Model Checking CSL^{TA} with Deterministic and Stochastic Petri Nets

Elvio Gilberto Amparore and Susanna Donatelli
Dipartimento di Informatica,
Università di Torino, Italy
{amparore.elvio, susi}@di.unito.it

Abstract

CSL^{TA} is a stochastic temporal logic for continuous-time Markov chains (CTMC), that can verify the probability of following paths specified by a Deterministic Timed Automaton (DTA). A DTA expresses both logic and time constraints over a CTMC path, yielding to a very flexible way of describing performance and dependability properties. This paper explores a model checking algorithm for CSL^{TA} based on the translation into a Deterministic and Stochastic Petri Net (DSPN). The algorithm has been implemented in a simple Model Checker prototype, that relies on existing DSPN solvers to do the actual numerical computations.

Keywords: Stochastic Model Checking, DSPN.

1. Introduction

Continuous time Markov chains (CTMC) are used in many applications that consider stochastic systems, ranging from queuing networks to biochemical processes. CTMCs are analyzed for transient and steady state probabilities, possibly enriched with transitions and state rewards. In order to verify more complex measures, for example to consider only a subsets of all possible CTMC behaviours, some ingenuity is required, that usually results in ad-hoc modifications of the CTMC (for example creating absorbing states) or in the definition of new transitions whose average rewards correspond to the measure of interest. In the last years, stochastic temporal logics have been established as an effective language for user-defined stochastic properties, in particular to describe dependability properties [8] as well as user-defined properties for non-ergodic CTMCs, as often arise when considering systems with failures. A typical example of a formula ϕ is: *does the system have a probability greater than α of reaching a goal state in t seconds from the initial state s_0, without passing through any repair state?*

Given a system property expressed as a stochastic temporal logic formula Φ, a stochastic model checker answers the question: is property Φ true for CTMC $\mathcal{M}$? Stochastic model checkers are built on classical steady state and transient solutions of CTMCs. Indeed it is also possible to determine if Φ is true or not for $\mathcal{M}$ by using a manually modified CTMC, but the main advantage of a stochastic model checkers is that the process is fully automatic, and the user can focus his/her attention on the semantics of the property more than on how to compute it.

Many stochastic logics have been proposed, as CSL[6], asCSL[5], CSL^{TA}[1] and other variants. The CSL formula $\mathcal{P}_{>0.95}(ok\ \mathcal{U}^{[0,50]}\ goal)$ is true for a CTMC if there is at least a 95% probability of reaching a *goal*-state in the next 50 seconds, going through *ok*-states only. Labels like *goal* or *ok* are attached to CTMC states. CSL has two operators to describe paths, namely *Until* ($\Phi\ \mathcal{U}^{[\alpha,\beta]}\Psi$) and *Next* ($\mathcal{X}^{[\alpha,\beta]}\Phi$). A limitation of CSL is that *path-formulas* are not very combinable: the following is not a CSL formula:

$$\mathcal{P}_{>0.95}(\Phi_1\ \mathcal{U}^{[\alpha,\beta]}\Phi_2\ \mathcal{U}^{[\gamma,\delta]}\Phi_3)$$

because in $\Phi\ \mathcal{U}^{[\alpha,\beta]}\Psi$ both Φ and Ψ have to be state formulas. See the work in [1] for a throughout discussion of this CSL limitation and for the relevance of concatenated timed intervals in path formulas.

To lift these limitations, variations of CSL have been proposed: asCSL [5] describes paths with *regular-expressions*, CSL^{TA} [1] uses *single-clock timed automata* [12], while the work in [9] uses *multiple-clocks timed automata*.

In asCSL paths can be specified in terms of actions and states, with a single time interval along each path [5, Sec. 3]. Path formulas are specified as $\mathcal{P}_{\bowtie\lambda}(\eta^{[\alpha,\beta]})$, where η is a regular espression over state properties and action names.

CSL^{TA} allows for state and action specification of paths, and for concatenated timed interval. Path formulas are specified as $\mathcal{P}_{\bowtie\alpha}(\mathcal{A})$, where $\mathcal{A}$ is a *single clock timed automata*.

In [9] properties are specified using an automata with an arbitrary number of clocks, which allows for arbitrary nesting of timed intervals.

These extended logics have different expressivity: a greater expressivity of the logic carries a greater complexity of the model checking algorithm. CSL and asCSL model checkers are built on the transient/stationary evaluation of a

(set of) modified CTMC(s). CSL$^{\text{TA}}$ requires the solution of a stochastic process that it is not a CTMC, but a more general stochastic process called *Markov Regenerative Process*[3]. If more than one clock is allowed, the underlying stochastic process is a Piece-wise Deterministic Process (PDP), and the verification of a property requires the generation and solution of a set of ordinary differential equations in the size of the PDP locations (and not in the size of the CTMC). There are some good CSL model checkers available (as PRISM [10] and MRMC [11]), while none is currently available for asCSL, for CSL$^{\text{TA}}$ and for the case of multiple clocks.

This paper concentrates on CSL$^{\text{TA}}$, and describes the definition and implemention of a model-checker for CSL$^{\text{TA}}$. To the best of our knowledge, no such implementation already exists. This paper proposes a very simple solution to this hurdle: the construction of an intermediate DSPN (a class of Stochastic Petri Nets with a MRP as the underlying stochastic process), solvable with an *existing* DSPN solver. Three DSPN solvers have been considered: TimeNET [7], DSPNexpress [2], and SPNica [3], but only the last two have been linked in our implementation.

Section 2 summarizes CSL$^{\text{TA}}$ and its model checking procedure, as described in [8], revisited to enlight the points on which our model-checker is based. Section 3 explains the procedure to model-check CSL$^{\text{TA}}$ using DSPN, while Section 4 describes the implementation, which is illustrated on a cyclic polling example, and discusses the strong and weak points of the proposed solution (and of the DSPN solvers used). Section 5 concludes the paper with a discussion of future work.

2. CSL$^{\text{TA}}$ syntax and semantics

The stochastic model verified by CSL$^{\text{TA}}$ is a CTMC with action names and state propositions, called ASMC (*Action and State-labeled Markov Chain*).
An ASMC is a tuple $\mathcal{M} =< S, Act, AP, lab, \mathbf{R} >$, where:

- S is a finite set of states,
- Act is a finite set of action labels,
- AP is a finite set of atomic propositions,
- $lab : S \to 2^{AP}$ is a state labeling function,
- $\mathbf{R}$: $S \times Act \times S \to \mathbb{R}_{\geq 0}$ is a rate matrix.

The rate matrix is used to take into consideration self-loops (since they may be labeled with actions). Rate matrices that differ only in the diagonal elements result in the same probability vectors, but may have different execution paths. Atomic propositions refer to the properties of interest observed in the system, such as *working* or *error*. Action names refer to the events that change the system state, like *arrival* or *completion*. A small ASMC is shown in figure 2.

An *infinite path* σ_∞ in $\mathcal{M}$ is an infinite sequence $s_0 \xrightarrow{a_0,\tau_0} s_1 \xrightarrow{a_1,\tau_1} \cdots$ of states alternated with pairs of actions and elapsed times, such that $\mathbf{R}(s_i, a_i, s_{i+1}) > 0$, $\forall i \geq 0$. A *finite path* σ is a finite sequence $s_0 \xrightarrow{a_0,\tau_0} s1 \xrightarrow{a_1,\tau_1} \cdots \xrightarrow{a_{n-1},\tau_{n-1}} s_n$ such that $\mathbf{R}(s_i, a_i, s_{i+1}) > 0, \forall i \in [0, n)$. $Path^{\mathcal{M}}(s)$ denotes the set of infinite paths of $\mathcal{M}$ starting in s. If P is a set of paths, then the notation $Pr_s^{\mathcal{M}}(P)$, refers to the overall probability of following a path $\sigma \in P$.

CSL$^{\text{TA}}$ is a language of logic expressions with two kinds of formulas: *state-formulas* and *path-formulas*. CSL$^{\text{TA}}$ is based on CSL, with the single significant difference that path-formulas are specified with a timed automaton $\mathcal{A}$ rather than with specific operators (like *Until* or *Next*).

Let $\lambda \in [0, 1]$ be a probability, $p \in AP$ an atomic proposition and $\bowtie \in \{\leq, <, >, \geq\}$ a comparison operator. A CSL$^{\text{TA}}$ state formula Φ is defined by:

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

The semantics of CSL$^{\text{TA}}$ in a state $s \in S$ is defined by:

$$\begin{array}{lcl}
\mathcal{M}, s \models p & \Leftrightarrow & p \in lab(S) \\
\mathcal{M}, s \models \neg\Phi & \Leftrightarrow & \mathcal{M}, s \not\models \Phi \\
\mathcal{M}, s \models \Phi_1 \wedge \Phi_2 & \Leftrightarrow & \mathcal{M}, s \models \Phi_1 \wedge \mathcal{M}, s \models \Phi_2 \\
\mathcal{M}, s \models \mathcal{S}_{\bowtie\lambda}(\Phi) & \Leftrightarrow & \sum_{s' \in S \wedge \mathcal{M}, s' \models \Phi} \pi(s, s') \bowtie \lambda \\
\mathcal{M}, s \models \mathcal{P}_{\bowtie\lambda}(\mathcal{A}) & \Leftrightarrow & Pr_s^{\mathcal{M}}\big(AccPath^{\mathcal{M}}(s, \mathcal{A})\big) \bowtie \lambda
\end{array}$$

with $\pi(s, s')$ the steady-state probability of state s', starting in state s, $\mathcal{A}$ an acceptance automaton, and $AccPath^{\mathcal{M}}(s, \mathcal{A})$ the paths of $\mathcal{M}$ that starts in s and are accepted by $\mathcal{A}$.

Evaluation of a CSL$^{\text{TA}}$ formula Φ in a state $s \in S$ (written as $\mathcal{M}, s \models \Phi$) is done in a bottom-up parser tree order. Intuitively, a state formula $\Phi = p$ is true in a state s iff s is labeled with the atomic proposition p (i.e. $p \in lab(s)$). The two formulas $\neg\Phi$ and $\Phi \wedge \Phi$ are the standard boolean formulas. $\mathcal{S}_{\bowtie\lambda}(\Psi)$ is true in $s \in S$ if the sum of the steady state probabilities $\pi(s, s')$ of Ψ-states (states $s' \in S$ where $s' \models \Psi$) is $\bowtie \lambda$. This operator requires the computation of the steady-state probabilities of $\mathcal{M}$.

The operator $\mathcal{P}_{\bowtie\lambda}(\mathcal{A})$ is true in $s \in S$ if the probability of $AccPath^{\mathcal{M}}(s, \mathcal{A})$ (the set of paths accepted by the automaton $\mathcal{A}$) is $\bowtie \lambda$. The computation of $Pr_s^{\mathcal{M}}\big(AccPath^{\mathcal{M}}(s, \mathcal{A})\big)$ is the most difficult problem to solve in order to evaluate a CSL$^{\text{TA}}$ expression, and this is precisely the problem tackled in this paper.

A clock x is a variable whose value $\bar{x}$ increases linearly with time. The timed automata of the CSL$^{\text{TA}}$ path formulas are *Deterministic Timed Automaton* (DTA) with a *single* clock x. DTAs are characterized by two types of edges:
Inner Edges, which are labeled with a clock constraint γ in the form: $\alpha < x < \beta$, with $\alpha, \beta \in \mathbb{R}_{\geq 0}$, and a set of allowed ASMC actions;
Boundary Edges, which are labeled with a time constraint γ in the form $x = \alpha$. No action is associated to a boundary edge, and the special symbol # is used.

The following conditions apply to a DTA (these limitations are required to obtain a MRP [1]).

- It accepts only finite paths.
- It is **Deterministic**: for each path σ in the ASMC, there exists a single path $\sigma_{\mathcal{A}}$ in the DTA that accepts or rejects σ.
- Each edge may have a time constraint γ. We write $\bar{x} \models \gamma$ to say that γ is satisfied at instant $\bar{x}$.

Locations of the DTA are labeled with *state proposition expressions* $\Delta(l)$, which are boolean expression over a set of Atomic propositions Σ, indicated as $\mathcal{B}_\Sigma$. CSL$^{\text{TA}}$ considers DTAs where state propositions in Σ can be both atomic propositions of the ASMC or CSL$^{\text{TA}}$ subformulas, like: $\Delta(l) = \mathcal{S}_{\bowtie\lambda}(\Phi_1) \vee \neg\Phi_2$. This allows the definition of nested path formulas.

Let x a clock, $Inner$ the set of inner constraints (constraints of the form $\alpha < x < \beta$) and $Boundary$ the set of boundary constraints (with form $x = \alpha$), then a DTA $\mathcal{A}$ is a tuple $\mathcal{A} =< \Sigma, Act, L, \Delta, Init, Final, \rightarrow >$ with:

- Σ: a finite alphabet of State Propositions,
- Act: a finite set of action names,
- L: a finite set of locations,
- $\Delta : L \rightarrow \mathcal{B}_\Sigma$: a location labeling function,
- $Init$: initial locations ($Init \subseteq L$),
- $Final$: final locations ($Final \subseteq L$),
- $\rightarrow \subseteq \ L \times \big((Inner \times 2^{Act}) \cup (Boundary \times \{\#\})\big) \times \{\varnothing, x\} \times L$: set of edges. The notation $l \xrightarrow{\gamma, A, r} l'$ is a synonym for $(l, \gamma, A, r, l') \in \rightarrow$.

Determinism can be ensured syntactically through a set of rules to be checked on the DTA (see [1], Section II-B)

Figure 1 shows two simple DTAs, drawn as a graph of locations and edges. *Boundary* edges are represented with dotted lines, while solid lines depict *Inner* edges. Initial and final locations are drawn with an entering edge and with a double contour, respectively. State proposition expressions $\Delta(l)$ are written aside their location.

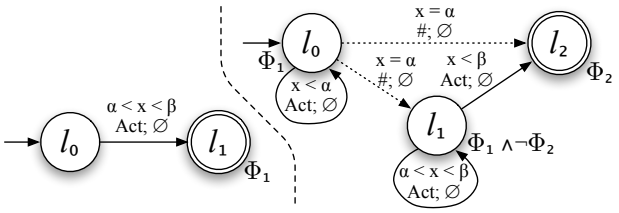

Figure 1. The DTAs $\mathcal{X}^{[\alpha,\beta]}\Phi_1$ and $\Phi_1\, \mathcal{U}^{[\alpha,\beta]}\Phi_2$.

The DTA $\mathcal{X}^{[\alpha,\beta]}\Phi_1$ (on the left) is straightforward, since there is a single edge between l_0 and l_1. We use the symbol Act to denote that all ASMC actions are allowed. The last symbol of the edge label may be one of the following symbols: $\{x\}$ or $\varnothing$. $\{x\}$ means that the clock is reset to 0 when this edge is followed, and $\varnothing$ means that the clock maintains its value when the edge is followed. The DTA $\Phi_1\, \mathcal{U}^{[\alpha,\beta]}\Phi_2$ (on the right) is a more complex DTA, with both *Inner* and *Boundary* edges, and with *Inner* self-loops in l_0 and l_1.

To complete the summary of CSL$^{\text{TA}}$ we need to define which paths σ of an ASMC $\mathcal{M}$ are accepted by a DTA $\mathcal{A}$.

A *configuration* of $\mathcal{A}$ is the state vector $(l, \bar{x})$ of the DTA, with l the current location and $\bar{x}$ the clock value. A finite path $\sigma_{\mathcal{A}}$ in a DTA is a sequence of configurations:

$$(l_0, \bar{x}_0) \xrightarrow{\delta_0, e_0} (l_1, \bar{x}_1) \xrightarrow{\delta_1, e_1} \cdots \xrightarrow{\delta_{n-1}, e_{n-1}} (l_n, \bar{x}_n)$$

with δ_i the elapsed time between two configurations, and e the edge followed. Path acceptance follows four rules:
(1) Inner edges are triggered by ASMC transitions: each transition $s_i \xrightarrow{a_i, \tau_i} s_{i+1}$ of $\mathcal{M}$ is read by a single Inner edge $e = (l, \gamma, A, r, l')$ of the DTA, leading to the DTA transition $(l, \bar{x}) \xrightarrow{\delta, e} (l', \bar{x}')$, with $\delta = \tau_i$. Clock, action and state constraints are satisfied, that is to say: $(\bar{x} + \delta) \models \gamma$, $\Delta(l') \models_\Sigma s_{i+1}$, $a_i \in A$, $\bar{x}' = (\bar{x} + \delta$ when $r = \varnothing$, 0 when $r = \{x\})$. If there is no edge able to read an ASMC transition, the path σ is rejected.
(2) Boundary edges are triggered by the elapse of time: a Boundary edge $e = (l, \gamma, \#, r, l')$ is taken independently from the ASMC. Let s be the state of $\mathcal{M}$ and $(l, \bar{x})$ the configuration of $\mathcal{A}$. When the clock x satisfies the guard $\gamma : x = \alpha$, and the state condition ($\Delta(l') \models_\Sigma s$), the edge e is taken, leading the DTA in $(l', \bar{x}')$, with $\bar{x}' = \bar{x} + \delta$ when $r = \varnothing$ and $\bar{x}' = 0$ when $r = \{x\}$.
(3) Boundary edges are urgent and have priority: if a Boundary edge e is able to fire, it fires immediately; moreover if both an Inner edge e_i and a Boundary edge e_b can fire, then e_b is followed first.
(4) Acceptance: when the DTA reaches a configuration $(l, \bar{x})$, with $l \in Final$, then the path σ of $\mathcal{M}$ is accepted.

Figure 2 shows an ASMC $\mathcal{M}$ and a DTA $\mathcal{A}$. DTA edges are marked with circled numbers for later references. Figure 3 shows two sample paths $\sigma_{\mathcal{M}}$, one accepted by $\mathcal{A}$ (case 1) and one rejected (case 2). In each example the first line shows the path σ of $\mathcal{M}$ and the pairs *<action, elapsed time>*. The second line shows the transitions made by $\mathcal{A}$ in order to read $\sigma_{\mathcal{M}}$. Inner edges of the path are shown with dotted lines.

Example 1 starts in s_0, l_0; after 3 seconds, the *Boundary* edge $e_{0,1}$ can fire and moves $\mathcal{A}$ in l_1. Then at instant 4 $\mathcal{M}$ moves to s_1 with an action a, and this triggers the edge $e_{1,2}$ of $\mathcal{A}$; also, the clock is reset. After 0.3 instants, $\mathcal{M}$ moves to s_2 with action b and $\mathcal{A}$ follows going to l_3. At this point a $Final$ location has been reached, and σ is accepted. Example 2 is quite similar, except that configuration $(l_2, 0)$ of $\mathcal{A}$ cannot read the transition $s_1 \xrightarrow{b,3} s_2$, because there is no Inner edge e starting from l_2 with all its constraints satisfied. Therefore $\mathcal{A}$ rejects σ. The determinism requirement ensures that for each $\sigma_{\mathcal{M}}$ there is a single $\sigma_{\mathcal{A}}$ that reads it.

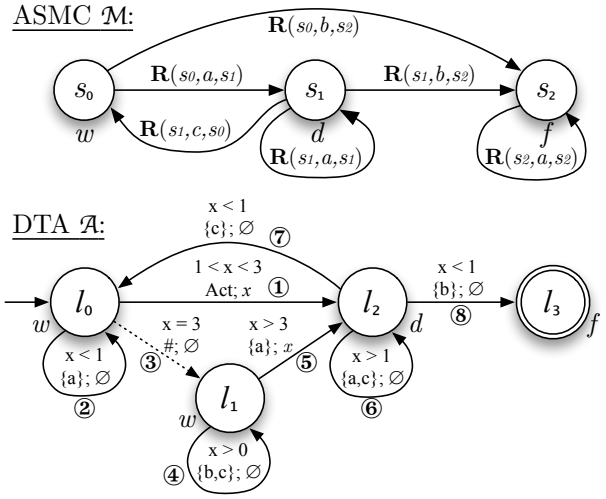

Figure 2. An ASMC $\mathcal{M}$ and a DTA $\mathcal{A}$.

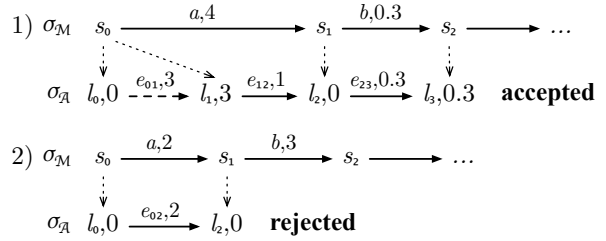

Figure 3. Two sample paths of $\mathcal{M}$ read by $\mathcal{A}$.

2.1. Model-checking of CSL$^{\text{TA}}$

The evaluation of the CSL$^{\text{TA}}$ operator $\mathcal{P}_{\bowtie\lambda}$ requires to know the overall probability of following accepted paths between the infinite paths that starts in s_0. This probability can be computed [1, sec. 3] on a "synchronized" stochastic process $\mathcal{M}\times\mathcal{A}$. Intuitively, in $\mathcal{M}\times\mathcal{A}$ the ASMC $\mathcal{M}$ and the clock x are concurrently enabled: the transitions of both are then "constrained" by the DTA $\mathcal{A}$.

Let $C = \{0, c_1, c_2, \ldots, c_n\}$ be the ordered set of values that appear in clock constraints of $\mathcal{A}$, and $next : c_i \to c_{i+1}$ a function that associates each of these constants to their subsequent (with $next(c_n) = \infty$). In the example of Figure 2 we have $C = \{0, 1, 3\}$.

The state vector of $\mathcal{M}\times\mathcal{A}$ has the form $<s, l, c>$ with $c \in C$; c denotes the *clock region* $\big(c, next(c)\big)$ reached by clock x. Inner edges may fire within a clock region, while Boundary edges may fire when the clock reaches a clock constant $c \in C$. Two special states are added to $\mathcal{M}\times\mathcal{A}$: $\top$ (accept) and $\bot$ (reject), so that:

- When $\mathcal{A}$ reaches a $Final$ location, $\mathcal{M}\times\mathcal{A}$ moves to $\top$.
- If $\mathcal{A}$ isn't able to read a transition of $\mathcal{M}$, then $\mathcal{M}\times\mathcal{A}$ moves to $\bot$.
- Otherwise, the stochastic process $\mathcal{M}\times\mathcal{A}$ stores in its state $<s, l, c>$ the current state s of $\mathcal{M}$, the location l of $\mathcal{A}$ and the clock region c.

$\mathcal{M}\times\mathcal{A}$ is a non-ergodic Markov Regenerative Process. A good introduction to MRPs can be found in [3].

The state space of $\mathcal{M}\times\mathcal{A}$ can be represented with a *Tangible Reachability Graph* (TRG), where states are a subset of $(S \times L \times C) \cup \{\top, \bot\}$, and transitions are either of deterministic duration or of Markovian duration. The MRP only includes *tangible* states (states with a non-zero sojourn time), namely states where Boundary edges are not enabled. Therefore we define $closure(s, l, c)$ as the next tangible state of $\mathcal{M}\times\mathcal{A}$ reached after following a sequence of (one or more) Boundary edges. More precisely:

- If $l \in Final$, then $closure(s, l, c) = \top$.
- If $\exists e = (l, \gamma, \#, r, l')$ enabled in $<s, l, c>$ (with $c \models \gamma$ and $s \models_\Sigma \Delta(l')$) then $closure(s, l, c) = closure(s, l', c')$, with $c' = (c$ if $r = \varnothing$, 0 otherwise $)$.
- If there is no Boundary edge enabled, $closure(s, l, c) = <s, l, c>$,

Arcs of the TRG can be divided into four classes:

[M] A simple Markovian move: a transition $\mathbf{R}(s, a, s')$ of $\mathcal{M}$ fires and the Inner edge $e = (l, \gamma, A, \varnothing, l')$ of $\mathcal{A}$ reads it, with no clock reset.
Formally, $<s, l, c> \xrightarrow{\mathbf{M}(a,e)} <s', l', c>$, with $s' \models_\Sigma \Delta(l')$, $a \in A$ and $(c, next(c)) \models \gamma$. If $l' \in Final$, then $\mathcal{M}\times\mathcal{A}$ moves to $\top$ instead of (s', l', c).

[M_res] A Markovian move with a clock reset, that can enable a sequence of Boundary edges.
Formally, $<s, l, c> \xrightarrow{\mathbf{M\_res}(a,e)} closure(s', l', 0)$ with the same constraints of the previous class of arcs.

[M_KO] A Markovian move that is rejected by $\mathcal{A}$: $<s, l, c> \xrightarrow{\mathbf{M\_KO}(a)} \bot$.

[D] Clock region $\big(c, next(c)\big)$ has expired. There is at most one single **D** arc from each TRG state, because there is only a single clock x.
Formally, $<s, l, c> \xrightarrow{\mathbf{D}(next(c))} closure\big(s, l, next(c)\big)$.

Figure 4 shows the TRG of the example in Figure 2. Edges of $\mathcal{A}$ are marked with circled numbers.

The verification of $\mathcal{P}_{\bowtie\lambda}(\mathcal{A})$ on the ASMC $\mathcal{M}$ reduces to checking whether *the steady-state probability of reaching state $\top$ in $\mathcal{M}\times\mathcal{A}$ is $\bowtie \lambda$*. Next section illustrates how this value is computed using a DSPN solver.

3. Model Checking $\mathcal{P}_{\bowtie\lambda}$ formulas with DSPNs

The starting point of our model checker is to generate the TRG of $\mathcal{M}\times\mathcal{A}$ by coding the rules of path acceptance with a DSPN. DSPNs (*Deterministic and Stochastic Petri Nets*) are a Generalized Stochastic Petri net superset, where timed

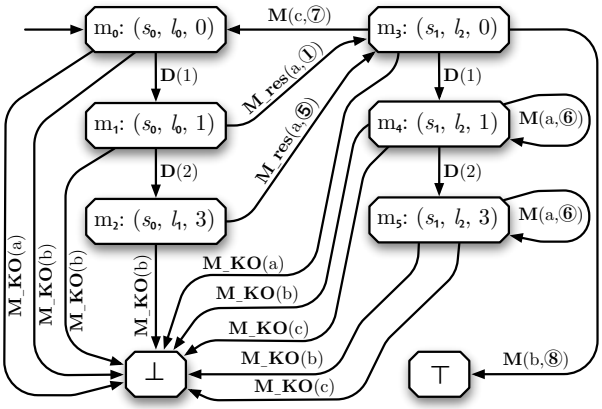

Figure 4. $\mathcal{M}\times\mathcal{A}$ TRG of the example in fig. 2.

transitions can be both exponentially and deterministically distributed. The stochastic process underlying a DSPN with deterministic transitions not concurrently enabled is a MRP. Good introductions to DSPNs can be found in [3] and [2].

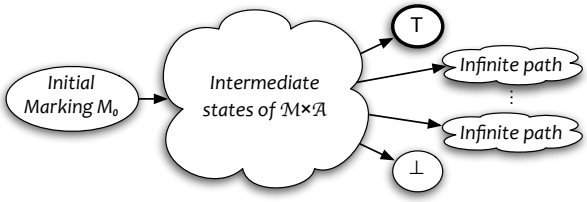

Figure 5. Typical structure of the TRG.

The structure of $\mathcal{M}\times\mathcal{A}$ is shown in figure 5. Infinite paths (those path $\sigma_{\mathcal{M}}$ that are never accepted nor rejected by $\mathcal{A}$) generate *Bottom Strongly Connected Components* (recurrence classes in CTMC terminology).

For DSPNs, we use the notation P($name$) to name specific places, and T($name$) to name transitions. With a slight abuse of the notation, we use P($name$)=k to mean that the place called $name$ has k tokens in the current marking. Figure 6 shows the DSPN subnets that compose the DSPN that generates a TRG isomorphic to the TRG of the stochastic process $\mathcal{M}\times\mathcal{A}$.

The DSPN is divided into 6 interconnected subnets. The first three (ASMC, Clock and DTA subnets) model the three components of the state vector $<s, l, c>$ of $\mathcal{M}\times\mathcal{A}$. We will define them later in this section. Transitions of the ASMC and Clock subnets can be concurrently enabled, while transitions in the DTA subnet are only enabled after the firing of a transition in the ASMC or Clock subnet.

After an ASMC transition fires, a sequence of DTA transitions may fire, to implement acceptance or rejection of the move. A DTA subnet transition can also fire when a clock reaches the boundary of the current clock region. To check for acceptance the DSPN explicitely models the set of actions and the set of state expressions that appear in the DTA as DSPN places, organized into three auxiliary subnets:

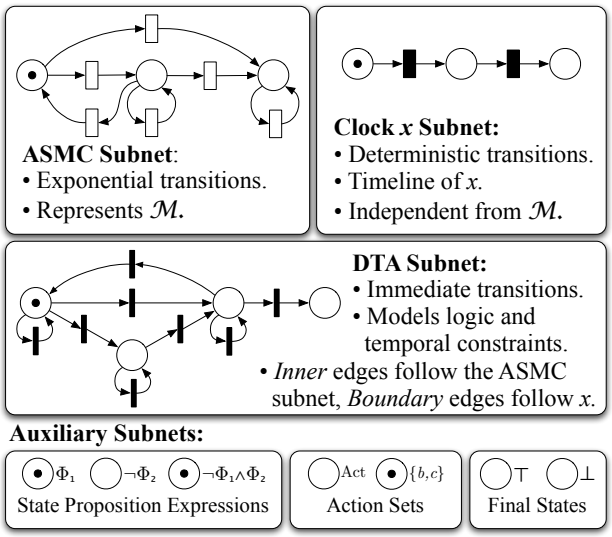

Figure 6. Subnets of the DSPN.

- **State proposition expressions of** $\mathcal{A}$: let $SpExpr = \{\Delta(l) \mid l \in L\}$ be the set of the expressions associated to $\mathcal{A}$ locations. This subnet has a place P($expr$) for each unique $expr \in SpExpr$. In the DSPN this invariant holds: *in each non-final marking* $<s, l, c>$*, if* $s \models_{\Sigma} expr$*, then P*($expr$)*=1, otherwise P*($expr$)*=0.*
- **Action sets of** $\mathcal{A}$: let $ActSets \subseteq 2^{Act}$ be the set of action sets that labels DTA edges, that is to say, $A \in ActSets = \{A : \exists(l, \gamma, A', r, l') \in \rightarrow\}$. Places P($A$), for $A \in ActSets$ are used to translate the action set constraints of Inner edges. In short, after each transition $\mathbf{R}(s_i, a, s_j)$ of the ASMC subnet fires, a token is put in each place P(A), $\forall A \in ActSets \ \wedge \ a \in A$.
- **Final states**: represents the final markings $\top$ and $\bot$. When a token is put in one of those places, a subnet of high priority immediate transitions removes all tokens from the DSPN, thus blocking any further behavior.

In the example of Figure 2, $SpExpr = \{w, d, f\}$ (the DTA location-labeling expressions $\Delta(l)$); also, $ActExpr = \{Act, \{a\}, \{b, c\}, \{a, c\}, \{b\}\}$. Each of these entries is represented by a place in the auxiliary subnets. We are interested in the steady-state probability of P($\top$) = 1.

The ASMC $\mathcal{M}$ subnet. Figure 7 shows the skeleton structure of the subnet for the ASMC in figure 2. The ASMC subnet is a 1-bounded SPN where each state $s \in S$ of $\mathcal{M}$ is mapped into a corresponding place P(s), and each transition $s_i \xrightarrow{a} s_j$ is represented by an exponential transition T(s_i, a, s_j) of rate $\mathbf{R}(s_i, a, s_j)$. At most one place is marked at any time. To account for actions and state

propositions, the following behaviors are enforced when each transition t=T(s_i, a, s_j) of the ASMC subnet fires:

- t removes the $SpExpr$s tokens for those expressions that become false from s_i to s_j, and adds a token in each $SpExpr$ place that becomes true from s_i to s_j. In this way, the invariant of the $SpExpr$ net is preserved.
- t adds a token in the special place P(*Moved*), to enable the DTA subnet that "reads" (s_i, a, s_j).
- t adds a token in each P(A) such that $a \in A, \forall A \in ActSets$. This is used to implement the action set constraint of Inner edges, as explained later.

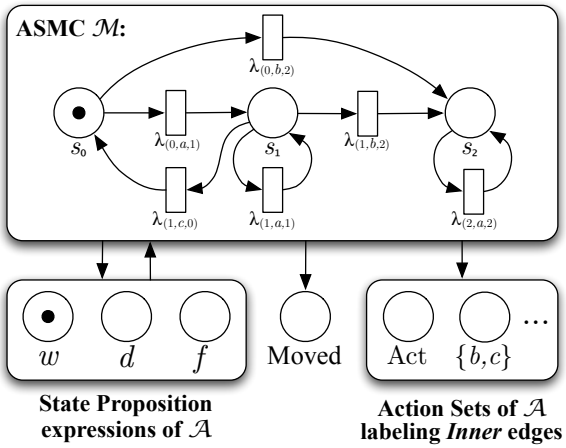

Figure 7. Structure of the ASMC subnet.

The Clock x subnet Figure 8 shows the subnet for $C = \{0, 1, 3\}$. The Clock x subnet is an independent process that models a clock timeline with a sequence of deterministic transitions. The deterministic transition of the k-th clock region is enabled when P$(Clock)$ has exactly k tokens. Therefore, deterministic transitions are enabled at most one at a time. No transition is enabled in the clock region $x > c_n$, since, by definition, c_n is the greatest value against which the timer x is compared. For each $c \in C$ there is one place P(x=c) used to enable *Boundary* edges with guard $\gamma : x = c$; these places are marked only in vanishing states. Note also the clock reset mechanism: when a token is put in P(x=0) the clock is immediately reset (P$(Clock)$=0), and any deterministic transition is preempted.

The DTA $\mathcal{A}$ subnet Figure 9 shows the skeleton of the DTA subnet generated for the example in figure 2. The DTA $\mathcal{A}$ subnet is a 1-bounded net of immediate transitions, where locations $l \in L$ are translated into places P(l), and *Inner*/*Boundary* edges $e \in \rightarrow$ are translated into immediate transitions T(e). Each place corresponding to a $Final$ location has an additional high-priority immediate transition

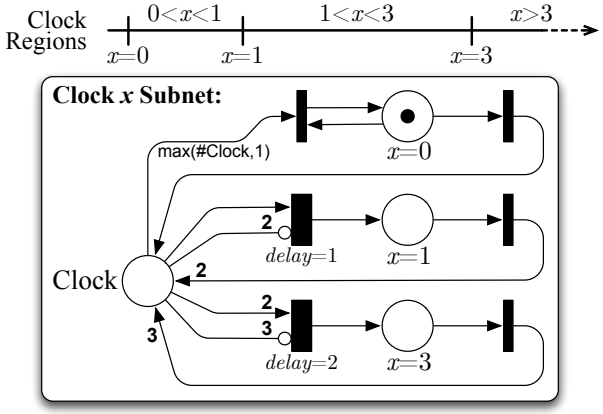

Figure 8. Structure of the Clock x subnet.

that moves the DSPN in the $\top$ marking (like P(l_3) in the example). The skeleton of figure 9 should be completed, since each transition T(e) has to follow the ASMC or the Clock when the constraints are satisfied. Therefore, we must add other input and output arcs to these transition T(e) in order to model the constraints.

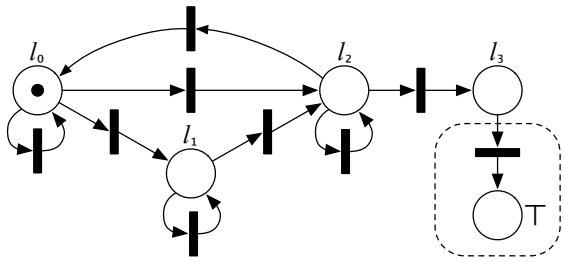

Figure 9. An example of DTA translation.

Let $e = (l_i, \gamma, A, r, l_j)$ be an *Inner* edge of $\mathcal{A}$, and $\mathbf{R}(s, a, s')$ an ASMC transition. The enabling conditions of T(e) should check that:

1. State propositions expression $\Delta(l_j)$ of the destination location l_j is satisfied ($s' \models_\Sigma \Delta(l_j)$). Therefore, T$(e)$ must test that the corresponding $SpExpr$ place P$(\Delta(l_j))$ is marked.
2. The ASMC action a is in the action set A ($a \in A$). This is implemented by removing a token in the corresponding $ActSet$ place P(A).
3. Clock guard $\gamma : c_\alpha < x < c_\beta$ is satisfied ($\bar{x} \models \gamma$). T(e) tests P$(Clock)$ for at least α tokens, but no more than β tokens. Observe that P$(Clock)$ is essentially a counter of the current clock region, and has 1 or more tokens in every non-terminal tangible marking.
4. Clock reset (if $r = \{x\}$): T(e) adds a token in P(x=0). As a consequence, the clock is reset to 0, as explained in section 3.
5. Each transition of $\mathcal{M}$ is "read" by exactly one *Inner*

edge (due to determinism): therefore T(e) removes the token from the special place P(*Moved*). If no *Inner* edge is enabled after an ASMC transition, the DSPN reaches $\perp$. This is implemented with a lower-priority immediate transition that removes the token in P(*Moved*) and adds it in P($\perp$) if no *Inner* edge fired.

Figure 10 shows the translation of an Inner edge e; previous constraints are denoted by circled numbers.

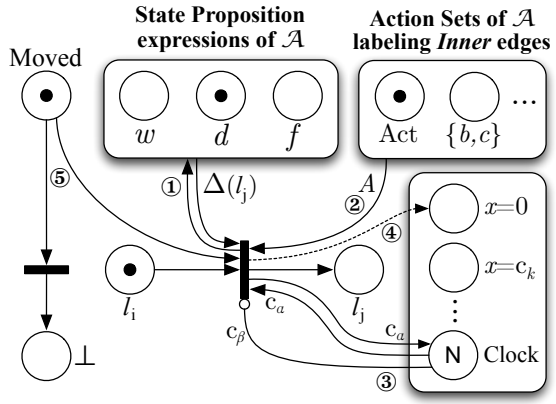

Figure 10. *Inner* **edge translation.**

When $e = (l_i, \gamma, \#, r, l_j)$ is a *Boundary* edge with γ : x=c_α and T(e) is its corresponding transition, the following conditions must hold for T(e) to be enabled:

1. State propositions expression $\Delta(l_j)$ of the destination location l_j must be satisfied ($s' \models_\Sigma \Delta(l_j)$). This is implemented as in the case of *Inner* edges.
2. Clock guard γ : x=c_α is satisfied ($\bar{x} \models \gamma$). Therefore, T(e) tests if place P(x=α) has a token.
3. Clock reset when $r = \{x\}$. This is implemented as in the case of *Inner* edges.

Figure 11 shows the translation of a Boundary edge e.

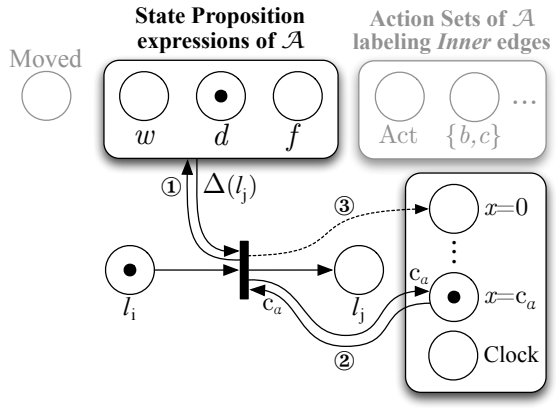

Figure 11. *Boundary* **edge translation.**

3.1. Considerations on the generated DSPN

The DSPN includes some additional clean-up behavior. The $ActSet$ subnet loses all its tokens after an *Inner* edge transition fires. In this way, these places "records" only the last action sets triggered by $\mathcal{M}$.

Priorities are assigned to immediate transition with a scheme that ensures that the DTA follows the ASMC/Clock transitions before any clean-up takes place. Moreover every place is cleaned when a final place P($\top$) or P($\perp$) is marked. The idea of representing *state propositions* and *actions* as DSPN places proved to be a very easy and powerful way to carry state and transition labels back into the DSPN.

The clock x subnet is represented with a single P(*Clock*) place instead of a timeline (as in [13]) because clock constraints c_α<x<c_β are easier to implement: we simply test for at least α tokens but less than β in P(*Clock*).

3.2. Correctness of the translation

The generated TRG of the DSPN is isomorphic to the stochastic process $\mathcal{M}\times\mathcal{A}$ in every tangible marking, according to this mapping:

- $\top$ and $\perp$ states of $\mathcal{M}\times\mathcal{A}$ corresponds to the two markings <P($\top$)> and <P($\perp$)>.
- A state $<s, l, c>$ corresponds to the tangible marking: <P(s), P(l), P($Clock$)=c, P($SpExpr$ true in s)>

and we need to prove that each arc of the TRG of the $\mathcal{M}\times\mathcal{A}$ process is also an arc in the DSPN TRG, in accordance with the four types of TRG arcs explained in section 2.1, and viceversa.

An **[M(a,e)]** arc is simply the result of the firing of transition T(s, a, s') of the ASMC $\mathcal{M}$ subnet, followed by the firing of T(e). An **[M_res(a,e)]** resets also the clock. The clock reset is carried out in the clock subnet. As a consequence of the reset, *Boundary* edges may fire, mimicking exactly the semantic of $closure(s', l', c)$. An **[M_KO(a)]** arc is the result of an ASMC transition not read by $\mathcal{A}$: this happens in the DSPN when the P($Moved$) place is not cleared by any *Inner* edge. Finally, a **[D(c)]** arc is implemented by the firing of a deterministic transition of the clock subnet. *Boundary* edges may fire only at a clock boundaries, as required by the semantic of the $\mathcal{M}\times\mathcal{A}$ process. It is important to note that the constraints of DTA edges T(e) prevent those edges to fire in every other occasions except when they have to follow ASMC/Clock transitions.

For the other direction of the proof, we can observe that in each tangible marking M of the DSPN TRG corresponding to the $<s, l, c>$ state of the $\mathcal{M}\times\mathcal{A}$ process, if an ASMC transition $\mathbf{R}(s, a, s')$ can/cannot fire, then the corresponding transition T(s, a, s') is enabled/disabled in M. Also, if a

D(c) arc can fire from $<s, l, c>$ (because $c \neq c_m$), then the corresponding deterministic transition in the Clock x subnet is enabled in M. Otherwise, no deterministic transitions are enabled in M. Therefore, the DSPN TRG is built with the same rules of the $\mathcal{M}\times\mathcal{A}$ process, and is isomorphic to the latter. A more detailed proof can be found at `www.di.unito.it/~susi/PDS10/TR.pdf`. Given a finite ASMC and a finite DTA, the state space generated by the DSPN is upper bounded: $|\text{TRG}| \leq \big(|S_{\mathcal{M}}| \times |L_{\mathcal{A}}| \times |C|\big)$. This makes this approach always feasible (with enough memory/time). However, the MRP steady state solution cost depends on the numerical solution method used.

4. Model Checker Implementation

The proposed algorithm has been developed in a small CSL$^{\text{TA}}$ model checker, structured as depicted in Figure 12. Numerical analysis is supplied by external DSPN solvers and we tested three solvers: SPNica[3], DSPNexpress[2] and TimeNET[7]. The tool, named `CslTa-Solver`, is written in C++ with approximately 7000 lines of code.

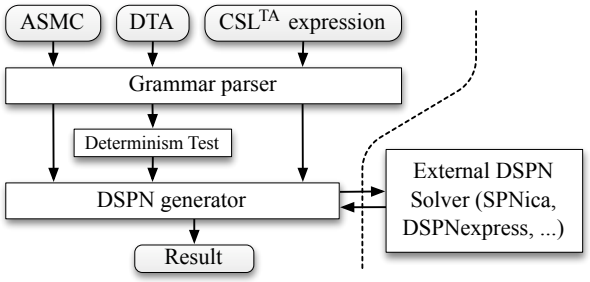

Figure 12. `CslTa-Solver` **software schema.**

`CslTa-Solver` uses a slightly more practical definition for DTAs, which is *parametric* in the action names, state propositions, and clock constants of the DTA. These parameters are then instantiated inside CSL$^{\text{TA}}$ espressions. In this way, common DTAs (like $\mathcal{X}^{[\alpha,\beta]}\Phi_1$ or $\Phi_1\mathcal{U}^{[\alpha,\beta]}\Phi_2$) can be re-used and shared by many CSL$^{\text{TA}}$ queries. DTA determinism is **verified**, both in parametric and in instantiated form.

In case of nested formula one or more states of the DTA are labelled with a path formula specified through a different DTA, therefore the model-checking procedure of Figure 12 is applied several times, one per DTA (and per state) involved, following a parse-tree bottom-up order.

4.1. Example: a Cyclic Polling Server

In our example the ASMC is generated by a *Cyclic Polling Server* (CPS), which is a widely studied queueing system. The CPS we consider is composed of two main elements: N single-buffered stations $s_{1\ldots N}$, each with arrival rate λ and a single server, which polls in ordered sequence each of the N stations. When, upon polling, the server finds a job waiting in the k-th station, the job is served with completion rate μ. Moving from queue to queue is an activity of rate γ, subject to a **failure rate** ρ (no repair is considered).

The ASMC state proposition s_k means that the server is at station k, while state proposition *brk* labels the failure states. Action srv_j is associated to the completion of a service at the j-th station, while $empty_j$ is associated to the case of the server that polls a station without a job in the queue.

The first query considered is: *what is the probability of finding all the queues full in the second round when the server serves them?* Figure 13 shows the DTA that describes our query for the case $N = 3$, asssuming the server starts from station 1.

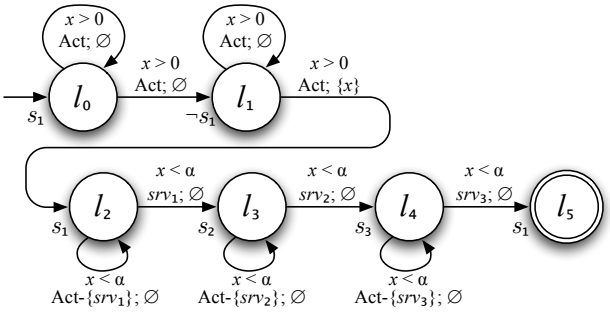

Figure 13. 3 **services in the second round.**

Locations l_0 and l_1 skips the first round of the server (with no time constraints). Then the clock x is reset and the DTA follows an entire round, requiring a srv_j action in each station, until s_1 is reached again. The second round is time bounded with $x < \alpha$.

Figure 14 plots the probabilities of finding all the $N = 3$ queues full within α seconds after an initial round, starting with a state in which all queues are empty and the server is polling station 1, with $\gamma = 10, \lambda = \mu/N$ and $\rho = 1$. Numerical results have been computed with SPNica.

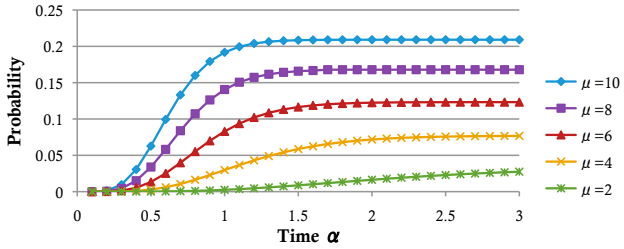

Figure 14. Results for the DTA of fig. 13.

We now show a variation of this path-expression: *what is the probability of completing at least a full round by α seconds, and that no failures in the system occur before*

α? Figure 15 shows the DTA used to specify the accepted paths. Locations l_0, l_1 and l_2 read a full round: if an empty queue is found, the DTA waits the next CPS round. A path is finally accepted by l_5 if it reaches a $\neg brk$ location before time α and no breaks occur before α.

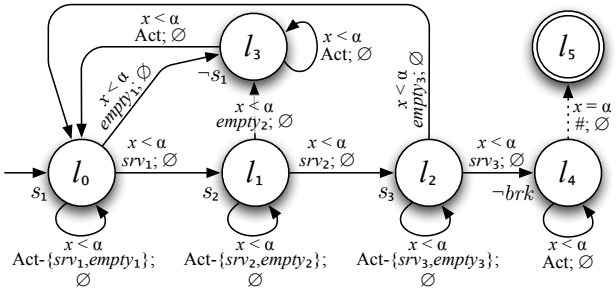

Figure 15. "Full round and no failure".

Results of the query are shown in figure 16. The probability of serving each station in a single round increases with α, but a larger α induces a lower probability of not being in a failure state by time α, leading to the particular behavior shown in the figure.

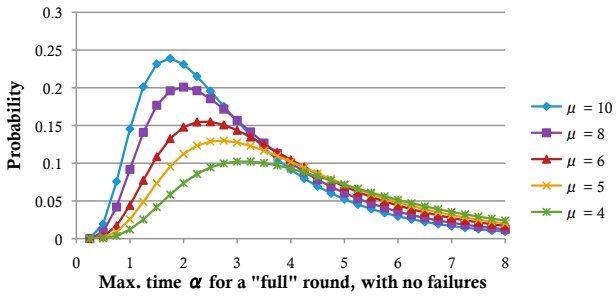

Figure 16. Results of DTA 15, for various μ.

Table 1 reports a sensitivity analysis for varying values of N, for the first query; the table lists the ASMC dimensions, the respective DSPN sizes, the number of Tangible Markings in the state space and the time needed to build the TRG with DSPNexpress on a 1.5Ghz PowerPC.

N	ASMC states	ASMC transit.	DSPN places	DSPN transit.	|TRG|	Time (seconds)
2	12	22	30	64	43	0.629
3	36	84	54	150	123	0.689
4	96	272	114	398	323	1.643
5	240	800	258	1070	803	10.143
6	576	2208	594	2814	1923	112.720
7	1344	5824	1362	7198	4483	1389.700
8	3072	14848	...	...	...	...
9	6912	36864	...	...	...	...
10	15360	89600	...	...	...	...

Table 1. DSPN sizes for the query in figure 15.

These queries shows that complex paths can be easily expressed with CSL^{TA} timed automata, and translated into a DSPN, although many DSPN solvers cannot deal with large DSPNs ($N > 7$ for our example).

4.2. Final evaluation

The proposed translation suffers from two main limitations: inadequacy of the available DSPN solvers and complexity of the MRP solution.

Our solution approach has pursued reuse first, but this reuse was less smooth than we expected. Indeed the DSPNs resulting from the translation have some peculiar characteristics: (1) the TRG is **non ergodic**, and may have 2 or more recurrence classes; (2) the number of places/transitions is huge (in the order of the number of the ASMC states); (3) the TRG may have more than a single arc from each pair of markings $<m_i, m_j>$ (a.k.a. *multiple arcs*); (4) the TRG may have self-loops that preempt a deterministic transition (this may be caused by a DTA edge with a clock reset in the first clock region $(0, c_1)$). Therefore, a DSPN solver should be able to treat each of these characteristics in order to fit our needs. We have considered 3 different DSPN solvers, and unfortunately no one could be used without a significant amount of modifications.

TimeNET: our installation of TimeNET crashes with DSPNs with few hundreds places, and it does not seem to be able to deal with self-loops. Since we did not have access to the source code, we put aside this tool.

SPNica: SPNica is based on Mathematica, it works fine, but it is very limited in the size of the solvable DSPN (around a thousand states), and it should be regarded as a "reference implementation" [3]. It correctly addresses all the four points above but for point (1), for which we have introduced a modification in the code.

DSPNexpress: We focused most of our efforts on this tool, since the source code was available. DSPNexpress, as distributed (actually DSPNExpressNG), is not able to deal with any of the four points above. Points (1) and (3) have been addressed by generating automatically a modified DSPN that circumvents the problems. Point (4) is not solvable, and therefore it is not possible to use DSPNExpress if the DTA has a reset in the first clock region (as in our example). For point (2) we have changed the maximum number of places and transitions in the code, leading to a solution which is correct but very slow (we suspect that this is due to the implementation of the enabling test, which is linear in the number of transitions).

Some examples of nets that cause malfunctions with the solvers can be found at `www.di.unito.it/~susi/PDS10/NETS`. At the time of writing, we prefer to use SPNica for small nets, because of its reliability, and DSPNexpress, with all its limitations,

for bigger nets.

For what concerns complexity, assuming a good DSPN solver is available, by far the most expensive step is the MRP solution (a step that may take place more than once for a formula, if nesting is present). Note that space complexity may also become an issue, since the solution of a MRP requires the solution of an embedded DTMC which is usually very dense (even if the original ASMC is sparse).

Whether the $\mathcal{M}\times\mathcal{A}$ MRP is generated through a DSPN or is directly produced from $\mathcal{M}$ and $\mathcal{A}$, the MRP produced is the same. Therefore its numerical solution has the same cost. It should be noted that computing the TRG of the DSPN or producing the TRG using the definition of the $\mathcal{M}\times\mathcal{A}$ process has the same asymptotical complexity, since the procedure is the same: in the former case the rules to be applied for generating from one state all its successors states are the enabling and firing rule that define the DSPN semantics, while in the latter the rules are the one that define the states and the arcs of the $\mathcal{M}\times\mathcal{A}$, as explained in Section 2.1. Of course even if the construction is the same, the realization may differ. A direct construction of the TRG may be more efficient, since it can be tailored to the specific problem, while the MRP solution behind a DSPN tool should perform better, since it takes advantage of many years of research in the field.

All DSPN tools considered have some limitations in the number of places and transitions. To avoid this problem, and to ease the task of the modeller, it could be a good idea to model-check systems expressed directly as Petri Nets (or in some other higher-level formalism).

5. Conclusions and Future Works

In this paper we presented a simple yet effective way of translating a CSL^{TA} query into an equivalent deterministic and stochastic Petri net. This allows to build a full CSL^{TA} model checker with a limited amount of software development, reusing existing DSPN tools. Unfortunately such a model-checker is only as good as the DSPN solvers used, and, despite the fact that we used DSPN tools that have been around for a while, we found a number of difficulties, as discussed before.

Current work is circumventing this problem through the development of a new DSPN solver which fulfills all the listed requirements. Even with such a solver, the model checking may not be feasible since the MRP numerical solution can become the bottleneck, being very expensive in time and space. We are currently working on the extension to non-ergodic MRPs of the space-efficient iterative solution proposed in [4] and on a decomposition-based approach that should allow to save space and time.

Acknowledgments. We would like to thank R. German for his help in modifying SPNica for non-ergodic DSPNs.

References

[1] Susanna Donatelli, Serge Haddad, Jeremy Sproston, "*Model Checking Timed and Stochastic Properties with CSL^{TA}*", IEEE Transactions on Software Engineering, vol. 35, no. 2, pp. 224-240, March/April, 2009.

[2] Cristoph Lindemann, "*Performance Modelling with Deterministic and Stochastic Petri Nets*", John Wiley & Sons Ltd, March 1998.

[3] Reinhard German, "*Performance Analysis of Communication Systems - Modeling with non-Markovian Stochastic Petri Nets*", John Wiley & Sons Ltd, 1 edition, May 17, 2000

[4] R. German. Iterative analysis of markov regenerative models. *Perform. Eval.*, 44:51–72, April 2001.

[5] Christel Baier, Lucia Cloth, Boudewijn R. Haverkort, Matthias Kuntz, Markus Siegle, "*Model Checking Markov Chains with Actions and State Labels*", IEEE Transactions on Software Engineering, vol. 33, no. 4, pp. 209-224, 2007.

[6] Adnan Aziz, Kumud Sanwal, Vigyan Singhal, Robert Brayton "*Model Checking Continuous Time Markov Chains*" ACM Trans. on Computational Logic, 2000

[7] German, R.; Kelling, Ch.; Zimmermann, A.; Hommel, G., "*TimeNET - A Toolkit for Evaluating Stochastic Petri Nets with Non-Exponential Firing Times*", Journal of Performance Evaluation, Elsevier, Netherlands, 1995

[8] Christel Baier, Boudewijn Haverkort, Holger Hermanns, and Joost-Pieter Katoen "*Model-Checking Algorithms for Continuous-Time Markov Chains*", IEEE Transaction on Software Engineering, Vol. 29, No. 7, July 2003

[9] Taolue Chen, Tingting Han, Joost-Pieter Katoen, Alexandru Mereacre "*Quantitative Model Checking of Continuous-Time Markov Chains Against Timed Automata Specifications*" Logic in Computer Science, Symposium on, Vol. 0, pages. 309-318, 2009

[10] A. Hinton, M. Kwiatkowska, G. Norman and D. Parker, "*PRISM: A Tool for Automatic Verification of Probabilistic Systems.*", Proc. 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06), 2006.

[11] Joost-Pieter Katoen, Ivan S. Zapreev, Ernst Moritz Hahn, Holger Hermanns, David N. Jansen, "*The Ins and Outs of the Probabilistic Model Checker MRMC*", Quantitative Evaluation of Systems, International Conference on, pp. 167-176, 2009.

[12] R. Alur and D. Dill "*Automata for Modelling Real-Time Systems*", In Proc. of ICALP'90, vol.443, 1990

[13] Bérard, B. and Cassez, F. and Haddad, S. and Lime, D. and Roux, O. H. "*When are Timed Automata weakly timed bisimilar to Time Petri Nets? Theoretical Computer Science*", Volume 403, Elsevier Science Publishers Ltd, 2008.