(Stochastic) model checking in GreatSPN

Elvio Gilberto Amparore, Marco Beccuti, Susanna Donatelli

Università di Torino, Dipartimento di Informatica
{amparore,beccuti,susi}@di.unito.it

Abstract. GreatSPN is a tool for the definition and solution of Generalized Stochastic Petri Nets (GSPN). This paper presents the model checking features that have been recently introduced in GreatSPN. Through a new (Java-based) graphical interface for the GSPN model definition, the user can now access the model checkers of three different logics: the classical branching temporal logic CTL, and two stochastic logics, CSL and its superset CSL^{TA}. This allows to integrate easily classical and probabilistic verification. A distinctive feature of the CTL model checker is the ability of generating counter-examples and witnesses. The CTL model checker is symbolic, the implementation being based on the symbolic data structures of the Meddly library [5], developed at Iowa State University, while the CSL^{TA} model checker uses advanced solution methods recently developed for Markov Renewal Processes.

1 Objectives

Generalized Stochastic Petri Nets (GSPNs) [14], extending the PN formalism with stochastic (exponential) delay for transitions, is a modeling formalism very effective in the representation and evaluation of discrete event dynamic systems. GSPNs have been extensively studied, and have been used for studying the behavior of hardware and software system, with very relevant applications in the field of flexible manufacturing systems, business work-flows and biological systems. GSPN have established a bridge between classical formalisms for analysis of concurrency, like classical Petri nets, and classical formalisms for performance evaluation, like queueing networks and Markov chains. As such, the supporting tools have paid particular attention to the integration of qualitative analysis (for classical correctness) with quantitative analysis (for performance evaluation). More recently this duality is well reflected in the advancement in model checking: of temporal logics (for qualitative verification) and of stochastic logics (for probabilistic verification). This paper describes the evolution of the GSPN tool of GreatSPN [3][4] to deal with model checking of temporal and stochastic logics.

GreatSPN tool was first released by the University of Torino in the late 1980's. It was the first GSPN tool with a graphical interface, and the first tool which supported the extension of GSPN to deal with colors. It has been used by several research groups in different universities and/or research centers. GreatSPN implements several efficient analysis algorithms for GSPN and its colored extension, now standardized as Stochastic Symmetric Net [7] (was Stochastic Well-formed nets - SWN - before): GSPN and SSN models can be analyzed through discrete event simulation, or through the generation of the underlying stochastic process (representing the model behavior in time)

and its solution with the computation of standard performance indices. GreatSPN also supports analysis of classical Petri net properties (like P- and T-invariants, check for liveness, deadlock freeness, etc). One of the most characterizing aspects of GreatSPN is its advanced analysis techniques for SSN.

Being a "tool with an history" has some advantages (lot of material, examples and experience around it), but also several drawbacks (software organization that follows old standard, solvers that do not exploit the most recent results on efficient data structure and obsolete Graphical User Interface (GUI)). Two years ago GreatSPN has been upgraded [4] with the introduction of decision diagram (DD) data structures for state space generation, the implementation is based on the DD library Meddly [5], developed at Iowa State University. Being able to compute very large state spaces is important, but it can be only partially useful if no adequate analysis techniques are provided for checking various forms of safety and liveness property, as provided by CTL and LTL model checkers. Being GreatSPN a tool oriented also towards performance evaluation it is also important that the CTL/LTL support is complemented by a probabilistic model checker for stochastic logics. The GreatSPN's GUI currently available, implemented using Motif toolkit, is more than 15 years old. It is not easy to use for newcomers, since it does not meet the modern GUI standards, and it is not easily portable (due to the current limited availability of updated Open Motif libraries).

To cope with the outlined drawbacks in this paper we present an upgrade of GreatSPN which features a new (Java-based) GUI and a model checking facility for CTL (with counterexamples and witnesses), CSL, and CSL^{TA}. This makes GreatSPN an advanced state-of-the-art tool for the integrated analysis of qualitative and stochastic properties of GSPNs.

The structure of the paper follows the indication provided in the supplemental material of the call for papers: Section 2 reports the new functionalities of the tool, Section 3 outlines the architecture and the installation information, Section 4 is an introduction to the tool from an user point of view, while Section 5 compare the behavior of the tool against other two similar model checkers (SMART and MARCIE) and discusses future improvements.

2 Functionality

In this section we give an overview of the GreatSPN's GUI and of the model checking functionalities.

GreatSPN's GUI. GreatSPN is now equipped with a new graphical user interface, designed and implemented to draw GSPN models. The interface is built in Java language, for enhancing portability, and provides a number of new features. It has WYSIWYG editing capabilities, cut/copy/paste, full undo/redo support, interactive editing and token simulation, and other capabilities expected from a modern GUI application.

Figure 2 shows the canvas structure of the new GUI. Editing is based on an SDI approach with multiple nets opened at the same time, shown in the top-left corner. Currently, the editor supports a full language for non-colored GSPN, with immediate/exponential transitions, multiplicity on arcs, weights, named constants, N-server

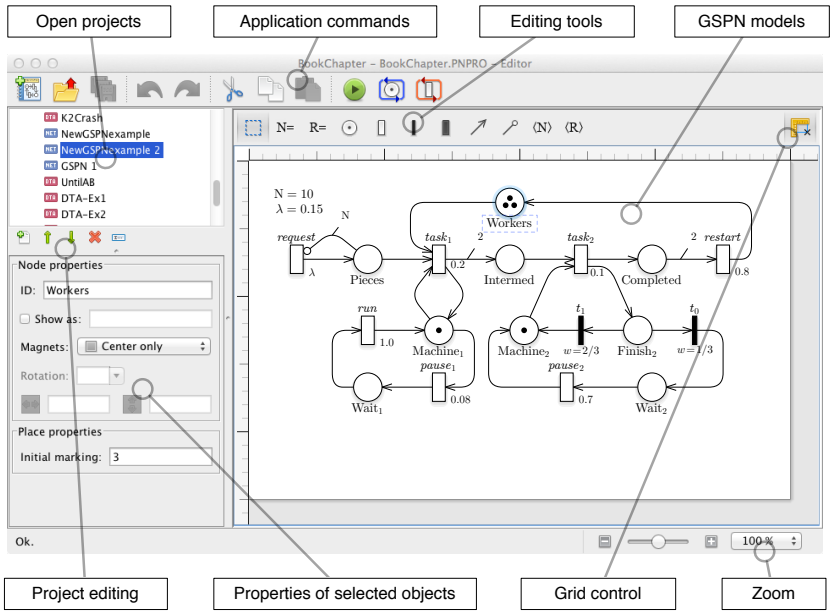


Fig. 1. Canvas structure of the new GreatSPN GUI.

transitions, and so on. Models can be exported in the GreatSPN format (net/def files) which allows one to run the whole set of GreatSPN solvers through command line. Models are drawn with an high quality vector-based rendered with LaTeX labels on objects and PDF printing, for publishing. An example is the GSPN of Figure 3 which has been drawn with the new GUI and exported in PDF format. In addition, the interface has an interactive token game simulation and an interactive visualization of P- and T-semiflows. Since some modeling capabilities are still missing (e.g. colors, result visualization), the old GUI is still provided in the GreatSPN package distribution.

CTL model checking. The CTL model checker of GreatSPN checks CTL formulas for Petri nets with priorities and inhibitor arcs. The model checker is implemented using the symbolic data structures (i.e. Multiway Decision Diagram - MDD, and Multi-Terminal MDD - MTMDD) provided by Meddly [5], an open-source library developed at Iowa State University. While model checking a CTL formula Φ for a GSPN model N of initial marking m_0, the first step is the computation of the set of reachable states (RS) of N: the states are generated and encoded in an MDD with a state space exploration methods based on the well-known saturation algorithm [9]. The MDD has one level per place, the bounds of each place can be taken by default (in which case the, pre-existing, GreatSPN computation of bounds is used, or can be automatically given by the user. The ordering of the places, and therefore of the MDD variables, can be automatically computed (with an ad-hoc heuristic based on P-invariants and clustering of places of common transition) or can be given in input by the user through an external file.

With respect to typical CTL formula a distinctive point is the presence of the three atomic propositions *deadlock*, *ndeadlock* and $en\{t\}$, where t is a transition name. The

first one holds in all markings in which no transition is enabled, the second one in all markings in which at least a transition is enabled, and the last one in all states in which t is enabled. Atomic propositions can also be rather complicated expressions over marking M of places, based on direct comparison between markings (like $M(p_1) \geq M(p_2)$) or on arithmetic expression over markings (like $2 * M(p_1) + M(p_2) = 5$).

The model checker computes the Sat-set of a formula using rather standard symbolic algorithms on decision diagrams [12], and answers "true" if the initial marking is part of the Sat-set . The operators *EF* and *EG* are instead implemented either with the classical algorithm or using a *constrained saturation* algorithm [9].

But the quality of a model checker is not only in its ability in analyzing very large state spaces, but also on how helpful it is in supporting the user in the verification process. It is for this reason that the model checker developed for GreatSPN includes a new symbolic algorithm that computes *tree-like counterexamples* [11] and witnesses: the algorithm, starting from the whole formula, recursively constructs the counter-examples for all its sub-formulas using a depth-first visit. For each sub-formula the Sat set of the states which satisfy the sub-formula negation is computed and stored in the MDD_0. Then, all the states reaching a state in the MDD_0 in i steps are encoded on the MDD_i, so that this set of MDDs can be efficiently visited to generate the shortest counter-example. To the best of our knowledge other model checkers of CTL for Petri nets do not include such feature.

CSL^{TA} model checking. The CSL$^{\text{TA}}$ model checker available in GreatSPN is taken from MC4CSL$^{\text{TA}}$, a probabilistic model checker for Markov chains [1] that has been developed in our research group. In CSL$^{\text{TA}}$ properties have the form $P_{\leq\alpha}(A)$, where A is a deterministic Timed Automaton (DTA), and the property is verified in a state s if the probability of all the CTMC paths, stemming from s and accepted by A, is less than or equal to α. Since the acceptance automaton A allows the use of a clock, the acceptance is based on the actions and states of the paths, as well as the time at which events happen. In MC4CSL$^{\text{TA}}$ the CTMC can be expressed in different formats, in particular those of PRISM and of MRMC [13], and, of course, that of GreatSPN , which is the one that we have included in GreatSPN. The tool implements both forward and backward analysis, which means, in less technical terms and from an user point of view, that the user can ask for whether state s satisfies a formula (analysis forward), or can require the computation of the Sat-set, the set of all states that verify the formula. The model checking of a CSL$^{\text{TA}}$ formula requires the steady state solution of a Markov Renewal Process, as described in [2]. When the CSL$^{\text{TA}}$ model checker evaluates CSL formula its complexity automatically reduces to that of a CSL model checker.

3 Architecture

The architecture of the additional features introduced in the GreatSPN framework is depicted in Fig. 2 where the framework components are presented by rectangles, the component invocations by solid arrows, and the models/data by dashed arrows. For the CTL model checking the new GreatSPN GUI is used to design GSPN models, which are then used in the solution processes calling the two programs: ***struct*** and ***RGMEDD***.

The ***struct*** program generates the input needed for automatically computing the ordering and the bounds of the DD variables used to encode in MDD both the RS and the next state function. Indeed exploiting only the net structural information *struct* generates the net P-semiflows and its place bounds (i.e. upper/lower bound) storing them in the two files *.pinv* and *.bnd* respectively. Its execution is optional since bounds and ordering can be given in input as well.

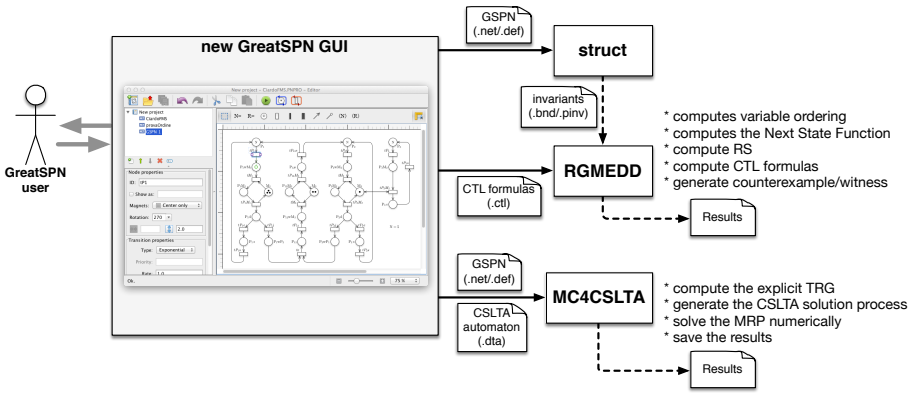


Fig. 2. Architecure of the new GreatSPN GUI for CTL.

The ***RGMEDD*** program is responsible for the CTL model checking and it works by doing 3 steps: encoding of the next state function, RS generation and CTL verification. Encoding of the next state function uses a $2*N$ level MDD (if N is the number of places) while the initial marking is encoded in an N level MDD, reading the *.net* and *.def* files that describe the GSPN. The MDD variables' ordering and their bounds can be specified by the user or automatically derived exploiting the information stored in the file *.pinv* and *.bnd*. The generation of the RS of the net using the saturation algorithm applied on the above generated MDDs. This task is performed using the "simple" interface of Meddly which automatically handles the complex aspects of using DD as the node garbage collection and the operation cache. RS generation is launched by:

ord_rgMEDD "net_directory/net" [-B INTEGER] [-P/-F] [-h INTEGER]

where the first parameter specifies the net name and the optional parameters in square brackets are meant to specify an upper bound for all the MDD variables (*-B*), to enable the automatic generation of a variable ordering exploiting the P-semiflows (*-P*), to specify directly the variable ordering through the file *.place* (*-F*), and to fix the maximum cache dimensions in Bytes (*-h*).

CTL verification assumes the formulas are specified in a *.ctl* file. The full Sat-set is computed, and the output can be limited to true/false (with respect to the initial marking) or to the whole listing of the Sat-set. Optionally the shortest counter-example or a witness can be displayed, if applicable. To make the CTL operators more efficient, they are developed combining the high level MDD operators provided by the "simple" interface of Meddly with more specific ones directly implemented through the Meddly

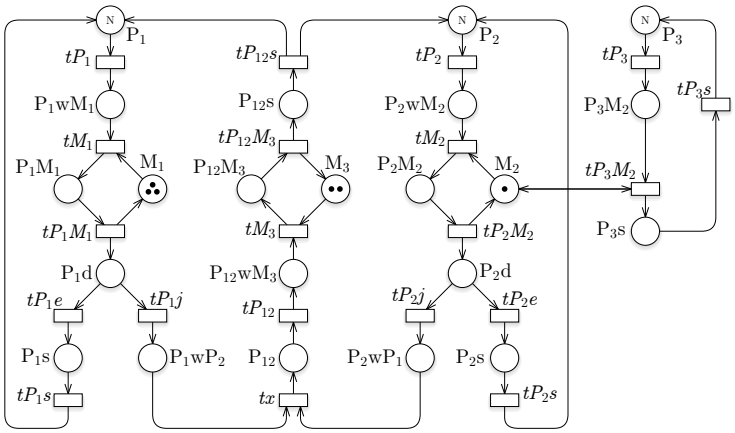


Fig. 3. The FMS model, drawn with the new GUI editor.

expert-level. Indeed, the Meddly expert-level interface allows the user to directly implement her/his own operators and access to the advanced features of the library (like the setting of the storing policy associated with the nodes or the directly access to the unique and computation tables). The CTL model checking can be executed by command-line adding the option *-C* to the RS generation command. Two further options are possible: *-o* to store all the states belonging to the formula Sat set into a file and *-c* to enable the counter-example generation.

Installation instructions The GreatSPN tools and manuals are freely available for non-commercial use at `http://www.di.unito.it/~greatspn/index.html`, while the new GUI interface is available at `http://www.di.unito.it/~amparore/mc4cslta/editor.html`. For the reviewers: a virtual box image with the tools pre-installed is available at `http://www.di.unito.it/~greatspn/VBox`.

4 Use case

This section presents an user point of view of the GreatSPN's model checking functionalities, through the use of the classical FMS net. The objective of an user in using our tool is to acquire confidence on the correct behavior of the FMS, by checking both qualitative and quantitative properties, and then use the model for the computation of performance indices of interest.

The FMS GSPN of Figure 3 has three types of parts, and at any time there are N of each in the system (marking of places P_i). The FMS has three types of machines: M_1, M_2, and M_3. The three machines of type M_1 process only parts of type $P1$, while Machine M_2 can either process a part of type $P2$ or of type $P3$, one at a time. Finished parts of type $P1$ and $P2$ can be also assembled together by the two machines of type M_3. The analysis work-flow illustrated above is done on the FMS model as follows.

The first step in the analysis work-flow is to draw the model and perform basic analysis. The model is drawn in the GUI, and the analysis starts with semiflows com-

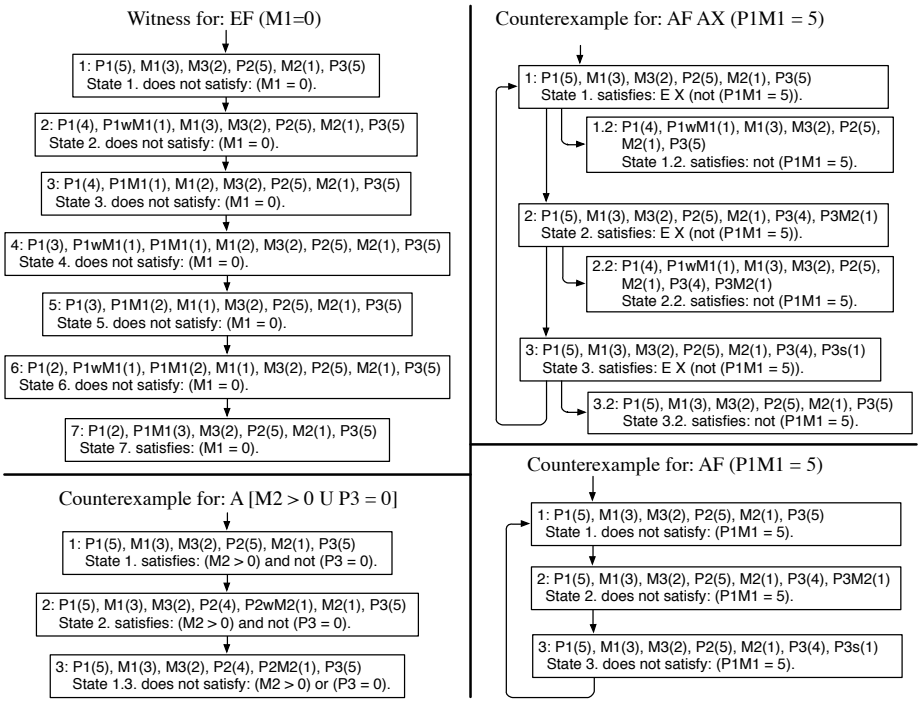
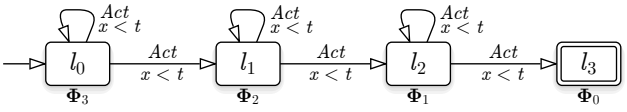


Fig. 4. CTL counterexamples and witnesses.

putation, which computes 6 P-semiflows and 4 T-semiflows that can be displayed in the GUI. The net being bounded, the RS can be computed. The CTL model checking session can then start. Properties peculiar of Petri nets should be checked first, like liveness of transitions, absence of deadlock, presence of a single home space (which ensures ergodicity on the underlying Markov chain), followed by the analysis of FMS-specific properties. For the first category we can check the property deadlock, which result in an empty Sat-set, since the system has no deadlock. We can check the liveness of a transition t through AG EF en(t): the property is true for $t = t_x$, which indicates that the synchronization of the parts to be worked by machine M_2 can always take place. For the second category we have a vast choice of properties. Figure 4 illustrates the result of the analysis for four formulas, resulting in a witness and in three counterexamples, for the net with $N = 5$. With formula EF $M_1 = 1$ we can check that all the machines of type M_1 are actually usable. The formula is true and a witness can be generated, which is a prefix of length 7, depicted in the top left of the figure. To study whether there is the need to have always a M_2 machine available for emptying P_3 we can check A ($M_2 > 0$ U $P_3 = 0$), which is false and generates the very simple counter-example shown in the left bottom part of the figure: in the first two states $M2 > 0$ is true, but $P_3 = 0$ is not, while in the third state neither of the two is true, which falsifies the Until. In CTL counterexamples can be more complex, with looping and branching structure, as shown by the two next examples. The formula AF($P_1 M_1 = 5$) is false (not all parts

of type P_1 can be worked by machine M_1 at the same time). The counter-example takes the form of a loop of three states (bottom right of the figure) in which $P_1M_1 = 5$ is always false, a path that can be considered as a witness of EG ($P_1M_1 \neq 5$). Finally, the top-right of the figure shows a nested counter-example for the nested formula AF AX ($P_1M_1 = 5$): it is loop of three states, and for each state there is a branching prefix of length 2 that falsify AX($P_1M_1 = 5$). It is obvious that, by nesting formulas, arbitrary complex counter-examples or witnesses can be generated. The current implementation prints out the branching counter-examples/witnesses in a textually nested manner, following the same approach of the nuSMV[10] tool: states are indicated with a dotted notation, in which each dot correpsonding to a level in the nesting, and loops are marked by the additional text "loop starts/ends here" properly aligned with respect to testing. The text reported in the boxes is exactly what comes out of the tool, and we have only added the arrows for better reading.

Finally we can integrate the analysis with stochastic model checking. Consider the formula E [$\Phi(3)$ U E [$\Phi(2)$ U E [$\Phi(1)$ U $\Phi(0)$]]], where $\Phi(k)$ is: (P2=N and P3=N and M1=k and M2=1 and M3=2), which is true in the initial marking and states that there are executions that use one after the other all the instances of M_1 without ever touching the other machines and the other parts.



		P>0.2 until_ab(1,10 | True, P1M1=2)			P>0.2 nested_untils(20 | Phi(3) to Phi(0))		
N	States	Prob	Time	Memory(KB)	Prob	Time	Memory(KB)
2	3444	0.4979	0.08	5672	does not apply		
3	48590	0.8307	2.13	50143	1,69E-04	2.38	3665
4	438600	0.9540	27.56	480370	1,46E-04	25.69	26865

Fig. 5. Reported results for the CSL^{TA} model checker.

To check how probable this (potentially unique) behavior is in a real system we can verify its stochastic counter part. The FMS model is enriched with an exponential distribution of delays and we check the formula $P_{\leq\alpha}(A)$, where A is the DTA shown in Figure 5, top, that accepts only the paths that satisfy the nested CTL Until above, with a timing constraint (if $t < \infty$) on the happening of the events. The same figures reports also the computed probabilities (if greater than 0.2 the property is satisfied) for different initial markings. Note that stochastic model checking is intrinsically much more expensive than CTL one and only relatively small CTMCs (around 10^6 states) can be checked.

5 Comparison

There are other tools that do CTL model checking for Petri nets, in particular SMART[8], MARCIE [15] and PNxDD follow a similar approach, being all based on various forms of DD. We have decided to consider here only SMART[8], which is strictly related to

FMS		RGMEDD				SMART 1.1				MARCIE 2013/10/17			
N	|RS|	nodes	peak	time	mem(KB)	nodes	peak	time	mem(KB)	nodes	mem-dd	time	mem(KB)
5	2,89E+06	224	328	0.01	2760	224	306	0.06	4,376	224	1	0.05	115,680
10	2,50E+09	579	908	0.02	3008	579	781	0.10	4,416	579	1	0.06	117,532
20	6,03E+17	1739	2799	0.11	3908	1739	2331	0.17	4,944	1739	1	0.09	118,328
50	4,24E+22	8819	15241	1.12	7640	8819	11781	0.57	14,112	8819	1	0.20	125,188
100	2,70E+26	32619	56766	8.00	21972	32619	43531	2.40	72,200	32619	1	0.70	158,760
200	1,95E+30	125219	218731	60.80	91856	125219	167031	16.53	462,820	125219	2	3.77	392,520

FMS	AG (M1 = 0 -> P1wM1 < N)			EF [M1 = 0]			AF AX (P1M1 = N)			A [M1 > 0 U P1 = 0]		
N	RGMEDD	marcie	Smart	RGMEDD	marcie	Smart	RGMEDD	marcie	Smart	RGMEDD	marcie	Smart
5	0.001	0.04	0.071	0.001	0.07	0.015	0.001	0.06	0.029	0.001	0.03	0.069
10	0.014	0.06	0.199	0.008	0.05	0.024	0.004	0.07	0.054	0.005	0.04	0.113
20	0.047	0.08	0.264	0.025	0.11	0.038	0.011	0.08	0.130	0.019	0.07	0.355
50	0.563	0.25	0.534	0.238	0.37	0.143	0.096	0.23	0.861	0.844	0.28	4.375
100	3.834	1.00	1.107	1.637	1.67	0.698	0.587	0.98	6.234	2.634	2.34	50.712
200	27.632	6.78	2.896	11.599	9.66	4.417	4.094	8.59	45.265	15.844	21.26	608.382

Philosophers-PT-*N*				RGMEDD					
				Reachability set				E [(Eat_1 = 0) U (Eat_2 = 1)]	E G (Eat_1 = 1)
N	|RS|	Places	Transitions	nodes	peak	time	mem(KB)	time	time
5	243	25	25	226	267	0.02	3,232	0.001	0.001
10	59049	50	50	242	288	0.02	3,616	0.001	0.001
20	3.48678e+09	100	100	642	745	0.04	5,224	0.003	0.004
50	7.17898e+23	250	250	4008	4258	0.16	10,216	0.025	0.094
100	5.15378e+47	500	500	15352	15848	0.51	19,924	0.088	0.039
200	2.65614e+95	1000	1000	20800	21808	2.34	38,848	0.168	0.270

Fig. 6. Experiments with FMS and Philosophers.

the Meddly library and MARCIE [15], which performed among the best tools at the latest Petri net model checking contest and to leave a throughout comparison to the next contest. For both tools we have used the version down-loadable from their web sites in December 2013. The comparison is for state space generation and CTL model checking only, since only GreatSPN supports counter-example generation and CSL^{TA} model checking. All experiments were conducted on an Intel i7 with 8 GB of RAM.

The two top tables of Figure 6 shows a comparison of the tools performances on the FMS model. Results for RS generation (top table) are consistent in terms of number of states and even in the final number of DD nodes (although the DD types are not exactly the same), peak number of nodes is not shown for MARCIE since it is not reported by the standard tool execution. The tools perform rather similarly for up to $N = 20$ (about 10^{17} states) while for bigger state spaces MARCIE shows better performances. All tools use the same order of variables, since, upon our tests, it is the best order for all the three tools. For model checking we have considered 4 different formulas, again for "not too big" models they all perform compute the Sat-set in very little time (always less than a minute but for a single case of Smart). We have carefully chosen the place involved in the atomic propositions, since GreatSPN is not too sensitive to this aspect, but the CTL model checker of MARCIE can perform arbitrarily bad depending on the position of the places in the ordering, a behavior that is under fixing in the tool and on which we could report in the camera ready, in case of acceptance.

For GreatSPN we have also reports the experiments on the philosopher model (lower table in Figure 6), since it is an example of nets in which the increase in the RS size is connected with an increase in the number of places and transitions. Again, we can generate very large state spaces in very little time and memory.

The new GreatSPN is still an evolving tool, in particular we are planning to integrate in the new graphical interface the whole set of solvers available in the previous GUI, while it is already complete the full graphical definition of DDTA and its integration in the GUI. We also plan to work more on the graphical rendering of the counterexamples, possibly with a display of the simulation traces in the GUI. CSL^{TA} model checking can only be applied to CTMC of at most a few million states: to mitigate this problem we are developing an export facility towards Cosmos [6], a stochastic model checker based on Monte Carlo simulation

References

1. Amparore, E., Donatelli, S.: Improving and assessing the efficiency of the $MC4CSL^{TA}$ model checker. In: EPEW 2013 (2013)
2. Amparore, E.G., Donatelli, S.: A component-based solution for reducible markov regenerative processes. Performance Evaluation 70(6), 400 – 422 (2013)
3. Baarir, S., Beccuti, M., Cerotti, D., De Pierro, M., Donatelli, S., Franceschinis, G.: The GreatSPN tool: recent enhancements. SIGMETRICS Perform. Eval. Rev. 36(4), 4–9 (2009)
4. Babar, J., Beccuti, M., Donatelli, S., Miner, A.S.: GreatSPN Enhanced with Decision Diagram Data Structures. In: Lilius, J., Penczek, W. (eds.) Petri Nets. Lecture Notes in Computer Science, vol. 6128, pp. 308–317. Springer (2010)
5. Babar, J., Miner, A.: Meddly: Multi-terminal and edge-valued decision diagram library. In: Quantitative Evaluation of Systems, International Conference on. pp. 195–196. IEEE Computer Society, Los Alamitos, CA, USA (2010)
6. Ballarini, P., Djafri, H., Duflot, M., Haddad, S., Pekergin, N.: COSMOS: a statistical model checker for the hybrid automata stochastic logic. In: (QEST'11). pp. 143–144. IEEE Computer Society Press, Aachen, Germany (Sep 2011)
7. Chiola, G., Dutheillet, C., Franceschinis, G., Haddad, S.: Stochastic well-formed coloured nets for symmetric modelling applications. IEEE Trans. on Comp. 42(11), 1343–1360 (1993)
8. Ciardo, G., Jones, III, R.L., Miner, A.S., Siminiceanu, R.I.: Logic and stochastic modeling with smart. Perform. Eval. 63(6), 578–608 (Jun 2006)
9. Ciardo, G., Lüttgen, G., Siminiceanu, R.: Saturation: an efficient iteration strategy for symbolic state space generation. In: (TACAS), LNCS 2031. pp. 328–342. Springer-Verlag (2001)
10. Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: NuSMV 2: An OpenSource Tool for Symbolic Model Checking. In: Computer Aided Verification, LNCS, vol. 2404, pp. 359–364. Springer Berlin (2002)
11. Clarke, E., Jha, S., Lu, Y., Veith, H.: Tree-like counterexamples in model checking. In: Logic in Computer Science, 2002. Proceedings. 17th IEEE Symposium on. pp. 19–29 (2002)
12. Emerson, E., Clarke, E.M.: Using branching time temporal logic to synthesize synchronization skeletons. Science of Computer Programming 2(3), 241 – 266 (1982)
13. Katoen, J.P., Zapreev, I.S., Hahn, E.M., Hermanns, H., Jansen, D.N.: The ins and outs of the probabilistic model checker MRMC. In: Proc. QEST'09 (2009)
14. Marsan, M.A., Balbo, G., Conte, G., Donatelli, S., Franceschinis, G.: Modeling with Generalized Stochastic Petri Nets. J. Wiley, New York, NY, USA (1995)
15. 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. pp. 91–100 (2011)