SAN LITE-SOLVER: a user-friendly software tool to solve SAN models

Afonso Sales
Pontifícia Universidade Católica do Rio Grande do Sul
Avenida Ipiranga, 6681 – 90619-900 – Porto Alegre – RS – Brazil
afonso.sales@pucrs.br

Keywords: structured Markovian models, Stochastic Automata Networks, MDD, software tool, numerical solution.

Abstract

Structured Markovian models are widely used to map and analyze the behavior of complex systems. However, the modeler must frequently need a deeply knowledge about specialized tools or limitations imposed on the solution of their models. This paper presents an easy and practical software tool, called SAN LITE-SOLVER, that applies the Power method to solve a Stochastic Automata Network (SAN) model, using a standard Multi-valued Decision Diagram (MDD) structure to compute and to store the model's reachable state space (RSS) and a Harwell-Boeing format (HBF) matrix which represents the underlying Markov chain (MC). The performance analysis of this new tool (in terms of memory used and CPU time to solve models) is presented and compared to the current approach used to solve SAN models.

1. INTRODUCTION

Markov Chains (MC) [1] is one of the most popular formalisms used to analyze and to study the system behavior of different kind of applications in several domains, such as chemistry, social science, biology, physics, computer science, to cite a few. Nevertheless, beyond its simplicity to model a system using a straightforward approach based in simple primitives (*states* and *transition* between states), depending the size of the modeled system, a Markovian model easily faces the well-known *state space explosion problem*. Moreover, this mapping task can be feasible for models with a few hundred states, but it becomes impracticable for large models (*i.e.*, with hundreds of thousand states).

In order to mitigate this mapping task, structured Markovian formalisms have emerged as another possibility to deal with this limitation, providing a compact description of a large system by the description of its subsystems and their correlation. Structured Markovian formalisms allow a high-level description of a system in a such way that the system abstraction may be done in a more intuitive manner.

However, this high-level description usually comprehends a set of *invalid* (or *unreachable*) states that are invalid combinations of *valid individual states*. This problem has been regarded in the study of several structured Markovian formalisms, such as *Performance Evaluation Process Algebra* (PEPA) [2], *Stochastic Petri Nets* (SPN) [3], and *Stochastic Automata Networks* (SAN) [4].

Many times structured Markovian models have the combination of all individual states, *i.e.*, the *product state space* (PSS), much larger than the (valid or) *reachable state space* (RSS). This kind of phenomenon is much common in structured formalisms due to few occurrences of independent transitions in the subsystems combined of frequent synchronizations between them. Hence, for models where RSS $\ll$ PSS, even using specialized numerical methods to deal with structured representations, the solution of a structured model can become impracticable in terms of CPU time, due to the treatment of a large set of invalid (*unreachable*) states.

A constant challenge of structured modeling is to take advantage of the power of high-level description of problems, providing efficient solution methods. Following this idea, an interesting alternative is to use, for instance, solution methods that deal only with the *reachable state space* of a structured model instead of a structured solution approach, such as *Markovian Descriptor* in SAN or *Matrix Diagrams* in SPN. However, the discovery of these reachable states is not a simple task to be performed. *Multi-valued Decision Diagrams* (MDD) [5] are compact structures that allow to store and to manipulate large sets of data, such as large amounts of states. MDDs have been efficiently employed for generating and manipulating the reachable state space of models described by the SAN [6] and SPN [7] formalisms.

This paper presents a user-friendly software tool, called SAN LITE-SOLVER, that computes the steady-state probability of a model described by the SAN formalism, using a MDD structure to store and to manipulate the model's reachable state space. The main aim of this tool is to make as simple as possible the interaction between user and tool, as well as to provide an efficient solution for large models (*i.e.*, models with a large number of reachable states). It is important to keep in mind that the ideas applied in the design of SAN LITE-SOLVER can also be employed to other formalisms, such as STDEVS [8], for obtaining the probability spaces in an efficient way.

The remainder of this paper is organized as follows. In Section 2 it is briefly presented the SAN formalism and its main

modeling features, as well as the current approach used to solve a SAN model. Section 3 presents SAN LITE-SOLVER showing an overall solution scheme used by this tool and the tool usage with some examples as well. In Section 4, it is presented some tool performance analysis, comparing its performance (*i.e.*, memory cost and CPU time) with the current approach used to solve a SAN model. Finally, Section 5 summarizes the contribution of this paper and suggest future work to improve the proposed software tool.

2. STOCHASTIC AUTOMATA NETWORKS

Stochastic Automata Networks (SAN) is a structured formalism originally proposed by Plateau [4] and it provides a high-level abstraction to represent continuous and discrete-time Markovian models [9, 10]. The main idea of this formalism is to model a complex and large system by a set of "*small*" subsystems, *i.e.*, to model a system composed of modules with *independent behaviors* and *occasional interdependencies*. Beyond an easy and intuitive model mapping, one of the advantages of using the SAN formalism is to provide a compact storage and efficient solution of *large systems*, mitigating the state space explosion problem [11]. This compact representation of a SAN model is described by an tensor algebra formula known as *descriptor* [9, 4], which represents the underlying Markov chain.

Each subsystem of a SAN model is represented by a *stochastic automaton*. The *transitions* between *states* of each automaton represent the transitions of a stochastic process in continuous-time or discrete-time considering, respectively, an exponential or geometric distribution.

As the SAN formalism is based on the Markov Chains (MC) formalism [1], a continuous-time SAN model has an underlying continuous-time Markov chain (CTMC), whereas a discrete-time SAN model has an underlying discrete-time Markov chain (DTMC). In the context of this paper, only continuous-time SAN models[1] are considered.

There is a wide scope of SAN modeling applications mainly focused on: (i) the evaluation of parallel and distributed computer systems [12], as well as applied to QoS assessment in multi-tier web services [13]; (ii) the analytical modeling of *ad hoc* wireless networks [14]; (iii) the behavior of concurrent processors, extracting performance and reliability indices of some parts of the Linux scheduling algorithm for NUMA machines [15]; (iv) modeling of fault-tolerant systems [16]; and (v) the performance analysis of software development teams in globally distributed projects [17].

2.1. Automata

A stochastic automaton is a mathematical model of a system that has discrete input and outputs. The system can remain in any possible *state* or internal configuration. The current state of a system contains all information in relation to the previous inputs and also contains all relevant information to determine the future (next) state of the system (*i.e.*, describing the system behavior) [18].

Based on this property, it is possible to describe a stochastic automaton as a *set of states* and a *set of transitions* between states [19]. The use of the term *stochastic* related to the automaton means that the treatment of the time is related to *random variables*, which can consider an *exponential distribution* in continuous-time or *geometric distribution* in discrete-time.

In a graphical point of view, a *stochastic automata networks* can be represented by a *set of directed graph*, where each graph is associated to an automaton. The *nodes* of a graph represent the *states* of an automaton and the *arrows* (or *directed edges*) between states represent the *transition* from one state to another (see Figure 1).

Local states of a system modeled by the SAN formalism are the individual states of each automaton of the model. And *global states* are composed of the set of local states of each automaton of a SAN model. The change in a global state is determined by the change of one or more local states. Transitions allow the changes between states and they are triggered by the occurrence of *events*. Each transition has one or more associated events.

Figure 1 presents a SAN model[2] with two automata ($A1$ and $A2$), where $A1$ has three local states (F, G, H) and $A2$ has also three states (X, Y, Z). This model has then nine global states: FX, GX, HX, FY, GY, HY, FZ, GZ, HZ.

2.2. Events

Events are responsible for triggering transitions that change the global state of the model. One or more events can be associated to a single transition. Each event must have an *occurrence rate* and a routing probability[3]. The occurrence rate and the routing probability may have a *constant* or *functional* value (*i.e.*, the rate value may vary in function to the state of other automata). This non-determinism related to the occurrence of different events associated to a same transition can be treated by a Markovian process, *i.e.*, all enabled events can occur and their occurrence rate define the frequency of occurrence of these events.

There are two types of events: *local* or *synchronizing*. *Local events* change the local state of *only one automaton*. This kind of event is used to characterize the *independent* behavior of an automaton. *Synchronizing events* change the local state of *two or more automata* simultaneously, *i.e.*, the occurrence

[1] See [10] for a formal description about discrete-time SAN models.

[2] Examples of description of SAN models can be found in the PEPS webpage at *http://www-id.imag.fr/Logiciels/peps/index.html*.

[3] The absence of probability is accepted if only one transition can be fired by an event.

of a synchronizing event on one automaton *obliges* the occurrence of this same event on all corresponding automata. This kind of interaction between automata describes the relation of interdependency between them.

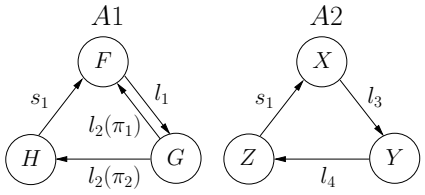
Both local and synchronizing events may have different routing probabilities associated to a same event. The routing probabilities are used to determine in which *proportion* the possible transitions can be fired. For instance, in Figure 1, the occurrence of event l_2 with rate β changes automaton $A1$: from state G to F with probability equal to π_1; or state G to H with probability equal to π_2 (where $\pi_1 + \pi_2 = 1$).

2.3. Function Description

The interaction between automata can also be expressed by *functions*. Functions can be associated to rates or routing probabilities. In this case, the rate or the routing probabilities of an event can assumed different values in function to the state of other automata. The usage of *functional rates* or *functional probabilities* can also be employed by both synchronizing or local events.

In Figure 1, note that the rate of event l_4 is not a constant rate, but a *functional rate* expressed by function f. The interpretation of a function in the SAN formalism can be viewed as the evaluation of a non-typed expression, where each comparison is evaluated to *true* (value 1) or *false* (value 0). In this example, the occurrence of event l_4 changes automaton $A2$ from state Y to Z with rate equal to δ if automaton $A1$ is in state H. Otherwise, l_4 does not occur if $A1$ is not in state H (*i.e.*, the expression of function f is evaluated to *false*, determining a *zero rate* and hence the event does not happen).

Type	Event	Rate
loc	l_1	α
loc	l_2	β
loc	l_3	γ
loc	l_4	f
syn	s_1	ω



$f = (state\ A1 == H) \times \delta$

$\pi_1 = 0.6$
$\pi_2 = 0.4$

Figure 1. An example of a SAN model.

The usage of functions is a powerful feature of the SAN formalism, since it allows to represent very complex behaviors of a system in a very compact way.

2.4. Reachability

Given an initial state of a structured model, the computation of all reachable states of the model can not be a simple task. The product state space (PSS) of a SAN model is easily computed by the cartesian product of all local states of all automata. The model presented in Figure 1 has a PSS equals to nine (*i.e.*, $3 \times 3 = 9$) states. On the other hand, the reachable space state (RSS) can be computed by the successive firing of events from an initial state. In this example (Figure 1), given as initial state FX, performing successive firing of events, the model's RSS is equal to: FX, GX, HX, FY, GY, HY, HZ. Note that states FZ and GZ are never reachable starting to state FX due to the nature of the events.

2.5. Current approach to solve SAN models

Solving a SAN model is not a simple task, and it is needed to use specialized software tools clearly designed to deal with the (Kronecker) structure of a SAN model. *Performance Evaluation of Parallel Systems* (PEPS) [20] is the current software tool used to numerically solve and analyze a SAN model. PEPS delivers stationary and transient exact solutions and it also provides solution using advanced iterative [21, 9] and simulation [22, 23] methods. The solution approach applied by PEPS consists on multiplying a probability vector by a non-trivial structure (known as *Kronecker descriptor*) in a process called as *Vector-Descriptor Product* (VDP) [9].

PEPS is a software composed of three modules corresponding to each phase of the performance evaluation process: *description*, *compiling*, and *solution*. This approach of solving a SAN model by modules used by PEPS improves the maintenance and development of the software tool. However, it becomes more difficult the software usage by the modeler, since it is necessary a wide knowledge about the modeling and solving process of a SAN model, even to obtain fast results for simple SAN models.

PEPS provides either a full probability vector (*i.e.*, a vector that contains the probabilities of all states of the model, including the *unreachable* states) or a list of evaluated reward functions defined by the modeler. These functions are called *integration functions* and they are expressed as arithmetic functions over the global states and their corresponding probabilities. For the SAN model presented in Figure 1, the integration function IF_H, for example, sum all probabilities of the system modeled by the SAN model of being in state H (*i.e.*, the sum of the probabilities of being in states HX, HY, and HZ). This integration function is expressed by:

$$IF_H = (state\ A1 == H)$$

Even though PEPS takes advantage of a compact description of the model using a tensor format (*i.e.*, a Kronecker descriptor), the solution of some SAN models can be prohibitive in terms of memory consumption. For example, the solution of a SAN model using PEPS needs to compute a probability vector for all *global states*, *i.e.*, the size of the vector is bounded by the *product state space* of the model. This is a strong limitation imposed by PEPS in terms of memory usage, specially for models where RSS $\ll$ PSS.

3. SAN LITE-SOLVER

Beyond using the compact format of a SAN model to solve it by the VDP process [9], it is also possible to compute all transitions of the model and represent them in a single matrix (*i.e.*, a transition matrix) that corresponding to the underlying Markov chain. However, depending of the nature of the model's events, the generation of this transition matrix can be prohibitive in terms of memory consumption and/or CPU time.

SAN LITE-SOLVER is a free software tool coded in C++ and compiled using g++ version 4.2.1 (GCC – The GNU Compiler Collection) with optimization options (-O3) and dynamic linkage. SAN LITE-SOLVER computes the steady-state probability vector of a SAN model, using a *Multi-valued Desicion Diagram* (MDD) [5] to compute and store the model's reachable state space (RSS) and it also represents the underlying Markov chain (MC), *i.e.*, the transition matrix of the model, by a sparse matrix in a Harwell-Boeing format (HBF) [1].

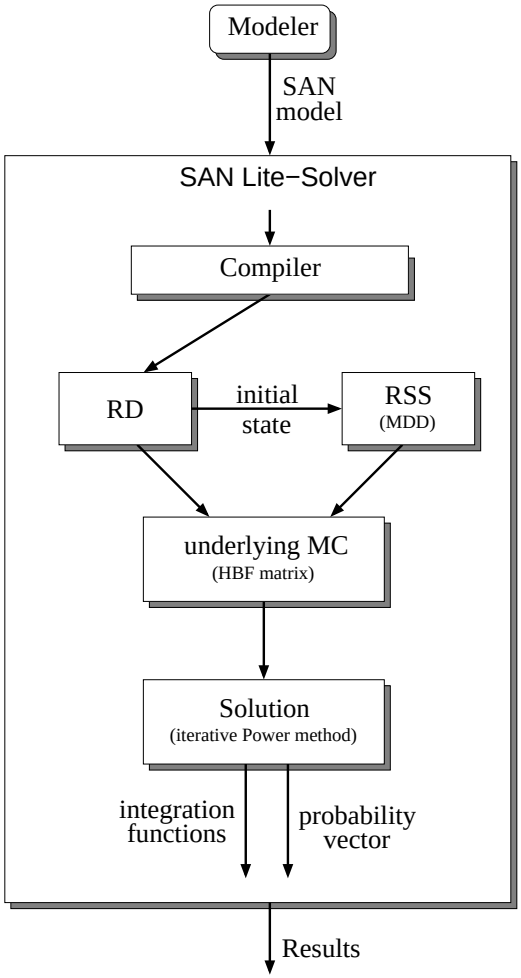


Figure 2. Overall solution scheme.

SAN LITE-SOLVER applies an efficient algorithm to generate the model's RSS [6, 10] and it also uses a sophisticated approach based on the usage of the model's RSS to compute the underlying MC. In fact, the idea of this tool is to facilitate the usage by the modeler for obtaining as fast as possible the results of a SAN model. Comparing to the current approach (*i.e.*, the PEPS software tool), SAN LITE-SOLVER spends more memory to store the underlying MC (since it is represented by a sparse matrix instead of a Kronecker descriptor), however it allows to employ a *lite* approach to numerically solve a SAN model, *i.e.*, the numerical solution is computed by an iterative process known as *Vector-Matrix Product* (VMP).

In Figure 2, it is shown an overall solution scheme for solving SAN models by SAN LITE-SOLVER. This SAN model solution process can be expressed by the following steps: (i) compile the SAN model and represent all transitions between states by a Kronecker-based encoding scheme, called *Reachability Descriptor* (RD); (ii) given an initial state and the computed RD of the model, generate and store the model's RSS using a MDD structure; (iii) compute the underlying MC of the model by the usage of RD and MDD, storing in a HBF matrix; and finally (iv) from this HBF matrix, apply the iterative Power method to obtain the solution of the SAN model, resulting the probability vector of all *reachable* states and the evaluated integration functions.

3.1. Tool usage

SAN LITE-SOLVER is a very easy and simple software tool for solving SAN models. In fact, SAN LITE-SOLVER is a command-line software tool that receives a SAN model described in a text file and shows the result of the integration functions defined in the model. SAN LITE-SOLVER is used as simple as:

```
san-lite-solver <SAN model> [ options ]
```

Moreover, SAN LITE-SOLVER provides a list of options that can help on the analysis and study of the modeled system. In Table 1, it is presented all available options for using SAN LITE-SOLVER. Some examples of usage for these options can be found in the software tool webpage at *http://www.inf.pucrs.br/afonso.sales/san-lite-solver*.

Note that options "-hbf" and "-sim" can provide a file that contains the underlying MC of the model, which can be used by other software tools in order to apply different solution methods (*e.g.*, direct methods, such as *LU factorization*; or even other iterative methods, such as *GMRES* [24]). Another possible application to those options is to provide a huge (and sparse) transition matrix that can be used by simulation methods in order to find an approximated solution of the model.

Table 1. Options for using SAN LITE-SOLVER.

Option	Description
-res	Saves the result of all integration functions into a file "*.res*".
-int	Computes the result of all integration functions using the probability vector from a file "*.vct*" (it also needs the file "*.mdd*").
-tra	Computes the number of transitions in the underlying MC and shows an estimated memory consumption to store it into a HBF matrix.
-loa	Loads the probability vector from a file "*.vct*".
-vct	Saves the probability vector into a file "*.vct*".
-ite	Maximum number of iterations to be performed (default value: 1,000,000).
-tol	Tolerance value for convergence verification (default value: 1e-10).
-mdd	Saves a standard MDD structure that represents the model's RSS into a file "*.mdd*".
-stt	Saves all reachable states into a file "*.stt*" (using the state's indices).
-lab	Saves all reachable states into a file "*.stt*" (using the state's labels).
-ddd	Saves a standard MDD structure that represents the model's RSS into a file "*.dot*".
-dmc	Saves the underlying MC of the model into a file "*.dot*".
-hbf	Saves the underlying MC of the model into a file "*.hbf*" (*column-oriented* type).
-sim	Saves the underlying MC of the model into a file "*.hbf*" (*row-oriented* type).

4. TOOL PERFORMANCE ANALYSIS

In this section, the performance of both tools (SAN LITE-SOLVER and PEPS) are compared in terms of memory consumption to store all necessary structures and CPU time to solve SAN models. The experiments were performed on a machine with Intel Core i7 quad-core, running at 2.2 GHz with 4 GB of main memory. In order to illustrate this performance analysis, some modeling examples[4] with different characteristics are chosen, such as:

- **Wireless *ad hoc* Networks** (WN) [14] – a model that represents a chain of N mobile nodes in a wireless network running over the IEEE 802.11 standard for *ad hoc* networks;
- **Kanban system** (KS) [6] – a model of manufacturing system with four similar stations, where "kanbans" (*tags*) control the flow of pieces. Each station has four queues, where N is the number of kanbans available in each station;
- **Resource Sharing** (RS) [6] – the classical example of resource sharing with different network combinations of P processes that share R resources;
- **Dining Philosophers** (DP) [6] – a model for the classical problem of N silent philosophers sitting around a circular table where they alternately think and eat;
- **First Available Server** (FAS) [25] – a model used to analyze the server availability considering N servers. In this example, packages arrive at a server switch block, depart through the first output port (or server) that is not busy, as long as at least one server is not blocked.

Table 2 shows the performance results of both software tools (SAN LITE-SOLVER and PEPS) in terms of memory consumption. In this table, columns *PSS* and *RSS* indicate, respectively, the product and reachable state spaces of the models. Regarding the SAN LITE-SOLVER memory costs, columns *RD*, *MDD*, and *HBF* correspond to the costs, respectively, to store the reachability descriptor, the MDD structure that contains the model's RSS, and the sparse matrix that corresponds to the underlying Markov chain. And regarding the PEPS memory costs, column *Descriptor* indicates the total memory used to store the descriptor and its auxiliary variables.

Regarding the memory consumption presented in Table 2 for models WN and KS, it is easy to notice that the SAN LITE-SOLVER memory usage for storing *RD*, *MDD*, and *HBF* is considerable lower than the memory used by PEPS to store the descriptor and its auxiliary variables. And specially for model KS (N=4), where the model's RSS $\ll$ PSS, the SAN LITE-SOLVER memory consumption remains in a low-level, while it becomes almost prohibitive for PEPS in an ordinary machine. It is important to remember that the models used by PEPS are bounded by the product state space. On the other hand, for the remaining models, SAN LITE-SOLVER have been used more memory to store all necessary structures, but even for those large models (*i.e.*, models with million states) it is still possible to store the sparse matrix (*i.e.*, HBF matrix) in the ordinary computer's main memory.

Table 3 shows the performance results in terms of CPU time to generate all necessary structures, as well as the CPU

[4] A brief explanation of each model is here presented. Please refer to the literature in order to obtain more details about those SAN models.

Table 2. Performance analysis in terms of memory consumption.

			SAN LITE-SOLVER				PEPS
Model	PSS	RSS	RD	MDD	HBF	Total	Descriptor
WN (N=16)	19,131,876	1,766	0.02 MB	0.03 MB	0.12 MB	**0.17 MB**	4.58 MB
WN (N=18)	172,186,884	4,622	0.03 MB	0.03 MB	0.33 MB	**0.39 MB**	41.07 MB
WN (N=20)	1,549,681,956	12,102	0.03 MB	0.03 MB	0.91 MB	**0.97 MB**	369.50 MB
KS (N=2)	531,441	4,600	0.01 MB	0.03 MB	0.53 MB	**0.57 MB**	4.23 MB
KS (N=3)	16,777,216	58,400	0.01 MB	0.11 MB	8.15 MB	**8.27 MB**	132.05 MB
KS (N=4)	244,140,625	454,475	0.01 MB	0.35 MB	71.13 MB	**71.49 MB**	1.88 GB
RS (P=14, R=11)	196,608	16,278	0.03 MB	0.55 MB	3.83 MB	4.41 MB	**0.07 MB**
RS (P=17, R=7)	1,048,576	41,226	0.04 MB	0.65 MB	8.67 MB	9.36 MB	**0.27 MB**
RS (P=20, R=5)	6,291,456	21,700	0.06 MB	0.69 MB	3.57 MB	4.32 MB	**1.53 MB**
DP (N=15)	14,384,907	470,832	0.03 MB	0.02 MB	79.70 MB	79.75 MB	**3.44 MB**
DP (N=16)	43,046,721	1,136,689	0.04 MB	0.02 MB	203.62 MB	203.68 MB	**10.28 MB**
DP (N=17)	129,140,163	2,744,210	0.04 MB	0.03 MB	518.64 MB	518.71 MB	**30.81 MB**
FAS (N=20)	1,048,576	1,048,576	0.03 MB	0.01 MB	200.00 MB	200.04 MB	**0.27 MB**
FAS (N=21)	2,097,152	2,097,152	0.03 MB	0.01 MB	416.00 MB	416.04 MB	**0.52 MB**
FAS (N=22)	4,194,304	4,194,304	0.04 MB	0.01 MB	864.00 MB	864.05 MB	**1.02 MB**

time to solve models using both software tools (SAN LITE-SOLVER and PEPS). In this table, columns *RD+MDD*, *HBF*, and *Solution* related to SAN LITE-SOLVER indicate, respectively, the execution times to generate the reachability descriptor and the MDD structure, the transition matrix stored in a HBF matrix, and the time spent in the iterative Power method (with a 1e-10 precision tolerance) for solving SAN models. In regard to the PEPS performance, column *Solution* also shows the CPU time spent to solve a SAN model by PEPS using the iterative Power method with the same precision tolerance.

Remark that the results presented in Table 3 show that SAN LITE-SOLVER can be orders of magnitude faster than PEPS, even computing all reachable states and generating the underlying Markov chain before solving a model. And in some cases SAN LITE-SOLVER can solve models that are prohibitive in terms of memory consumption by PEPS, since it is needed to allocate a probability vector of size equals to the product state space of the model. This fact can be confirmed on model WN (N=18 and N=20) in Table 3, where the PSS is of hundreds of million states and the model's RSS is about few thousand states. The same phenomenon happens on models KS (N=4) and DP (N=17).

5. CONCLUSION

One of the advantages of using a structured modeling formalism is to use a high-level abstraction to describe a system. Stochastic Automata Networks (SAN) is a powerful structured formalism that allows to describe a whole system in a modular way (*i.e.*, the modeling of a system by "small" subsystems). However, in some cases, this modularity may lead to a large number of invalid combinations of the model, and hence making impossible its solution.

In this paper, it was presented a new software tool (called SAN LITE-SOLVER) that allows the solution of SAN models in a fast, easy and intuitive way, specially when it is compared to PEPS (the current software tool used to solve SAN models) [20].

PEPS is a still valuable software tool to solve large models where PSS and RSS are really close and really large (*i.e.*, models with tens of millions of states). But, for the most part of the models that use a dozen of automata for describing a system (*i.e.*, models limited to a few million states), SAN LITE-SOLVER provides an efficient way to solve SAN models when compared to PEPS. In addition, it is also possible using SAN LITE-SOLVER to export the underlying Markov chain of the structured model to a file in order to apply different solution methods, such as *direct* or *simulation* methods.

As future work it is natural to assume the addition to other iterative solution methods, such as *GMRES* [24], as well as the *uniformization* method for transient solutions. It is also feasible to propose a SAN LITE-SOLVER parallel version for current heterogenous architectures (*i.e.*, machines that provide a parallel behavior from CPU and GPU).

It is important to keep in mind that the main ideas presented in this paper responsible for the SAN LITE-SOLVER design are not restricted to the SAN formalism. An analogous software tool could be easily developed for other formalisms, such as *Performance Evaluation Process Algebra* (PEPA) [2] or STDEVS [8], using the same techniques applied to the SAN LITE-SOLVER software tool.

Readers interested in the usage of SAN LITE-SOLVER may freely download it accessing the software tool webpage at *http://www.inf.pucrs.br/afonso.sales/san-lite-solver*.

Table 3. Performance analysis in terms of CPU time.

			SAN LITE-SOLVER				PEPS
Model	PSS	RSS	RD+MDD	HBF	Solution	Total	Solution
WN (N=16)	19,131,876	1,766	≈0.00 s	0.03 s	1.65 s	**1.68 s**	1.76 day
WN (N=18)	172,186,884	4,622	≈0.00 s	0.08 s	6.54 s	**6.62 s**	– – –
WN (N=20)	1,549,681,956	12,102	≈0.00 s	0.24 s	19.85 s	**20.09 s**	– – –
KS (N=2)	531,441	4,600	≈0.00 s	0.06 s	0.09 s	**0.15 s**	298.89 s
KS (N=3)	16,777,216	58,400	≈0.00 s	0.88 s	2.12 s	**3.00 s**	16,201.98 s
KS (N=4)	244,140,625	454,475	0.01 s	7.52 s	24.72 s	**32.25 s**	– – –
RS (P=14, R=11)	196,608	16,278	0.01 s	0.86 s	0.09 s	**0.96 s**	8.06 s
RS (P=17, R=7)	1,048,576	41,226	0.01 s	2.93 s	0.15 s	**3.09 s**	55.65 s
RS (P=20, R=5)	6,291,456	21,700	0.01 s	1.84 s	0.03 s	**1.88 s**	223.06 s
DP (N=15)	14,384,907	470,832	≈0.00 s	11.63 s	20.03 s	**31.66 s**	5,026.45 s
DP (N=16)	43,046,721	1,136,689	≈0.00 s	30.47 s	55.54 s	**86.01 s**	16,860.77 s
DP (N=17)	129,140,163	2,744,210	≈0.00 s	79.47 s	150.74 s	**230.21 s**	– – –
FAS (N=20)	1,048,576	1,048,576	≈0.00 s	18.89 s	24.49 s	**43.38 s**	63.26 s
FAS (N=21)	2,097,152	2,097,152	≈0.00 s	40.45 s	55.31 s	**95.76 s**	138.44 s
FAS (N=22)	4,194,304	4,194,304	≈0.00 s	82.19 s	120.55 s	**202.74 s**	301.57 s

ACKNOWLEDGMENTS

The author receives grant from CAPES, Brazilian Ministry of Education (PNPD 02388/09-0). This work is also partially financed by CAPES (AEX 9225/11-0).

BIOGRAPHY

Afonso Sales is university lecturer in Computer Science at *Pontifical Catholic University of Rio Grande do Sul* (PUCRS) and *Federal University of Rio Grande do Sul* (UFRGS), Porto Alegre, Brazil. He is also currently a fellow researcher in *Performance Evaluation Group* (PEG) and *PaleoProspec* project at PUCRS and *Parallel and Distributed Processing Group* (GPPD) at UFRGS. Afonso got his Ph.D. (2009) in Computer Science degree from *Grenoble Institute of Technology* (Grenoble INP), France. He has wide knowledge about state space generation techniques using decision diagrams and numerical solution methods based on structured description of Markovian models. Afonso also spent three years as Software Engineer at Hewlett-Packard Brazil R&D team. His research interests include stochastic modeling and simulation, continuous and discrete time modeling, structured Markovian formalisms, such as Stochastic Automata Networks (SAN) and Stochastic Petri Nets (SPN), structured and Kronecker-based approaches for Markov analysis, model checking, as well as performance evaluation of systems applied to several domains, such as software engineering, performance testing, computer networks, and parallel and distributed computing.

REFERENCES

[1] W.J. Stewart. *Introduction to the numerical solution of Markov chains*. Princeton University Press, 1994.

[2] S. Gilmore, J. Hillston, L. Kloul, and M. Ribaudo. PEPA nets: a structured performance modelling formalism. *Performance Evaluation*, 54(2):79–104, 2003.

[3] M. Ajmone-Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. John Wiley & Sons, 1995.

[4] B. Plateau. On the stochastic structure of parallelism and synchronization models for distributed algorithms. In *Proc. of the 1985 ACM SIGMETRICS Conf. on Measurements and Modeling of Computer Systems*, pages 147–154, Austin, Texas, 1985. ACM Press.

[5] T. Kam, T. Villa, R.K. Bryaton, and A. Sangiovanni-Vincentelli. Multi-valued decision diagrams: theory and applications. *Multiplie-Valued Logic*, 4(1-2):9–62, 1998.

[6] A. Sales and B. Plateau. Reachable state space generation for structured models which use functional transitions. In *6th International Conference on the Quantitative Evaluation of Systems (QEST'09)*, pages 269–278, Budapest, Hungary, 2009. IEEE Computer Society.

[7] G. Ciardo, G. Lüttgen, and A.S. Miner. Exploiting interleaving semantics in symbolic state-space generation. *Formal Methods in System Design*, 31(1):63–100, 2007.

[8] R. Castro, E. Kofman, and G.A. Wainer. A Formal Framework for Stochastic Discrete Event System Specification Modeling and Simulation. *Simulation*, 86(10):587–611, 2010.

[9] P. Fernandes, B. Plateau, and W.J. Stewart. Efficient descriptor-vector multiplication in Stochastic Automata Networks. *Journal of the ACM*, 45(3):381–414, 1998.

[10] A. Sales. *Réseaux d'Automates Stochastiques: Génération de l'espace d'états atteignables et Multiplication vecteur-descripteur pour une sémantique en temps discret*. PhD thesis, Institut Polytechnique de Grenoble (Grenoble INP), Grenoble, France, September 2009. Brigitte Plateau (advisor).

[11] L. Brenner, P. Fernandes, and A. Sales. The Need for and the Advantages of Generalized Tensor Algebra for Structured Kronecker Representations. *International Journal of Simulation: Systems, Science & Technology (IJSIM)*, 6(3-4):52–60, 2005.

[12] L. Baldo, L. Brenner, L. G. Fernandes, P. Fernandes, and A. Sales. Performance Models for Master/Slave Parallel Programs. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 128(4):101–121, 2005.

[13] R.M. Czekster, P. Fernandes, A. Sales, T. Webber, and A.F. Zorzo. Stochastic model for QoS assessment in multi-tier web services. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 275:53–72, 2011.

[14] F.L. Dotti, P. Fernandes, A. Sales, and O.M. Santos. Modular Analytical Performance Models for Ad Hoc Wireless Networks. In *3rd International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt'05)*, pages 164–173, Trentino, Italy, 2005. IEEE Computer Society.

[15] R. Chanin, M. Corrêa, P. Fernandes, A. Sales, R. Scheer, and A.F. Zorzo. Analytical Modeling for Operating System Schedulers on NUMA Systems. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 151(3):131–149, 2006.

[16] C. Bertolini, L. Brenner, P. Fernandes, A. Sales, and A.F. Zorzo. Structured Stochastic Modeling of Fault-Tolerant Systems. In *12th IEEE/ACM International Symposium on Modelling, Analysis and Simulation on Computer and Telecommunication Systems (MASCOTS'04)*, pages 139–146, Volendam, The Netherlands, October 2004. IEEE Computer Society.

[17] P. Fernandes, A. Sales, A.R. Santos, and T. Webber. Performance Evaluation of Software Development Teams: a Practical Case Study. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 275:73–92, 2011.

[18] J.E. Hopcroft and J.D. Ullman. *Introduction to automata theory, languages and computation*. Addison-Welsey, 1979.

[19] B. Plateau and J.M. Fourneau. A methodology for solving Markov models of parallel systems. *Journal of Parallel and Distributed Computing*, 12:370–387, 1991.

[20] L. Brenner, P. Fernandes, B. Plateau, and I. Sbeity. PEPS2007 - Stochastic Automata Networks Software Tool. In *Int. Conf. on the Quantitative Evaluation of Systems (QEST'07)*, pages 163–164, Edinburgh, UK, 2007. IEEE Computer Society.

[21] R.M. Czekster, P. Fernandes, A. Sales, and T. Webber. Restructuring tensor products to enhance the numerical solution of structured Markov chains. In *6th International Workshop on the Numerical Solution of Markov Chains (NSMC)*, pages 36–39, Williamsburg, VA, USA, September 2010.

[22] R.M. Czekster, P. Fernandes, A. Sales, D. Taschetto, and T. Webber. Simulation of Markovian models using Bootstrap method. In *Summer Computer Simulation Conference (SCSC)*, pages 564–569, Ottawa, ON, Canada, July 2010. The Society for Modeling & Simulation International.

[23] P. Fernandes, J. M. Vincent, and T. Webber. Perfect Simulation of Stochastic Automata Networks. In *Int. Conf. on Analytical and Stochastic Modelling Techniques and Applications (ASMTA'08)*, volume 5055 of *LNCS*, pages 249–263, 2008.

[24] Y. Saad and M. H. Schultz. GMRES: A Generalized Minimal RESidual Algorithm for Solving Nonsymmetric Linear Systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3):856–869, 1986.

[25] A. Benoit, P. Fernandes, B. Plateau, and W. J. Stewart. On the benefits of using functional transitions and Kronecker algebra. *Performance Evaluation*, 58(4):367–390, 2004.