the matrix-free one is very evident. Tests were run on a Intel core Duo 2.4GHz with $4G$ bytes of memory.

5 Conclusion

This paper presents the new version of the CSL^{TA} model checker MC4CSL^{TA}, which represents a total innovation with respect to the previous version, since it includes a new solution approach which builds on some recently published results on MRP solution and on the construction of a zoned DTA, presented in this same paper. The tool has been evaluated for correctness and performance against the well-known CSL model checker Prism (on the subset of the DTA which can be equivalently expressed as a CSL property) and with the statistical model checker Cosmos, for whose formulas that go beyond CSL. The reported tests, as well as some other tests reported in [13], suggest that the new version of MC4CSL^{TA} is a mature tool, able to deal with very large state spaces, where, again, large is intended as "large for being a stochastic process".

The construction of the zoned DTA will be the basis for our future work on the tool. In particular the application of the component-based method for MRPs, paired with the analysis of the ZDTA, can lead to an on-the-fly implementation of the tool: the state space is built component by component, only when it is actually needed for the computation of the probability of the success state $\top$.

References

1. Amparore, E.G., Donatelli, S.: MC4CSL^{TA}: an efficient model checking tool for CSL^{TA}. In: International Conference on Quantitative Evaluation of Systems, Los Alamitos, CA, USA, IEEE Computer Society (2010) 153–154
2. Amparore, E.G., Donatelli, S.: A component-based solution for reducible markov regenerative processes. Performance Evaluation **70** (2013) 400 – 422
3. Kwiatkowska, M., Norman, G., Parker, D.: PRISM: Probabilistic Model Checking for Performance and Reliability Analysis. Performance Evaluation **36** (2009) 40–45
4. Ballarini, P., Djafri, H., Duflot, M., Haddad, S., Pekergin, N.: 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), Aachen, Germany, IEEE Computer Society Press (2011) 143–144

5. Aziz, A., Sanwal, K., Singhal, V., Brayton, R.: Model-checking continuous-time Markov chains. ACM Transactions on Computational Logic **1** (2000) 162–170
6. Katoen, J.P., Zapreev, I.S., Hahn, E.M., Hermanns, H., Jansen, D.N.: The ins and outs of the probabilistic model checker MRMC. Performance Evaluation **68** (2011) 90–104
7. Schwarick, M., Heiner, M., Rohr, C.: Marcie - model checking and reachability analysis done efficiently. In: Quantitative Evaluation of Systems (QEST), 2011 Eighth International Conference on. (2011) 91–100
8. Donatelli, S., Haddad, S., Sproston, J.: Model checking timed and stochastic properties with CSL^{TA}. IEEE Transactions on Software Engineering **35** (2009) 224–240
9. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Comp. Science **126** (1994) 183–235
10. Chen, T., Han, T., Katoen, J.P., Mereacre, A.: Model checking of continuous-time Markov chains against timed automata specifications. Logical Methods in Computer Science (2011) **7** (2011)
11. Barbot, B., Chen, T., Han, T., Katoen, J.P., Mereacre, A.: Efficient ctmc model checking of linear real-time objectives. In: TACAS'11. (2011) 128–142
12. Ballarini, P., Djafri, H., Duflot, M., Haddad, S., Pekergin, N.: HASL: An expressive language for statistical verification of stochastic models. In: Proceedings of the 5th International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS'11), Cachan, France (2011) 306–315
13. Amparore, E.G.: States, actions and path properties in Markov chains. PhD thesis, University of Torino, Italy (2013)
14. German, R.: Iterative analysis of Markov regenerative models. Performance Evaluation **44** (2001) 51–72
15. Amparore, E.G., Donatelli, S.: Revisiting the Iterative Solution of Markov Regenerative Processes. Numerical Linear Algebra with Applications, Special Issue on Numerical Solutions of Markov Chains **18** (2011) 1067–1083
16. Amparore, E.G., Donatelli, S.: Model Checking CSL^{TA} with Deterministic and Stochastic Petri Nets. In: Proceedings of the 2010 IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), IEEE Computer Society Press (2010) DSN-PDS 2010.
17. Amparore, E.G., Donatelli, S.: The MC4CSL^{TA} model checker. http://www.di.unito.it/~amparore/mc4cslta/ (2013)
18. Baarir, S., Beccuti, M., Cerotti, D., Pierro, M.D., Donatelli, S., Franceschinis, G.: The GreatSPN tool: recent enhancements. SIGMETRICS Performance Evaluation Review **36** (2009) 4–9
19. Lecca, Priami: Cell Cycle Control in Eukaryotes - Prism case studies. http://www.prismmodelchecker.org/casestudies/cyclin.php (2011)
20. Lecca, P., Priami, C.: Cell cycle control in eukaryotes: A BioSpi model. In: Proc. Workshop on Concurrent Models in Molecular Biology (BioConcur'03). Electronic Notes in Theoretical Computer Science (2003)
21. Aalst, W.: Business process management demystified: A tutorial on models, systems and standards for workflow management. In: Lectures on Concurrency and Petri Nets. Volume 3098 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2004) 1–65
22. Amparore, E.G., Ballarini, P., Beccuti, M., Donatelli, S., Francheschinis, G.: Expressing and Computing Passage Time Measures of GSPN models with HASL. In: 34th International Conference on Application and Theory of Petri Nets and Concurrency, Milano, Italy, LNCS 7927, Springer Berlin Heidelberg (2013)