A CTL Model Checker for Stochastic Automata Networks[⋆]

Lucas Oleksinski, Claiton Correa, Fernando L. Dotti, and Afonso Sales [⋆⋆]

PUCRS - FACIN, Porto Alegre, Brazil
{lucas.oleksinski, claiton.correa}@acad.pucrs.br,
{fernando.dotti, afonso.sales}@pucrs.br

Abstract. Stochastic Automata Networks (SAN) is a Markovian formalism devoted to the quantitative evaluation of concurrent systems. Unlike other Markovian formalisms and despite its interesting features, SAN does not count with the support of model checking. This paper discusses the architecture, the main features and the initial results towards the construction of a symbolic CTL Model Checker for SAN. A parallel version of this model checker is also briefly discussed.

1 Introduction

Stochastic Automata Networks (SAN) was proposed by Plateau [12], being devoted to the quantitative evaluation of concurrent systems. It is a Markovian formalism that allows modeling a system into several subsystems which can interact with each other. Subsystems are represented by automata and interactions by synchronizing transitions of cooperating automata on same events. Dependencies among automata can also be defined, using functions. Functions evaluate on the global state of the automata network and can be used to specify the behavior of specific automata. The use of functions allows the description of complex behaviors in a very compact way [1]. Quantitative analysis of SAN models is possible using specialized software tools (*e.g.*, PEPS [13] or SAN Lite-Solver [14]), fundamentally allowing one to associate probabilities to the states of the model, using a steady state or transient analysis.

While developing models for involved situations it is highly desirable to reason about their computation histories and thus model checking becomes important. Indeed, many formalisms for quantitative analysis count with the support of specialized model checking tools. In the context of CTMC-based model checking, we can mention PRISM [8], SMART [4] and CASPA [7]. Such support however is lacking for SAN. In this paper we report our results towards the construction of the first SAN model checker. In this initial version, the tool is restricted to CTL model checking opposed to the stochastic verification as offered by the aforementioned tools.

[⋆] Paper partially sponsored by CNPq (560036/2010-8) and FAPERGS (PqG 1014867).
[⋆⋆] Afonso Sales receives grant from PUCRS (Edital 01/2012 – Programa de Apoio à Atuação de Professores Horistas em Atividades de Pesquisa na PUCRS).

2 Tool overview

Fig. 1 illustrates the main processing steps of the SAN model checker. It has as input a model written in the SAN modeling language [13], a CTL (*Computation Tree Logic*) property, and an additional information if a witness or a counterexample to the property is desired. As output it offers the answer whether the property is true or false, and a witness or counterexample as chosen. The tool supports the standard CTL where atomic propositions are assertions about the global state of the automata network according to the SAN language [13].

The compilation of the SAN model generates a Markovian descriptor which is used as the system transition relation, *i.e.*, a set of tensors which operated by generalized Kronecker algebra allows the achievement of next states. The initial states of the model are those considered as reachable in the reachability declaration of the SAN model. Multi-valued Decision Diagrams (MDD) are used to encode the Reachable State Space (RSS) of the SAN model, which is calculated using an extension [15] of the saturation based approach [3]. The satisfaction sets calculation (SAT in Fig. 1) follows a breadth-first search algorithm. During this process, the RSS is labelled with all subformulas of the input formula.

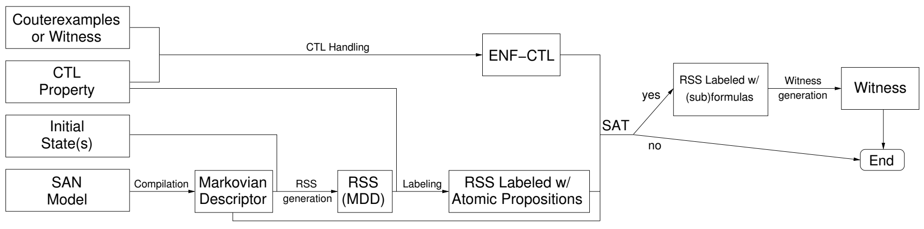

Fig. 1. The tool architecture.

Whenever a counterexample is desired, the tool negates the input formula to generate a witness. The witness generator supports ENF-CTL operators and generates trace structured witnesses. To enrich witness information, whenever a branching is avoided the respective state of the trace is annotated with the subformula that holds from that state.

A parallel approach was proposed that replicates the entire RSS and assigns specific partitions of the state space to be computed by different nodes. Each node may locally compute successor states even these cross partition borders, without requiring communication. Communication is only required for fix-point calculation, which is executed as rounds of synchronization between nodes.

3 Experiments

We report CTL model checking results[1] on both sequential and parallel implementations of the model checker through set of experiments with two different models: the dining philosophers (DP) problem [15] and a model for an *ad hoc*

[1] As mentioned, our tool does not perform stochastic verification and thus numerical analysis is not carried out.

wireless network protocol (WN) [6]. For the DP model, starvation, mutual exclusion, deadlock presence and deadlock absence were checked for model variations with and without deadlock, respectively. Considering the DP model with 15 philosophers, corresponding to RSS of 470,832, all mentioned properties needed about 500 MB memory and 240 CPU seconds (using one core of a Intel Xeon Quad-Core E5520 2.27 GHz machine). For some properties, such as deadlock absence, the tool allows verification of a model with 20 philosophers which has 38,613,965 reachable states requiring around 600 MB memory and 1,650 CPU seconds. We experienced the parallel version in a cluster with 15 processors for the same DP model above with 15 philosophers. The worst speed-up took place with the "deadlock absence" property and the best speed-up took place with the "starvation" property, with speed-ups of 6 and 11, respectively. The verification of the deadlock absence property took a peak of 330 MB *per* node memory while the sequential solution took 500 MB. The starvation property took a peak of 140 MB *per* node memory while the sequential execution took 660 MB.

The WN model was build to obtain the end-to-end throughput traversing a route of *ad hoc* nodes, taking into consideration the interference range among nodes. Thus the model is build such that no two interferring nodes transmit at the same time, resulting in no packet losses. Properties assuring this behavior have been shown. With 28 *ad hoc* nodes the model resulted in a RSS of 568,518 and the property assuring that no two interfering nodes transmit at the same time required verification time around 130 CPU seconds and 361.46 MB memory. Using the parallel approach for the same WN case, for the same property as above reported, models with 24, 26 and 28 *ad hoc* nodes were verified with 15 processors, leading to speed-ups of 4.91; 4.40 and 7.40, respectively. For models with 28 *ad hoc* nodes, the verification had a peak of 289.08 MB *per* node memory while the sequential solution 361.46 MB.

To assess the tool correctness we have carried out experiments with the NuSMV tool. More specifically, a set of SAN models have been translated to the NuSMV, generating transition systems equivalent to the SAN model's underlying Markov Chains, and have been checked for the same CTL properties, leading to same results.

4 Conclusions and Future Works

In this paper we presented, at the authors' best knowledge, the first tool for model checking SAN models. We have discussed its key features, performance results and also initial results on a parallel version. As a first version of the SAN model checker, it has shown coherent results leading to a high confidence in its correctness, however with low performance. Even considering hardware differences, the results reported by PRISM [2], CASPA [7] and SMART [5] can be clearly considered much superior.

In this version we have adopted a Kronecker-based representation of the transition relation due to the usage of functional elements in the transition matrices which are necessary to represent SAN abstractions. The use of Kronecker representation has a considerable impact since the computation of next states implies that several tensors have to be operated, in a meaningful order, according to the number of submodels (automata) and synchronizing events. Moreover, whenever functions are used, they have to be evaluated in this process. In contrast, decision diagram based representation [11, 10] would result in a more direct computation of transitions. This aspect is to be addressed in future works. Another aspect that we want to address is the use of saturation based model checking algorithms. A first step in this directed has been made in [15] where the reachable state space generation of SAN models is computed in a *saturated* way using both decision diagrams and Kronecker representations. Related works using decision diagrams, such as in [9], can contribute in relation to this aspect and must be more closely investigated.

References

1. L. Brenner, P. Fernandes, and A. Sales. The Need for and the Advantages of Generalized Tensor Algebra for Structured Kronecker Representations. *Int. Journal of Simulation: Systems, Science & Technology (IJSIM)*, 6(3-4):52–60, 2005.
2. PRISM (Probabilistic Model Checker). http://www.prismmodelchecker.org/.
3. G. Ciardo, G. Lüttgen, and R. Siminiceanu. Saturation: An Efficient Iteration Strategy for Symbolic State-Space Generation. In *TACAS*, volume 2031 of *LNCS*, pages 328–342. Springer, 2001.
4. G. Ciardo, A.S. Miner, and M. Wan. Advanced features in SMART: the stochastic model checking analyzer for reliability and timing. *SIGMETRICS Performance Evaluation Review*, 36(4):58–63, 2009.
5. G. Ciardo, Y. Zhao, and X. Jin. Ten Years of Saturation: A Petri Net Perspective. *Transactions Petri Nets and Other Models of Concurrency*, 5:51–95, 2012.
6. F.L. Dotti, P. Fernandes, A. Sales, and O.M. Santos. Modular Analytical Performance Models for Ad Hoc Wireless Networks. In *WiOpt 2005*, pages 164–173.
7. M. Kuntz, M. Siegle, and E. Werner. Symbolic Performance and Dependability Evaluation with the Tool CASPA. In *FORTE Workshops 2004*, volume 3236 of *LNCS*, pages 293–307, Toledo, Spain, 2004. Springer.
8. M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of Probabilistic Real-time Systems. In *CAV'11*, volume 6806 of *LNCS*, pages 585–591. Springer-Verlag, 2011.
9. K. Lampka and M. Siegle. Activity-local symbolic state graph generation for high-level stochastic models. In *13th MMB*, pages 245–264. VDE Verlag, 2006.
10. K. Lampka and M. Siegle. Analysis of Markov reward models using zero-suppressed multi-terminal BDDs. In *VALUETOOLS*, page 35, 2006.
11. A.S. Miner and D. Parker. Symbolic Representations and Analysis of Large Probabilistic Systems. In *Validation of Stochastic Systems*, volume 2925 of *LNCS*, pages 296–338. Springer, 2004.
12. B. Plateau. On the stochastic structure of parallelism and synchronization models for distributed algorithms. In *ACM SIGMETRICS Conf. on Measurements and Modeling of Computer Systems*, pages 147–154, Austin, USA, 1985. ACM Press.
13. PEPS Project. http://www-id.imag.fr/Logiciels/peps/userguide.html.
14. A. Sales. SAN lite-solver: a user-friendly software tool to solve SAN models. In *SpringSim (TMS-DEVS)*, pages 44:9–16, Orlando, FL, USA, 2012. SCS/ACM.
15. A. Sales and B. Plateau. Reachable state space generation for structured models which use functional transitions. In *QEST'09*, pages 269–278, Budapest, Hungary.