CSL Model Checking for the GreatSPN Tool*

Davide D'Aprile, Susanna Donatelli, and Jeremy Sproston

Dipartimento di Informatica, Università di Torino, Torino, Italy
{daprile,susi,sproston}@di.unito.it

Abstract. CSL is a stochastic temporal logic that has been defined for continuous time Markov chains, and that allows the checking of whether a single state, or a set of states, satisfies a given probabilistic condition defined over states or over a path of states. In this paper we consider the problem of CSL model checking in the context of Generalized Stochastic Petri Nets. We present a translation from Generalized Stochastic Petri Nets to the input formats of two well-known CSL model checkers, namely ETMCC and PRISM. The transformation to ETMCC is realized at the Markov Chain level, while that to PRISM is defined, as much as possible, at the net level. The translations are applied to a multiserver polling model taken from the literature.

1 Introduction

Generalized Stochastic Petri Nets (GSPN) [1] and their coloured counterpart Stochastic Well-formed Nets (SWN) [1,6] are widespread formalisms for the modelling of complex systems. They have been designed for performance evaluation, but they can, and have been, used also to validate qualitative properties of the system under study, either using *structural* properties, checkable on the net itself without building the state space, or using *reachability* analysis.

Recently a stochastic logic for Continuous Time Markov Chains (CTMC) has been defined [4,5], named Continuous Stochastic Logic (CSL), which collapses in a single logic the ability to specify qualitative and quantitative properties. CSL properties are verified at the state-space level using the model checking paradigm [8].

The ability to verify GSPN and SWN against CSL properties would increase the qualitative and quantitative verification capability of GSPN (and SWN) tools: in particular we think that this feature is particularly significant for SWN models, because of the limited availability of algorithms and tools to check structural properties on SWN (and in general on coloured nets). Moreover SWN is a very useful formalism for two main motivations: its ability to represent compactly complex models thanks to the ability of identifying tokens in a net using colours, and the possibility of computing the state space and the underlying CTMC in an efficient manner through the exploitation of symmetries [6]. Last, but definitely not least, we concentrate on this specific class of coloured nets for

* Supported in part by the MIUR-FIRB project PERF.

C. Aykanat et al. (Eds.): ISCIS 2004, LNCS 3280, pp. 543–553, 2004.

pragmatic reasons, because we have full access to a tool for the definition and analysis of GSPN and SWN, named GreatSPN [10], which has been developed over the last 20 years at our department. We add that there is not a wide choice of alternative tools for *stochastic* coloured nets: a simple form of colours is implemented in the APNN tool [3] of the University of Dortmund, and a notion of colours based on replication is implemented in UltraSAN [16] of the University of Illinois.

Our mid-term goal is indeed to have CSL facilities for SWN nets, and in this paper we report on the first step we have undertaken in this direction: to add to GreatSPN a Csl model-checking facility for GSPNs. Instead of building a new model checker we have linked GreatSPN to two well known CSL-model checkers: ETMCC [12], a joint effort of the Universities of Erlangen and Twente, and Prism [15], of the University of Birmingham. We have not considered for the moment the Csl model checker built-in in APNN [3], instead preferring to use two "stand-alone" model checkers, the motivation being that we hope they can be not too difficult to connect with the GreatSPN modules for SWN, and because there is already some reported experience in interfacing other tools (ETMCC has been interfaced with the Petri net tool DaNAMiCS [9], and the process algebra tool TIPP [11], while Prism has been interfaced with the PEPA process algebra [13]).

The paper is organized as follows. Section 2 provides the necessary background on Csl and on the tools used in this work (ETMCC, Prism, and GreatSPN), Section 3 introduces the translations from GreatSPN to ETMCC and Prism, translations which are used in Section 4 to show examples of Csl model checking for a GSPN model taken from the literature. Section 5 concludes the paper.

2 Preparing the Ground: Csl, ETMCC, Prism, and GreatSPN

In this section we provide a brief, informal introduction of Csl followed by a presentation of the three tools involved in this work: ETMCC, Prism, and GreatSPN. Due to space limitations, the presentation of the tools is only partial, centered on the main features used for our goals.

2.1 Model Checking Csl Properties

The syntax of Csl [4,5] is defined as follows:

$$\Phi ::= a \mid \Phi \wedge \Phi \mid \neg\Phi \mid \mathcal{P}_{\bowtie\lambda}[X^I\Phi] \mid \mathcal{P}_{\bowtie\lambda}[\Phi U^I\Phi] \mid \mathcal{S}_{\bowtie\lambda}[\Phi]$$

where $a \in AP$ is an atomic proposition, $I \subseteq \mathbb{R}_{\geq 0}$ is a nonempty interval, interpreted as a time interval, where $\mathbb{R}_{\geq 0}$ denotes the non-negative reals, $\bowtie \in \{<, \leq, \geq, >\}$ is a comparison operator, and $\lambda \in [0, 1]$ is interpreted as a probability. Csl formulae are evaluated over CTMCs whose states are labelled with the subset of atomic propositions AP that hold true in that state. Atomic propositions

represent elementary assertions that are either true or false in a state (such as *inCriticalSection*, *faultDetected*). As usual, $\Phi_1 \vee \Phi_2$ abbreviates $\neg(\neg\Phi_1 \wedge \neg\Phi_2)$, $\Phi_1 \Rightarrow \Phi_2$ abbreviates $\neg\Phi_1 \vee \Phi_2$, and X and U abbreviate $X^{[0,\infty)}$ and $U^{[0,\infty)}$ respectively.

The interpretation of the formulae of CSL is as follows: a state s satisfies the atomic proposition a if state s is labelled with a, $\wedge$ and $\neg$ have the usual interpretation, while s satisfies $\mathcal{S}_{\bowtie\lambda}[\Phi]$ if the sum of the steady state probabilities, obtained by letting the CTMC evolve from state s, of the states that satisfy Φ is $\bowtie \lambda$. A state s satisfies the "bounded next" formula $\mathcal{P}_{\bowtie\lambda}[X^I\Phi]$ if the probability that the transition taken from s leads to a state satisfying Φ, and that the duration of the transition lies in I, is $\bowtie \lambda$. Finally, a state s satisfies the "bounded until" formula $\mathcal{P}_{\bowtie\lambda}[\Phi_1 U^I \Phi_2]$ if the probability of Φ_2 being true in a future execution from s after d time units have elapsed, where $d \in I$, and where the execution remains in states satisfying Φ_1 until Φ_2 is reached, is $\bowtie \lambda$. Examples of CSL formulae, taken from [5], are the following. The formula $\mathcal{S}_{\leq 10^{-5}}[a]$ is true if the probability of being in a state labelled by the atomic proposition a in steady-state is not greater than 0.00001. The formula $\mathcal{P}_{\leq 0.01}[aU^{[10,20]}b]$ is true if the probability of being in a b-labelled state after between 10 and 20 time units have elapsed, while remaining in a-labelled states before that point, is not greater than 0.01. Finally, the formula $\mathcal{P}_{\geq 0.5}[\neg aU^{[10,20]}\mathcal{S}_{\geq 0.8}[b \vee c]]$ is true if, with probability at least 0.5, we will reach a state between 10 and 20 time units (while avoiding a-states) in which the probability of being in a b- or c-labelled state in equilibrium is at least 0.8. The formal semantics of CSL can be found in [5].

Model checking a CTMC against a CSL formula Φ amounts to computing the set of states $Sat(\Phi)$ such that $s \in Sat(\Phi)$ if and only if Φ is true in state s. Model checking a formula Φ consists of computing the set $Sat(\Phi)$ of states which satisfy Φ by computing the sets of states which satisfy the subformulae of Φ; first, the sets of states satisfying the "shortest" subformulae of Φ (that is, the atomic propositions) are computed, then these sets are used to compute the sets of states satisfying progressively "longer" subformulae. When probability bounds are present in the formula (in the cases of $\mathcal{P}_{\bowtie\lambda}$ or $\mathcal{S}_{\bowtie\lambda}$) the model-checking algorithm requires the computation of transient or steady state probabilities of the original CTMC as well as, possibly, a number of additional CTMCs built through manipulation of the original one [5].

2.2 Three Tools: ETMCC, PRISM, and GREATSPN

The ETMCC Tool. ETMCC is a prototype tool supporting the verification of CSL-properties over CTMCs [12]. Once the rate matrix R and the labelling L (which can be regarded as a function from the set of states to the power set of atomic propositions, so that $L(s) \subseteq AP$ represents the set of atomic propositions true in state s) of a CTMC have been determined, model checking of CSL formulae can take place.

ETMCC has been written in JAVA and provides a simple graphical user interface. The input models for ETMCC are CTMCs, that can be provided in

two different, ASCII-based formats. The set AP of atomic propositions and the labelling L is also specified in an input file (in the publicly available version the cardinality of AP is limited to 67). It is obvious that, for large CTMCs, the two input files should be automatically produced by some higher-level formalism: two tools supporting a high-level formalism that are already interfaced with ETMCC are the process algebra tool TIPP [11] and the SPN tool DaNAMiCS [9].

In terms of solution algorithms ETMCC supports the usual set of choices for Markov chains: Power method, Gauss-Seidel, Jacobi, JOR and SOR, as well as a solution for bounded until U^I and bounded next X^I based on numerical solution of a set of Volterra integral equations [12]. The CTMC is stored in sparse format.

The Prism **Tool.** Prism supports analysis of Discrete Time Markov Chains, Markov Decision Processes, and Csl model checking of CTMCs. An important characteristic of Prism is its ability to work with symbolic data structures like BDDs and MTBDDs [7], which can store the state space of the model efficiently. Three verification options are possible: fully symbolic (infinitesimal generator and probability vector as MTBDDs), fully sparse (infinitesimal generator as sparse matrix and the full probability vector) and hybrid (infinitesimal generator as MTBDDs and the full probability vector). In terms of solution algorithms Prism supports the usual set of choices for Markov chains, including the Power method, Gauss-Seidel, Jacobi, JOR and SOR. The Prism input language is based on the Reactive Modules language [2].

Model checkers only allow to check whether a certain formula is satisfied in a state, or to determine all the states that satisfy a formula. Csl formulae may include a probability bound, which has to be specified by the user. An interesting feature of Prism is its ability to determine the value of this probability if it is left unspecified (by substituting the value with a "?") in the formula.

Prism has a graphical interface for the management of modules and properties and for the visualization of results of multiple experiments. Model-checking analyses can be executed from the graphical user interface or from the command line.

The GreatSPN **Tool.** GreatSPN [10] is a tool for the definition and analysis of GSPN and SWN. Analysis in GreatSPN is supported by structural analysis of the net (computation of P- and T- invariants, deadlocks and traps), reachability analysis (liveness of transitions, ergodicity of the CTMC) and computation of performance indices through CTMC solutions or simulation. GreatSPN has a graphical interface for the definition and analysis, but all solution algorithms can also be launched from the command line thanks to a number of pre-defined scripts. The portion of GreatSPN that we shall use in this work relates to the computation of the state space and of the underlying CTMC.

3 Using PRISM and ETMCC with GREATSPN

As described in Section 2, CSL model checking of CTMCs requires three main ingredients: (1) a model, the underlying semantics of which is a CTMC, (2) a set AP of atomic propositions (simple strings) and a labelling function L from states of the Markov chain to subsets to AP, and (3) a CSL model checker.

CSL model checking of GSPNs requires a slightly modified set of ingredients: (1) a GSPN model, (2) a set of atomic propositions expressed in terms of net elements like the number of tokens in a place, comparison between marking of different places, and enabling degree of transitions, and (3) a CSL model checker.

In the following we illustrate two different approaches to GSPN model checking using preexisting tools: the first one is the interface with PRISM, which is realized at the net level, the second one is the interface with ETMCC, which is realized at the CTMC level.

3.1 GREATSPN and PRISM

The PRISM input language is a state-based language that is built upon modules and variables. The state of the system is a valuation of the variables of each of the system's modules. A module is a set of declarations and commands: each command is of the form GUARD $\longrightarrow$ RATE : UPDATE, where GUARD specifies a logical condition on the system variables, RATE is the rate of the exponentially distributed delay associated to the command, and UPDATE is a set of assignments which specify the new values of the variables in terms of old ones (and a prime is used to distinguish the new value from the old one).

We define a translation to PRISM modules only for SPNs; that is, for GSPNs without immediate transitions (the classical SPN formalism defined by Molloy in [14]). An SPN is defined as (P, T, I, O, W, m_0), where P is the set of places, T is the set of transitions, $I, O : P \times T \to 2^{\mathbb{N}}$ is the set of input and output arcs with associated multiplicity, $W : T \to \mathbb{R}_{\geq 0}$ defines the rate of the exponential distributions associated to transitions, and $m_0 : P \to \mathbb{N}$ describes the initial marking. Due to space reasons we assume the reader is familiar with the SPN semantics.

For a bounded SPN, a unique PRISM module is created using the following rules:

1. For each place $p \in P$ a variable of name P is declared. Letting the bound for place p be denoted by b_p, we can let the range of the variable P be $[0, b_p]$. The variable P is initialized to the value of $m_0(p)$.
2. For each transition $t \in T$ a new command of the form GUARD $\longrightarrow$ RATE: UPDATE is added to the module, where GUARD represents the enabling condition of t, expressed as the logical conjunction of as many predicates as there are input places to t. Each predicate is of the form P $\geq I(p, t)$, where P is the name of the PRISM variable for place p. We set RATE to $W(t)$. Finally, UPDATE represents the modification to the marking due to the firing of t,

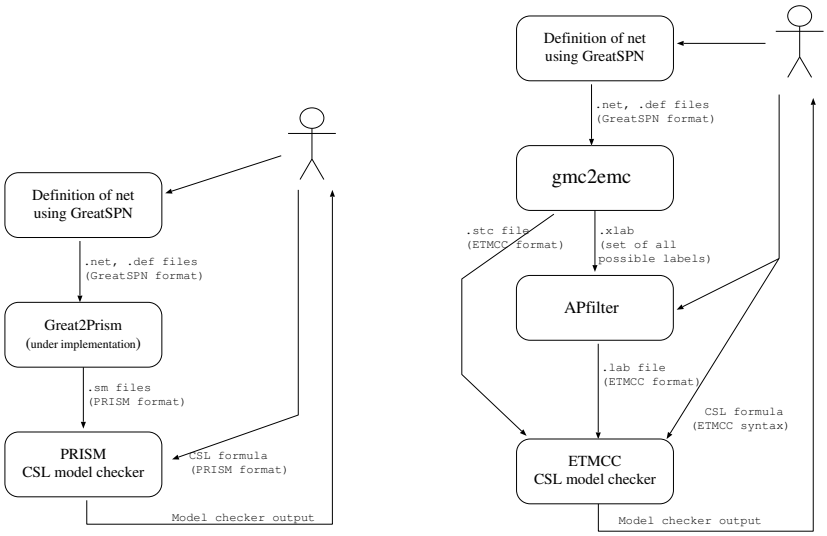

Fig. 1. Using PRISM and ETMCC to model check GreatSPN models

built as the conjunction of $|P|$ assignments, one for each place $p \in P$. The conjunct for place $p \in P$, with associated PRISM variable P, is defined by

$$\text{P'} = \begin{cases} \text{P} + O(p,t) - I(p,t) & \text{if } I(p,t) > 0 \wedge O(p,t) > 0 \\ \text{P} - I(p,t) & \text{if } I(p,t) > 0 \wedge O(p,t) = 0 \\ \text{P} + O(p,t) & \text{if } I(p,t) = 0 \wedge O(p,t) > 0\ . \end{cases}$$

Note that similar translations have been used to obtain PRISM modules of SPN models on the PRISM website [15].

With regard to the atomic propositions, in PRISM the set AP is implicitly defined, by allowing the user to include in a CSL formula any logical condition on the values of the variables. In the proposed translation place names are mapped one-to-one to variable names, and therefore any logical expression on place marking is allowed and is realized trivially.

The formalism of SPN can also be extended with *inhibitor arcs*, for which the presence of a given number of tokens in a place may preclude the firing of an otherwise-enabled transition. For example, inhibitor arcs of degree 1 can be incorporated in our translation in the following manner: for each transition $t \in T$, the enabling condition GUARD is defined as above, but is extended with an additional conjunct of the form P$= 0$ for each inhibitor arc which points at t, where P is the variable associated with the source place of the inhibitor arc.

Figure 1 (left) describes the user point of view of the proposed approach: the GSPN model is defined using the graphical interface of GREATSPN, and structural analysis can be performed for a quick check of model correctness. The output of the GREATSPN interface (two files in ASCII that describe the GSPN) are passed to the translator (that is still under construction), that produces a

PRISM module. CSL formulae are specified by the user through the graphical interface of PRISM using logical expressions on place names.

3.2 GREATSPN and ETMCC

The generation of the ETMCC input model from GREATSPN is simple, thanks to the use of two off-line solvers of GREATSPN: NEWRG for the generation of the reachability graph of the GSPN, NEWMT for the generation of the CTMC underlying the GSPN model, and SEEMTX for the visualization in ASCII of the CTMC itself. The output of SEEMTX is very similar to the .stc input file format of ETMCC, and the translation poses no particular problems.

More delicate is instead the definition of the .lab input file to ETMCC, which lists the set of atomic propositions and, for each CTMC state, the atomic propositions satisfied in that state. An atomic proposition in ETMCC is simply a string (with some syntactical limitations): we have chosen to define the atomic propositions by considering all possible numbers of tokens in places in all the reachable markings of the net, and defining properties for equality, greater than, etc. For example if p is a place in the net with $0, 1$, or 2 tokens, then the set AP will include the strings: peq0, peq1, peq2, pgt0, pgt1, and pgt2. For $i \in \{0, 1, 2\}$, a state is labelled with peq$i$ (pgt$i$, respectively) if the corresponding marking has i tokens in place p (more than i tokens in place p, respectively). However, this approach tends to generate a (usually needlessly) large AP set, because only some of the atomic propositions defined are referred to in a given CSL formula. Also recall that the publicly-released version ETMCC version only allows for a maximum cardinality of 67. Therefore we have built a filter that selects only the atomic propositions used in the formula being checked.

The perspective of the user with regard to the translator is depicted in Figure 1 (right). As in Section 3.1, the user commences by defining the GSPN using GREATSPN. The gmc2emc converter is then used to produce the CTMC file (with extension .stc), and an intermediate file in .lab format (called .xlab) that contains the list of atomic propositions and the association with the states. The converter gmc2emc makes use of the off-line GREATSPN solvers mentioned above. The user then specifies the atomic propositions of interest and the AP-filter module eliminates unnecessary elements from AP.

4 Example of CSL Model Checking

The translators defined in the previous section are now applied to an example taken from the literature: a cyclic multiserver polling system, as for example in [1]-chapter 9. Polling systems comprise a set of stations and a number of servers shared among the stations and that move from station to station. The servers follow a given order and a precise policy determining when and for how long the server should serve a given station before moving to the next one. Figure 2 depicts an SPN model of a polling system consisting of $N = 4$ stations and S servers. In each station i there are K clients that execute the cycle composed by

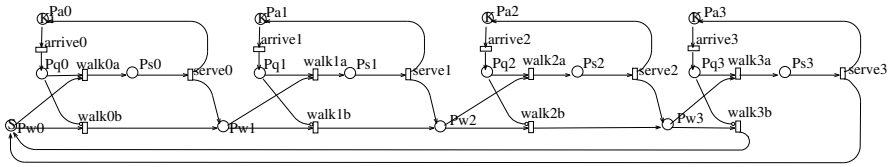

Fig. 2. SPN of a four-stations multiple server polling system

```
// variables: station1
Ps1 : [0..K];
Pw1 : [0..S];
Pa1 : [0..K] init K;
Pq1 : [0..K];

// commands: station1
// of transition walk1a
[] (Pq1>0) & (Pw1>0) -> 1 : (Pq1'=Pq1-1) & (Pw1'=Pw1-1) & (Ps1'=Ps1+1);
// of transition walk1b
[] (Pq1=0) & (Pw1>0) -> 1 : (Pw1'=Pw1-1) & (Pw2'=Pw2+1);
// of transition serve1
[] (Ps1>0) -> 1 : (Ps1'=Ps1-1) & (Pw2'=Pw2+1) & (Pa1'=Pa1+1);
// of transition arrive1
[] (Pa1>0) -> 1 : (Pa1'=Pa1-1) & (Pq1'=Pq1+1);
```

Fig. 3. Fragment of PRISM code for one station

places Pa_i, Pq_i, and Ps_i. A client in Pa_i is doing some local work (transition $arrive_i$ which has single server policy), it then goes to a queue place (Pq_i) where it waits for a server. When a server arrives, one client in the queue is selected and receives service (place Ps_i and transition $serve_i$ which has infinite server policy). A server which has provided service to queue i "walks" to the next queue (queue $(i+1)$ mod 4 since we are assuming a circular ordering), represented by a token in place Pw_i. When the server arrives at station (transition $walk_i a$) it provides service if some client is waiting in the queue; if not it moves on to the next queue (transition $walk_i b$). In Figure 3 we show the fragment of the PRISM code in which the variables and commands of one station are defined. A different translation of the system to a PRISM model can be found at the PRISM webpage.

Our experiments have considered three models. Model A has one client in each station, a single server and all transitions of rate 1. Model B has ten clients in station 0, two clients in the remaining stations, and two servers. All rates in model B are set to 1, apart from the rate of arrival of clients in station 0, which is 0.25, and the rate of arrival for all other stations is set to 0.5. Model C has thirty clients in station 0, four clients in the remaining stations, and three servers. All rates in model C are set to 1, apart from the rate of arrival of clients in station 0, which is 0.4, and the rate of arrival for all other stations is set to 0.25. All transitions in each model have a single server policy. Models A, B and C have 96, 7902 and 360104 states respectively.

We have verified the following properties using PRISM and ETMCC. Note that our atomic propositions are written as comparisons of the number of tokens

in a place with natural numbers: such comparisons can be written in terms of variables in the case of PRISM (for example, P> 0), or as the strings introduced in Section 3.2 in the case of ETMCC (for example, pgt0).

Property (1) - Steady state probability of at least one client in queue 0:

$$\mathcal{S}_{=?}\ [Pq_0 > 0]\ .$$

We are using here the PRISM notation $\mathcal{S}_{=?}$ to indicate that we do not want to check a value, but we ask the tool to compute the steady state probability for all states that verify $Pq_0 > 0$. In ETMCC we have checked instead $\mathcal{S}_{>0}\ [Pq_0 > 0]$; as expected, this property is satisfied in all states in all models, because the models are ergodic. During the computation ETMCC outputs also the aggregated probability for states satisfying $Pq_0 > 0$. This aggregated probability was computed also using GreatSPN, to increase our confidence in the translations.

Property (2) - Absence of starvation (clients waiting in a queue will be served):

$$(Pq_0 > 0 \Rightarrow \mathcal{P}_{\geq 1}\ [true\ U\ Pq_0 = 0])\ .$$

In all models, the property is true in all states reachable from the initial state (including those in which $Pq_0 = 0$, since the second operand of the implication is satisfied in all states of each model). This property that does not require the CTMC solution, and instead relies on reachability analysis of the model's underlying graph.

Property (3) - Probability of service within a deadline: Since all transitions have infinite support and the CTMC is ergodic, then all states will have a non-null probability of service within a deadline, while only the states in which the service is being provided will have a 1 probability. The CSL formula that we have checked is

$$\mathcal{P}_{=?}\ [(Pq_0 > 0 \wedge Ps_0 = 0)\ U^{[0,5]}\ Ps_0 > 0]\ .$$

The output of PRISM and ETMCC lists the probability of satisfying the until formula from each state.

Property (4) - Reproducibility of the initial marking: Since we are working only with reachable states, we can check the simple property

$$\mathcal{P}_{\geq 1}\ [true\ U\ \text{“init”}],$$

where “init” is a logical condition that fully characterizes the initial marking (and it is therefore different for the various systems that we have verified). This property is satisfied by all states in all models.

Property (5) - Reproducibility of the initial marking with a deadline:

$$\mathcal{P}_{=?}\ [true\ U^{[0,10]}\ \text{“init”}].$$

This property is similar to property (3); it is satisfied by all states and the same comments as above apply.

The tools produce the probability of reaching the “init” state from any of the states of the model. This probability is 1 from the “init” state itself and

is instead very low for all other states: for model A, these probabilities for all states are less than 0.01, whereas, after changing the interval to [0,1000], the probabilities of all states are less than 0.25.

Property (6) - Circularity of a server: We wish to check whether a server will present itself more than once at station 0.

$$\mathcal{P}_{\geq 1}\ [G\ (Pw_0 = 1 \Rightarrow \mathcal{P}_{\geq 1}\ [X\ (Pw_0 = 0 \Rightarrow \mathcal{P}_{\geq 1}[true\ U\ Pw_0 = 1])])],$$

where $\mathcal{P}_{\geq 1}\ [G\Phi]$ (read "globally Φ with probability 1") abbreviates the CSL formula $\neg\mathcal{P}_{\leq 0}\ [true\ U\ \neg\Phi]$. The property is satisfied by all states in all models.

Comment. We observed that the speed of the two tools ETMCC and PRISM in obtaining results for our polling system models was comparable. Note that we chose the fully sparse verification engine of PRISM in order to be able to make this comparison; instead, PRISM also supports MTBDD and hybrid verification engines, which can facilitate verification of larger systems. We experimented with the Jacobi and Gauss-Seidel options, and generally found that verification using Gauss-Seidel was more efficient for our models (we terminated the experiments using Jacobi on model C after one hour). Finally, we observed that for models B and C (which both have thousands of states) methods for "filtering" the results of model checking were required, in order to output the probabilities for a small number of states of interest.

5 Conclusions

In this paper we have reported on our investigation to add CSL model checking capabilities to GREATSPN. For the time being the work has concentrated on GSPN models for which two CSL model checkers have been considered: ETMCC and PRISM. For the first one the implementation has been completed and it is available through the first author of this paper: indeed in this case, since the interfacing takes place at the CTMC level, an implementation was necessary to be able to experiment with GSPN of non trivial size. For PRISM instead the implementation is still under definition, although a syntactic translation from SPN to PRISM modules, upon which the implementation can be based, has been defined. The translation can be used to manually transform a SPN into a PRISM module relatively easily.

References

1. Ajmone Marsan, M., Balbo, G., Conte, G., Donatelli, S., Franceschinis, G.: Modelling with Generalized Stochastic Petri Nets. John Wiley (1995)
2. Alur, R., Henzinger, T.A.: Reactive Modules. Formal Methods in System Design **15** (1999) 7–48
3. APNN Web Page. `http://ls4-www.cs.uni-dortmund.de/APNN-TOOLBOX/toolbox_en.html`
4. Aziz, A., Sanwal, K., Singhal, V., Brayton, R.: Model-Checking Continuous Time Markov Chains. ACM Transactions on Computational Logic **1** (2000) 162–170

5. Baier, C., Haverkort, B., Hermanns, H., Katoen, J.-P.: Model-Checking Algorithms for Continuous-Time Markov Chains. IEEE Transactions on Software Engineering **29** (2003) 524–541
6. Chiola, G., Dutheillet, C., Franceschinis, G., Haddad, S.: Stochastic Well-Formed Coloured Nets for Symmetric Modelling Applications. IEEE Transaction on Computers **42** (1993) 1343–1360
7. Clarke, E.M., Fujita, M., McGeer, P., McMillan, K., Yang, J., Zhao, X.: Multi-Terminal Binary Decision Diagrams: An Efficient Data Structure for Matrix Representation. In: Proceedings of the International Workshop on Logic Synthesis (IWLS'93). (1993) 6a:1–15
   Also available in Formal Methods in System Design **10** (1997) 149–169
8. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press (1999)
9. DANAMiCS Web Page.
   `http://www.cs.uct.ac.za/Research/DNA/DaNAMiCS/DaNAMiCS.html`
10. The GreatSPN Tool. `http://www.di.unito.it/~greatspn`
11. Hermanns, H., Herzog, U., Klehmet, U., Mertsiotakis, V., Siegle, M.: Compositional Performance Modelling with the TIPPtool. Performance Evaluation **39** (2000) 5–35
12. Hermanns, H., Katoen, J.-P., Meyer-Kayser, J., Siegle, M.: A Tool for Model-Checking Markov Chains. International Journal on Software Tools for Technology Transfer **4** (2003) 153–172
13. Hillston, J.: A Compositional Approach to Performance Modelling. Cambridge University Press (1996)
14. Molloy, M.K.: Performance Analysis Using Stochastic Petri Nets. IEEE Transaction on Computers **31** (1982) 913–917
15. PRISM Web Site. `http://www.cs.bham.ac.uk/~dxp/prism`
16. UltraSAN Web Site. `http://www.crhc.uiuc.edu/UltraSAN/index.html`