PEPS2015 - Stochastic Automata Networks Software Tool

Paulo Fernandes[1], Afonso Sales[1], and Luis Zani[1]

{paulo.fernandes, afonso.sales}@pucrs.br, luis.zani@acad.pucrs.br
Computer Science Department
PUCRS University - Porto Alegre - BRAZIL

Abstract. This paper presents the new version of PEPS software tool, designed for solving models expressed using Stochastic Automata Networks (SAN). The SAN formalism is historically the first Markovian structured formalism employing Tensor (Kronecker) Algebra. PEPS tool is a live project, therefore a short timeline of its previous versions, as well as the new features included in 2015 version, are presented, namely: compact and efficient reachable state space storage, efficient and powerful just-in-time function evaluation, optimized numerical solutions (indirect methods), and symbolic solution (direct solution). We also mention the tool capabilities to solve very large models.

1 Introduction

The Stochastic Automata Networks (SAN) formalism is a structured stochastic Markovian formalism proposed by Plateau [14]. It is defined as an analytical method of modeling systems and it can be applied in different areas e.g. software engineering, reliability, parallel programming, concurrent systems, and performance evaluation of diverse systems. It allows us to represent an entire system by splitting it into a set of subsystems, each with an independent behavior itself (*local events), and also eventual interdependencies among each other (*synchronizing events and functional rates). In that sense, the framework first proposed by Plateau establishes a structured and modular way to describe discrete and continuous-time Markovian models.

The PEPS project started in the late 80's and its goal was to develop a software package capable of computing numerical solutions for the SAN formalism. The first version of the tool was presented in [13] and featured a basic vector-matrix multiplication, where the matrix columns were generated one by one in each iteration. Only the tensor formula of the Kronecker descriptor was stored, and the full matrix was never generated. Later on, another three official versions of the software tool were published.

Among several areas to which PEPS tool may be applied, it is convenient to cite the areas of computing and communication performance modeling, parallel and distributed systems, and finite capacity queueing networks. PEPS differs from other tools supporting extended automata, such as UPPAAL [17] and

KRONOS [18], in regards to scope, as the mentioned tools are more focused on modeling, validation and verification of real-time systems[1].

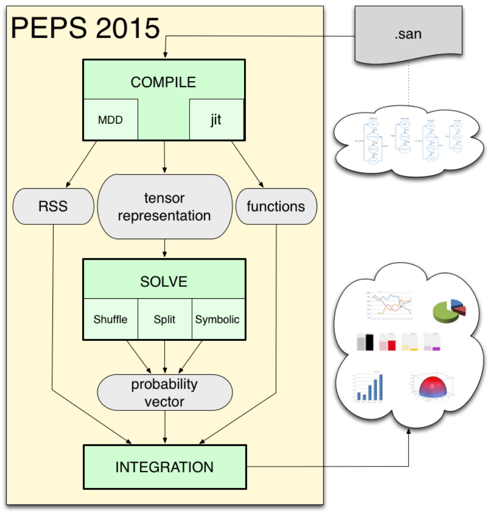


Fig. 1. Overview of PEPS 2015 modules and data

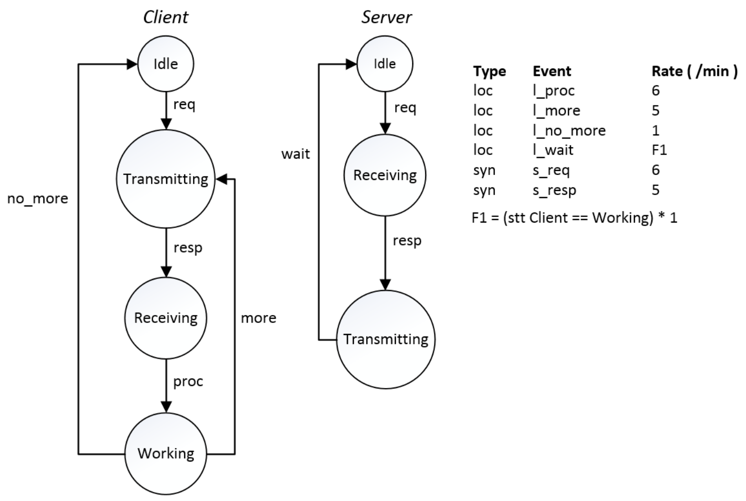
The core contribution of this paper, in regards to PEPS latest version, are essentially a compact and efficient reachable state space storage, efficient function evaluation, and optimized numerical solutions and symbolic solutions. These features, as well as the techniques and algorithms used in the tool, such as application of Multi-Valued Decision Diagrams (MDD), Tensor Algebra, Shuffle and Split algorithms, and symbolic solution, are shown in Fig. 1.

This paper is organized as follows: Section I introduces the idea proposed in this paper. Section II contextualizes the reader with a brief explanation of Stochastic Automata Networks. Section III presents pre-existing features that have been implemented in PEPS previous versions so far. Section IV describes the new features proposed in this paper for the latest version of the tool, which is under development in 2015. Section V presents a comparison of PEPS current version with the previous ones. Section VI shows the usage of the tool itself along with some examples. Finally, in Section VII, the conclusion summarizes this paper contribution and points out future works.

[1] Real-Time systems use networks of timed automata to model tasks that must be performed within strict time deadlines.

2 Stochastic Automata Networks

A SAN model can be seen as a collection of individual stochastic automata that operate almost independently of each other in order to represent different modules of a system. Each individual automaton consists of a number of states and rules that dictate the way it moves from one state to another. The state of the SAN is given by the current state of each of its component automata [14]. There can be any number of automata in a SAN model. In fact, if the model contains only one automaton, that would be merely a Markov Chain [9].



Type	Event	Rate (/min)
loc	l_proc	6
loc	l_more	5
loc	l_no_more	1
loc	l_wait	F1
syn	s_req	6
syn	s_resp	5

F1 = (stt Client == Working) * 1

Fig. 2. Example of a SAN model

For example, the SAN model in Fig. 2 describes a Client-Server system as a composition of two subsystems. The first one, named *Client*, possesses 4 local states: *Idle*, *Transmitting*, *Receiving*, and *Working*; and the second one, called *Server*, has three local states: *Idle*, *Transmitting*, and *Receiving*, in order to represent the basic functioning of the respective components. There are transitions between states called *local events*, for they only change the state of one automaton at a time, without affecting any other. In the described model, the local events are *proc*, for "process", *more*, for "more data to be transmitted", *no_more*, for "nothing else to be transmitted", and *wait*, for "wait for requests". There are also events which are called *synchronizing events*, which occur simultaneously in more than one automaton, making all of the affected automata change their states at the same time. In this case, the synchronizing events are *req*, for "request", and *resp*, for response. Note that they change states from "Transmitting" to "Receiving" on the client side, and vice-versa on the server side with only one occurrence.

Additionally, the *functional transition rates* determine that the average firing rate of the events is not a constant value [2]. In fact, there is a function (in this case, *F1*) that determines the possible values for this rate, depending on the state of other automata. in the example depicted in Fig. 2, the functional transition rate *F1* states that and the Server only changes states from “Transmitting” to “Idle” when the Client is at the state “Working”, meaning that while there is communication between the components, the server cannot go idle. The reader interested in further information about this subject may find more detailed material in [11].

3 Pre-existent Features

The previous version of PEPS software tool, PEPS2007 [2], contains features that have been developed and improved along the years, since the beginning of the project. This section presents each of these features with more details.

3.1 Textual Input (.san)

PEPS software tool takes files with the extension *.san* as an input. These files contain the detailed description of every component of a SAN model that is to be numerically resolved by the tool. The specification of a SAN is basically composed of five sections: identifiers, events, partial reachability, network, and results. An example of a *.san* file that represents the model depicted in Fig. 2 is presented on Section 5.3 of this paper.

Identifiers: The first section of a *.san* file is where the average firing rates for all the events in the model are defined. However, each rate is required to have a unique name, *i.e.*, an identifier.

Events: In this section, all events in the model have to be described. Hence, for each event, its type (local or synchronizing) and name are specified, as well as which identifier its rate corresponds to. Each event rate is associated with the identifier for that specific rate or to a function that represents a functional transition rate.

Partial Reachability: This is the section where the starting state of each automaton is defined. Since not all global states are reachable, the combination of the starting states is specified to be surely reachable. In other words, it is guaranteed that at least this global state is known to be a reachable state.

[2] Since it is a Markovian Formalism, SAN assumes that all rates represent the average of an exponential distribution. Hence, a constant rate stands for the average frequency of an exponentially distributed phenomenon [9].

Network: After defining the partial reachability of the model, all of the automata must be described. In this section of the file, the names, states, and the transitions along with its corresponding firing rates are defined for each automaton.

Results: The software tool resolves the SAN model given as input by means of numerical solution algorithms described in [5, 7, 8]. The output of the system is the probability of the states configuration to happen throughout the time. Hence, this section is where the states (either local or global) that one wants to know the probability of the model to be at are specified.

3.2 Automata Aggregation

Automata aggregation in SAN is meant to reduce the number of automata in the model, bringing some numerical benefits to the solution. Among these benefits, it is important to point out the theoretical and practical advantage, which is the decrease of total state space and reduced memory usage, respectively.

Alongside with the reduction of the total state space, additional benefits consequently come up, such as the elimination of some functional rates and unreachable states. It occurs because of the nature of the aggregation methods, which basically groups automata that are connected to the same state(s) into only one automaton.

There are essentially two aggregation methods for SAN formalism. The algebraic aggregation method uses properties of the generalized tensor algebra to reduce the number of automata. The semantic aggregation method is based on the relationship among replicated automata [1].

3.3 Just-in-Time Functions

PEPS also implements a *just-in-time* function evaluation, which basically creates individual files containing C++ coded functions for each functional transition rate defined in the SAN model. Then, the C++ function codes are compiled and linked with PEPS in compile time, in such a way that they can be called whenever they are needed, *i.e.*, when a function is evaluated in runtime.

3.4 Stationary and Transient Numerical Solutions (Shuffle Algorithm)

The Vector-Descriptor product is one of the most important operations to achieve both stationary and transient solutions for models described by Kronecker structured formalisms using iterative methods. The algorithm that is usually used to perform this operation is called the *Shuffle* algorithm [8].

A thorough study about matrices reordering and generalized tensor algebra properties aiming to optimize the evaluation of functional rates is given in the literature [8].

The Shuffle algorithm basically consists in exploiting the property of decomposing a tensor product into the ordinary product of tensor products in a normal form.

In other words, the Shuffle algorithm consists in the successive multiplication of a vector by each normal factor. In fact, vector v is multiplied by the normal factor, then the resulting vector is multiplied by the normal factor again, and so on until the last factor. The multiplication of a vector v by the i^{th} normal factor is equivalent to *shuffle* the elements of v in order to assemble $nleft_i \times nright_i$ vectors of size n_i and multiply them by matrix $Q^{(i)}$. So, assuming the matrix $Q^{(i)}$ is stored as a sparse matrix, the number of multiplications necessary to multiply a vector by the i^{th} normal factor is:

$$nleft_i \times nright_i \times nz_i \tag{1}$$

where nz_i is the number of nonzero elements of the i^{th} matrix of the tensor product term $Q^{(i)}$. Considering the number of multiplications to all normal factors of a tensor product term, the result is [5, 8]:

$$\prod_{i=1}^{N} n_i \times \sum_{i=1}^{N} \frac{nz_i}{n_I} \tag{2}$$

The reader may find extensive material in the literature [5] in regards to memory and CPU efficiency of the Shuffle algorithm.

3.5 Integration Functions

The final steps of the Shuffle algorithm calculate the probability of each global state to happen, as mentioned previously. In that sense, one of PEPS implemented features is the integration functions. The main idea of these functions is to sum the product of each row of the probability vector containing all the global states. Since not all global states will necessarily be reachable, some of these probabilities will be zero.

4 PEPS 2015 - New Features

The latest version of the software tool, PEPS2015, contains some additional features, aiming to achieve a better performance and memory-space usage. A more detailed explanation of each of these features can be found hereafter.

4.1 C-like Functions

One of PEPS pre-existent features is the already mentioned *just-in-time* function evaluation, that creates separate files containing C++ code for the functions defined in the SAN file. With the goal of increasing performance of the application, this created C++ code is meant to be replaced by a new way of interpreting the functions described in the model.

The intention here is to change the way the functions are defined in the SAN file. Given that currently the compiler creates individual files with C++ coded functions to be linked with PEPS, it seems more reasonable to let the SAN specification be more of a C-like function. This way, instead of creating separate files with C++ code in compile time and then calling the created function codes whenever it is necessary in runtime, the compiler will be able to interpret proper C++ code that was specified directly in the SAN model, all in compile time. Thus, performance improvements are expected since the application will be able to run function evaluation without the need of accessing any side resource because the entire process is done beforehand.

4.2 Reachable State Space Efficient Generation

A Multi-valued Decision Diagram (MDD) [10] is a compact structure that allows us to store and to manipulate large sets of structured information in a compact format. The information in a model can be structured by N components (or subsystems, *i.e.*, in our case, *automata*) where these components have some independent behavior and occasional interdependency.

Basically a MDD is a directed acyclic edge-labeled multi-graph, where there are some specific properties, such as: (i) nodes are organized into $N+1$ *levels*, where N represents the number of subsystems; (ii) level N has only one single *non-terminal* node (known as the *root*), whereas levels $N-1$ through 1 have one or more non-terminal nodes; (iii) level 0 has two *terminal* nodes: 0 and 1; (iv) a non-terminal node p at level l contains n_l arcs pointing to nodes at level $l-1$, where n_l indicates the number of local states of l-th subsystem; (v) there are not *duplicate* nodes, *i.e.*, nodes at a same level are unique.

Fig. 3 illustrates a MDD which represents the state space $\mathcal{S}$, subset of a cross-product of a system splitted in four subsystems (*i.e.*, $N=4$), where $\mathcal{S}^{(i)}$ represents the local state space of the *i-th* subsystem.

In Fig. 3, *non-terminal* nodes are depicted by *circles* and *terminal* nodes are depicted by *squares*. A given state x of the system is composed by the combination of the local states of each subsystem, *i.e.*, $x = x^{(N)} \ldots x^{(1)}$ of $\mathcal{S}$ is element of a subset represented by a N-level MDD if and only if the path through the MDD, starting at the level-N node, following downward pointer $x^{(l)}$ at level l, reaches terminal node 1. Dashed arcs and nodes are paths which lead only to terminal node 0 and in the rest of the paper, for reasons of clarity, will be omitted.

Also, MDD can be used to represent a set S of integer tuples by storing the characteristic function f_S of the set. Therefore sets can be manipulated using operations over MDDs on their characteristic functions (*e.g.*, the operation union on sets corresponds to disjunction on MDDs).

Saturation-based state-space generation is a successful symbolic method based on MDDs applied to generating state spaces of structured models, *e.g.*, Stochastic Petri Nets (SPN) models [3, 4, 12].

Sales and Plateau [15] presented an extension of the saturation method described in [4], which allows the use of *functional transitions*, *i.e.*, the interaction

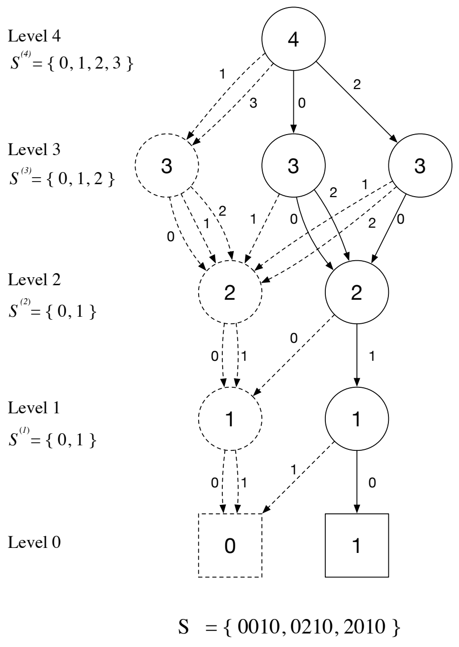


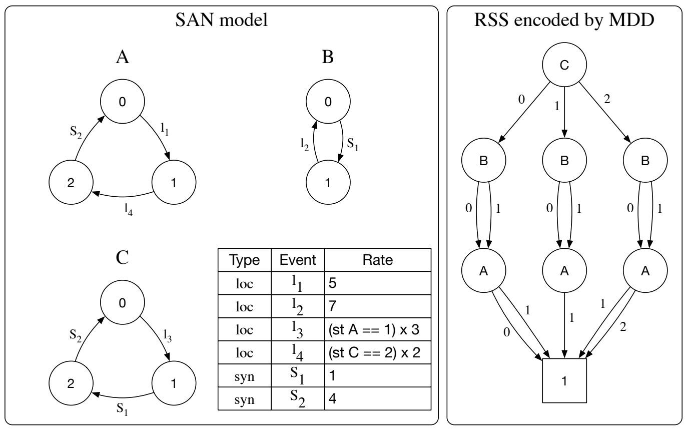
Fig. 3. A state space $\mathcal{S}$ represented by a MDD

between automata can be represented by a function. Functions can be associated to rates or routing probabilities on the events. In this case, the rate or the routing probabilities of an event can have different values in function to the state of other automata.

The main idea of the generation method is to compute the reachable space state (RSS) of the model by the successive firing of events from an initial state while the RSS is stored in a MDD. The method exploits the possibility of firing any event that affects a given MDD node and its descendant nodes, thus bringing the node to its *saturated* format. Besides, nodes are considered in a *bottom-up fashion* (*i.e.*, when a node is computed, all its descendant nodes are already in the *saturated* format). A node is considered as *saturated* if it encodes a set of states which are a fixed point in regards to the firing of any event at its level or at a lower level.

Fig. 4 shows a small SAN model with three automata ($N = 3$) and the corresponding RSS of this model re presented by a MDD, generated from initial state 000.

Functions are a powerful feature in the SAN formalism, since it allows us to represent very complex behaviors of a system in a very compact format. Moreover, the usage of MDDs to generate the RSS of a SAN model, which uses functions (having a small domain size), allows us to achieve the generation of large reachable state spaces with a small computational time, while keeping a low memory consumption.



Type	Event	Rate
loc	l_1	5
loc	l_2	7
loc	l_3	(st A == 1) x 3
loc	l_4	(st C == 2) x 2
syn	s_1	1
syn	s_2	4

Fig. 4. RSS of the SAN model represented by a MDD

4.3 Stationary and Transient Numerical Solutions (Split Algorithm)

SAN schemes that model real scenarios are naturally sparse: That's because the tensor sum structures generally make the local portion of the descriptor very sparse. Also, the synchronizing events make the descriptor quite sparse, since they are mostly used to describe exceptional behaviors.

The *Split* algorithm [5] allows each tensor product of matrices to be partitioned into two different groups: one with more sparse matrices and the other with more nonzero elements. This way, the Sparse algorithm [7] could be applied to the first group generating *Additive Unitary normal Factor* (AUNF). An AUNF is composed of three elements: a scalar value obtained by multiplying one nonzero element of each matrix in the Sparse-like part by each other; an input slice, which is a part of the vector *v* identified by the line row coordinates *i* of the nonzero elements multiplied; and an output slice, as a part of the vector *v* identified by the column coordinates *j* of the multiplied elements [5]. Each of the generated AUNF must then be multiplied (tensor product) by the second group of matrices using the Shuffle algorithm, as Slice handles only the first matrix in this case.

Considering this concept, the goal is to split the tensor terms into two sets of matrices and deal with them in different ways, almost individually. Therefore, the *Split* algorithm is considered a generalization for the encompassing all possibilities from a pure Sparse approach until the *Shuffle* algorithm, once it follows the idea of breaking the system into distinct parts to be dealt with.

In regards of efficiency, a deeply detailed material may be found in the literature [5], explaining the memory and CPU efficiency of the Split algorithm, as well as Sparse and Slice algorithms. Additionally, some optimizations on the Split algorithm were proposed by Czekster, Fernandes, and Webber in [7], in case the reader is interested.

4.4 Symbolic Solution

The main idea of the proposed symbolic solution is to deal with the tensor representation of the infinitesimal generator by adding terms to perform Gauss-Jordan Elimination steps [9]. The concept of functional elements is highly important for out method to deal with particular cases of the tensor structure and keep the symbolic operations simple enough to be handled.

The first step for the proposed symbolic solution is to prepare the matrix by performing the Gauss-Jordan method to obtain the infinitesimal generator. Once this process is done, the tensor representation of the starting of the Gauss-Jordan method will be:

$$Q^{(0)} = \bigoplus_{i=0}^{N-1} Q_l^{(i)} + \sum_{s \in S} \bigotimes_{i=0}^{N-1} Q_{s^+}^{(i)} + \sum_{s \in S} \bigotimes_{i=0}^{N-1} Q_{s^-}^{(i)} + \bigotimes_{i=0}^{N-1} Q_{lc}^{(i)} \tag{3}$$

The second step is to perform the matrix triangularization, in which the elements below the diagonal are eliminated through the application of the Gauss-Jordan to each row. The row scaling operations correspond to include functional elements to perform the necessary scale operation to each product of the descriptor. Once the matrix triangularization is complete, it is computationally simpler to store the tensor structure and perform the final step: a backward substitution procedure to resolve the system of equations. More detailed information about the method itself is presented by Fernandes, Lopes and Yeralan in [9].

5 Practical Capabilities

Table 1 shows some numerical results considering Product Space State (PSS) and type of model that the software tool can handle within reasonable time and space limits for PEPS current and previous versions.

Table 1. PEPS numerical results

PEPS version	**Model**	**PSS**	**Memory Used**	**Time to Solve**
1998	General Resource Sharing [19]	1,084,576	590 Bytes	33 sec.
2003	FAS - First Server Available [20]	33,554,432	11.2 KB	5.47 min.
2007	Master-Slave Parallel Program [5, 21]	65,367,200	38.54 KB	8.92 min.
2011	Globally Distributed Software Development Teams [22]	19,131,876	10.07 KB	6.19 min.
2015	Unreliable Production Lines [23, 24]	725,594,112	918.18 MB	133 hours

Based on the presented data, it is interesting to notice that the number of states in the network seems to be the driving factor for the final solution time. For the latest version of the tool, the execution time is fairly reasonable, for it resolves a model with a significantly larger number of states (725 million), using an acceptable amount of memory (less than 1 GB).

6 Software Tool

In order to make the way this tool works more explicit to the reader, this section presents an example of a *.san* file, as well as its compilation and execution processes. Regarding compatibility, the platforms that the software tool runs on are pointed out.

6.1 Environment

PEPS package has been tested and proven to be currently compatible with the following operating systems:

1. Linux - Ubuntu 14.04 64 bit
2. Mac OS X 10.8.2 - Mountain Lion
3. Mac OS X 10.9.2 - Mavericks
4. Mac OS X 10.10.2 - Yosemite

However, since the source code of PEPS is available upon request, the compilation and availability for other platforms is likely to be an easy task.

6.2 Example

The code shown in Code 1.1 is an example of a *.san* file, describing the SAN model depicted in Fig. 2, containing all components presented on Section 3.1 of this paper.

For this example, it was specified in the results section that the user expects to know the probability of the Client to be requesting, receiving, and processing. Also, the percentage of the time that the Client is transmitting data and the Server is receiving data and vice-versa. At last, the percentage of time that both Client and Server are idle.

6.3 Usage

An example of usage of the software tool is shown in this section. Both compilation and resolution of the *.san* file presented in the previous section can be seen in Fig. 5. For this demonstration, the compilation process does not make use of any aggregation method [1] and the resolution process uses the Power Method [11] to solve the model[3].

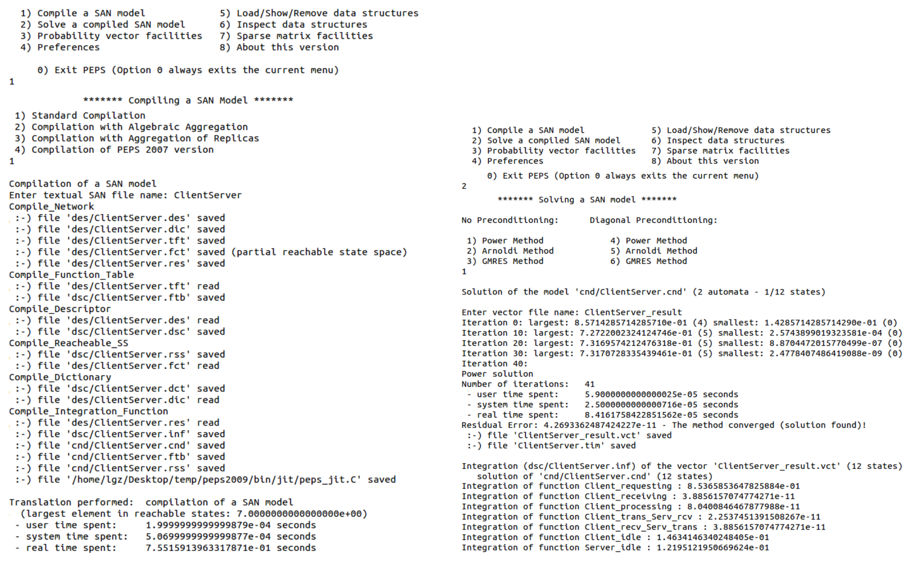
[3] PEPS versions of 2003, 2007, and obviously 2015 also offer the possibility to perform stationary solutions using Arnoldi and GMRES methods, and also transient solution using Uniformization method [16] is available.

```
identifiers

  //rates = per minute
  r_proc    = 6;
  r_more    = 5;
  r_no_more = 1;
  r_req     = 6;
  r_resp    = 5;

  F1 = (st Client == Working) * 1;

events

  loc l_proc    (r_proc);
  loc l_more    (r_more);
  loc l_no_more (r_no_more);
  loc l_wait    (F1);
  syn s_req     (r_req);
  syn s_resp    (r_resp);

// Both Client and Server start in idle.
partial reachability = ((st Client == Idle)
                     && (st Server == Idle));

network ClientServer (continuous)

  aut Client
    stt Idle         to (Transmitting)  s_req
    stt Transmitting to (Receiving)     s_resp
    stt Receiving    to (Working)       l_proc
    stt Working      to (Transmitting)  l_more
                     to (Idle)          l_no_more

  aut Server
    stt Idle         to (Receiving)    s_req
    stt Receiving    to (Transmitting) s_resp
    stt Transmitting to (Idle)         l_wait

results

  Client_requesting = (st Client == Transmitting);
  Client_receiving = (st Client == Receiving);
  Client_processing = (st Client == Working);
  Client_trans_Serv_rcv = ((st Client == Transmitting )
                        && (st Server == Receiving));
  Client_recv_Serv_trans = ((st Client == Receiving )
                         && (st Server == Transmitting));
  Client_idle = (st Client == Idle);
  Server_idle = (st Server == Idle);
```

Code 1.1. Example of a SAN file

The compilation generates a report of produced intermediate files, and the time took to compile. In this toy example, the model was compiled in 0.2 milliseconds of user time. Analogously, the solution presents a report of how many iterations were necessary until achieving convergence (41 for this example), the time spent (0.05 milliseconds for this example) and the computation of all integration functions.

According to the output of PEPS 2015, depicted in the right hand side of Fig. 5, the probability of both Client and Server automata to be idle is less than fifteen percent, which means that, for these parameters, the model presents satisfying results in regards to occupancy, for the Server is far from overloaded.



```
  1) Compile a SAN model            5) Load/Show/Remove data structures
  2) Solve a compiled SAN model     6) Inspect data structures
  3) Probability vector facilities  7) Sparse matrix facilities
  4) Preferences                    8) About this version

     0) Exit PEPS (Option 0 always exits the current menu)
1

             ******* Compiling a SAN Model *******
 1) Standard Compilation
 2) Compilation with Algebraic Aggregation
 3) Compilation with Aggregation of Replicas
 4) Compilation of PEPS 2007 version
1

Compilation of a SAN model
Enter textual SAN file name: ClientServer
Compile_Network
 :-) file 'des/ClientServer.des' saved
 :-) file 'des/ClientServer.dic' saved
 :-) file 'des/ClientServer.tft' saved
 :-) file 'des/ClientServer.fct' saved (partial reachable state space)
 :-) file 'des/ClientServer.res' saved
Compile_Function_Table
 :-) file 'des/ClientServer.tft' read
 :-) file 'dsc/ClientServer.ftb' saved
Compile_Descriptor
 :-) file 'des/ClientServer.des' read
 :-) file 'dsc/ClientServer.dsc' saved
Compile_Reacheable_SS
 :-) file 'dsc/ClientServer.rss' saved
 :-) file 'des/ClientServer.fct' read
Compile_Dictionary
 :-) file 'dsc/ClientServer.dct' saved
 :-) file 'des/ClientServer.dic' read
Compile_Integration_Function
 :-) file 'des/ClientServer.res' read
 :-) file 'dsc/ClientServer.inf' saved
 :-) file 'cnd/ClientServer.cnd' saved
 :-) file 'cnd/ClientServer.ftb' saved
 :-) file 'cnd/ClientServer.rss' saved
 :-) file '/home/lgz/Desktop/temp/peps2009/bin/jit/peps_jit.C' saved

Translation performed:  compilation of a SAN model
  (largest element in reachable states: 7.0000000000000000e+00)
 - user time spent:     1.9999999999999879e-04 seconds
 - system time spent:   5.0699999999999877e-04 seconds
 - real time spent:     7.5515913963317871e-01 seconds
```

```
  1) Compile a SAN model            5) Load/Show/Remove data structures
  2) Solve a compiled SAN model     6) Inspect data structures
  3) Probability vector facilities  7) Sparse matrix facilities
  4) Preferences                    8) About this version
     0) Exit PEPS (Option 0 always exits the current menu)
2
       ******* Solving a SAN model *******

No Preconditioning:      Diagonal Preconditioning:

 1) Power Method             4) Power Method
 2) Arnoldi Method           5) Arnoldi Method
 3) GMRES Method             6) GMRES Method
1

Solution of the model 'cnd/ClientServer.cnd' (2 automata - 1/12 states)

Enter vector file name: ClientServer_result
Iteration 0: largest: 8.5714285714285710e-01 (4) smallest: 1.4285714285714290e-01 (0)
Iteration 10: largest: 7.2722002324124746e-01 (5) smallest: 2.5743899019323581e-04 (0)
Iteration 20: largest: 7.3169574212476318e-01 (5) smallest: 8.8704472015770499e-07 (0)
Iteration 30: largest: 7.3170728335439461e-01 (5) smallest: 2.4778407486419088e-09 (0)
Iteration 40:
Power solution
Number of iterations:   41
 - user time spent:     5.9000000000000025e-05 seconds
 - system time spent:   2.5000000000000716e-05 seconds
 - real time spent:     8.4161758422851562e-05 seconds
Residual Error: 4.2693362487424227e-11 - The method converged (solution found)!
 :-) file 'ClientServer_result.vct' saved
 :-) file 'ClientServer.tim' saved

Integration (dsc/ClientServer.inf) of the vector 'ClientServer_result.vct' (12 states)
   solution of 'cnd/ClientServer.cnd' (12 states)
Integration of function Client_requesting : 8.5365853647825884e-01
Integration of function Client_receiving : 3.8856157074774271e-11
Integration of function Client_processing : 8.0400846467877988e-11
Integration of function Client_trans_Serv_rcv : 2.2537451391508267e-11
Integration of function Client_recv_Serv_trans : 3.8856157074774271e-11
Integration of function Client_idle : 1.4634146340248405e-01
Integration of function Server_idle : 1.2195121950669624e-01
```

Fig. 5. Compilation and resolution of the Client-Server SAN model

7 Conclusion

Fig. 1 summarizes the general vision of the PEPS 2015 software high level modules, as well as the data and information flow. In this figure is represented in the right hand side the input of PEPS 2015, a textual .san file describing an imagined SAN model, and the ultimate PEPS 2015 output the results computed by the application of integration functions over the computed probability vector, *i.e.*, the quantitative estimations concerning the modeled reality. The three main modules of PEPS 2015 are the Compile, Solve and Integrate modules.

The Compile performing translation of a SAN model into a MDD description of the reachable state space, a tensor representation of transition matrix (descriptor), and the integration functions responsible to deliver the quantitative results. It comprises two specialized submodules, one responsible to generate the MDD representation of the reachable state space, and the other responsible to generate the *just-in-time* (jit) generated functions for the tensor representation and the integration functions.

The Solve module implements all solution iterative methods available (Power, Arnoldi, GMRES and Uniformization) offering Shuffle and Split vector-descriptor multiplications, but also the symbolic solution through Gaus-Jordan Elimination. In such way, this module is the more sensitive module in terms of computational efficiency, both in terms of CPU and memory usage.

Finally, the Integrate module is the simplest and more straight forward module, since it basically visits the reachable state space computing the integration functions and weighting the results by the computed probabilities.

PEPS software tool is a live project, hence there is ongoing work to add new features and keep improving it by seeking for even better techniques for vector-descriptor multiplication to speed up both transient and stationary numerical solutions, as well as more sophisticated structures for reachable state space manipulation. The new features proposed here led to a new version of PEPS as the main contribution of this paper. As new techniques are proposed in the future, they may result in even newer and optimized versions of PEPS tool. The interested researcher and practitioner may find more thorough information about PEPS in the webpage `http://www.inf.pucrs.br/peg/peps/`.

References

[1] Benoit, A., Brenner, L., Fernandes, P., Plateau, B.: "Aggregation of stochastic automata networks with replicas. Linear Algebra and its Applications," 386, 111-136, 2004.

[2] Brenner, L., Fernandes, P., Plateau, B.: "PEPS2007 - Stochastic Automata Networks Software Tool" in Proceedings of the 4th International Conference on the Quantitative Evaluation of Systems (QEST 2007). Edinburgh, Scotland, p. 163-164. September, 2007.

[3] Ciardo G., Lüttgen G., Siminiceanu R.:"Saturation: An Efficient Iteration Strategy for Symbolic State-Space Generation," in Proceedings of the 7h Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'01). Lecture Notes in Computer Science. Springer-Verlag, 2001. Vol. 2031, p. 328-342.

[4] Ciardo G., Lüttgen G., Miner S.:"Exploiting interleaving semantics in symbolic state-space generation," Formal Methods in System Design. 2007. Vol. 31(1), p. 63-100.

[5] Czekster R. M., Fernandes P., Vincent J. M., Webber T.: "Split: A Flexible and Efficient Algorithm to Vector-descriptor Product" in International Workshop on Tools for solving Structured Markov Chains, ValueTools 2007. Vol. 321(83), p. 1-8, October, 2007, Nantes, France.

[6] Czekster, Ricardo M., Fernandes, P., Webber, T.: "GTAEXPRESS: a Software Package to Handle Kronecker Descriptors," in Proceedings of the 6th International Conference on the Quantitative Evaluation of Systems. QEST '09, p. 281-282. IEEE Computer Society, September, 2009.

[7] Czekster R. M., Fernandes P., Webber T.:"Efficient Vector-Descriptor Product Exploiting Time-Memory Trade-offs," ACM SIGMETRICS Performance Evaluation Review. New York, USA: ACM Press. 2011. p. 1-9.

[8] Fernandes, P., Plateau, B., Stewart, William. J.: "Efficient Descriptor-Vector Multiplication in Stochastic Automata Networks," Journal of the ACM (JACM), Vol. 45(3), p. 381-414, 1998.

[9] Fernandes, P., Lopes, L., Yeralan, S.: "Symbolic Solution of Kronecker-based Structured Markovian Models," in Proceedings of the IEEE 21st International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS). Los Alamitos, CA, USA: IEEE Press, 2013. p. 405-409.

[10] Kam T., Villa T., Bryaton, R. K., Sangiovanni-Vincentelli, A.:"Multi-valued decision diagrams: theory and applications," Multiple Valued Logic, Vol. 4(1-2), p. 9-62, 1998.

[11] Langville, Amy N., Stewart, William J.:"The Kronecker product and stochastic automata networks," Journal of Computational and Applied Mathematics, Vol. 167(2), p. 429-447, 2004.

[12] Miner A. S., Ciardo G.:"Efficient Reachability Set Generation and Storage Using Decision Diagrams," in Proceedings of the International Conference on Applications and Theory of Petri Nets (ICATPN'99). Lecture Notes in Computer Science. Williamsburg, VA, USA: Springer-Verlag Heidelberg, June 1999. Vol. 1639, p. 6-25.

[13] Plateau, B., Fourneau, J. M., Lee, K. H.: "PEPS: A Package for Solving Complex Markov Models of Parallel Systems" in Modeling Techniques and Tools for Computer Performance Evaluation, Ed. Puigjaner, R., Potier, D. Springer US, 1989. p. 291-305.

[14] Plateau B., Stewart W. J.:"Stochastic Automata Networks". Computational Probability. Kluwer Academic Press, 1997. p. 113-152.

[15] Sales A., Plateau B.:"Reachable state space generation for structured models which use functional transitions," in Proceedings of the 6th Int. Conf. on the Quantitative Evaluation of Systems (QEST'09). Budapest, Hungary: IEEE CS, 2009. p. 269-278.

[16] Uysal, E., Dayar, T.: "Iterative methods based on splittings for stochastic automata networks," European Journal of Operational Research, 110(1), 166-186. 1998.

[17] "UPPAAL", `http://www.uppaal.org/`

[18] "KRONOS", `http://www-verimag.imag.fr/DIST-TOOLS/TEMPO/kronos/`

[19] Fernandes, P.: "Méthodes numériques pour la solution de systèmes Markoviens à grand espace d'États," Institut National Polytechnique de Grenoble, France, 1998.

[20] Benoit A., Brenner L., Fernandes P., Plateau B.: "The PEPS software tool," In: 13th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation, 2003, Urbana. Lecture Notes in Computer Science. Vol. 2794, p. 98-115. Berlin: Springer-Verlag, 2003.

[21] Baldo, L., Brenner, L., Fernandes, L. G., Fernandes, P., Sales, A.: "Performance Models for Master/Slave Parallel Programs," Electronic Notes In Theoretical Computer Science, 128(4):101-121, April 2005.

[22] Fernandes, P., Sales, A., Santos, A. R., Webber, T.: "Performance evaluation of software development teams: a practical case study," Electronic Notes In Theoretical Computer Science, 275:73-92, September 2011.

[23] Fernandes, P.; OKelly E. J.; Papadopoulos, C. T.; Sales, A.: "Analysis of exponential reliable production lines using Kronecker descriptors," International Journal of Production Research, 51(4):4240-4257, February 2013.

[24] Fernandes, P.; OKelly E. J.; Papadopoulos, C. T.; Sales, A.: "Analysis of Exponential Unreliable Production Lines using Kronecker Descriptors," In: 9th Stochastic Models of Manufacturing and Service Operations (SMMSO 2013), Kloster Seeon, Germany, May 2013.