Symbolic Performance and Dependability Evaluation with the Tool CASPA

Matthias Kuntz[1], Markus Siegle[1], and Edith Werner[2]

[1] University of the Federal Armed Forces Munich, Department of Computer Science
{kuntz,siegle}@informatik.unibw-muenchen.de
[2] Georg-August-Universität Göttingen, Institut für Informatik
ewerner@informatik.uni-goettingen.de

Abstract. This paper describes the tool CASPA, a new performance evaluation tool which is based on a Markovian stochastic process algebra. CASPA uses multi-terminal binary decision diagrams (MTBDD) to represent the labelled continuous time Markov chain (CTMC) underlying a given process algebraic specification. All phases of modelling, from model construction to numerical analysis and measure computation, are based entirely on this symbolic data structure. We present several case studies which demonstrate the superiority of CASPA over sparse-matrix-based process algebra tools. Furthermore, CASPA is compared to other symbolic modelling tools.

1 Introduction

Symbolic data structures, such as binary decision diagrams (BDD) [3] and variants thereof, have proven to be suitable for the efficient generation and compact representation of very large state spaces and transition systems. In [13] it has been shown that in the context of compositional model specification formalisms such as process algebra, the size of the symbolic representation can be kept within linear bounds, even if the underlying state space grows exponentially. The key to such compact representation is the exploitation of the compositional structure of a given specification [14, 7, 24]. It is also known that in addition to functional analysis, performance analysis and the verification of performability properties can also be carried out on such symbolic representations [17, 24].

In this paper, we describe the new tool CASPA which offers a Markovian stochastic process algebra for model specification. CASPA generates a symbolic representation of the underlying labelled CTMC, which is based on multi-terminal BDDs (MTBDD), directly from the high-level model, without generating transition systems as an intermediate representation. In addition to specifying the model, the CASPA modelling language allows the user to specify different types of performance and dependability measures of interest. Numerical analysis and computation of measures are also carried out directly on the symbolic representation of the transition rate matrix of the underlying CTMC. To our knowledge, CASPA is the first stochastic process algebra tool whose implementation relies completely on symbolic data structures.

1.1 Related Work

Among other tools which are based on symbolic data structures are the model analyser SMART [4] and the probabilistic model checker PRISM [20, 21]. While SMART relies on multi-valued decision diagrams and matrix diagrams, PRISM – like CASPA – is based on multi-terminal binary decision diagrams. In Sec. 4, we compare CASPA to PRISM, mainly with respect to compactness of representation and effects of state space ordering. We also performed experiments with SMART, whose state space generation component seems to be even faster. However, we deliberately do not compare the numerical analysis component of SMART to that of CASPA, since this would basically boil down to a comparison of the algorithms of SMART and PRISM, which is not our focus here. In Sec. 4.1 we also compare CASPA to the work presented in [8], which is based on multi-valued decision diagrams and matrix diagrams. With respect to symbolic tools for stochastic process algebra, we also mention the work [11], where a PEPA specification (derived as an intermediate language from a UML specification) is used as input for PRISM.

1.2 Organisation of the Paper

The paper is organised as follows: In section 2 we introduce CASPA's specification language and give an overview of its architecture. Section 3 explains how the specification is translated to MTBDDs. In section 4 we demonstrate the usefulness of our approach by means of several case studies, which includes a comparison with other tools. Section 5 summarises our results and concludes with an outlook on future work.

2 Specification Language and Tool Architecture

In this section we briefly explain how a system and its measures of interest can be described in CASPA. We discuss an example specification and give some details on the tool's architecture and implementation.

2.1 System Specification

CASPA's specification language is derived from the stochastic process algebra TIPP [15, 12]. It provides operators for prefixing, choice, enabling, disabling, parallel composition and hiding. Infinite (i.e. cyclic) behaviour is specified by means of defining equations. All actions are associated with an exponential delay, which is specified by a rate parameter. The technique used for symbolic model representation (cf. Sec. 3) works only for finite state spaces. Therefore the grammar of the input language is such that recursion over static operators

```
(1)    /* Rate and constant definitions */
(2)    rate xi = 0.5;
(3)    rate gamma = 5;
(4)    rate mu = 0.3;
(5)    int max = 3;
(6)    /* System specification */
(7)    System       := (P(0) |[]| P(0)) |[b]| (hide a in Q(10))
(8)    Q(m [10])    := [m > 0] -> (a,xi); Q(m-1)
(9)                    [m = 0] -> (b,mu); Q(10)
(10)   P (n [max]) := [*] -> (b,gamma); P(n) + (c,gamma); stop
(11)                   [n > 0] -> (d,n*mu); P (n-1)
(12)                   [n < max] -> (a,0.3); P (n+1)
(13)   /* Measure specification */
(14)   statemeasure XXX (P{1}(n > 0) & !P{2}(n = max)) | Q(m = 4)
(15)   meanvalue YYY P{2}(n)
(16)   throughputmeasure ZZZ a
```

Fig. 1. Example CASPA specification

(i.e. parallel composition and hiding) is not allowed, which ensures that the underlying state space is finite.

The specification language allows the specification of parameterised processes, i.e. processes which carry one or more integer parameters. This feature is very useful for describing the behaviour of queueing, counting, or generally indexed processes. Within a parameterised process, the enabling of actions may be conditioned on the current value of the process parameters. In CASPA it is possible to define both rate and parameter constants. Parameters are always integer numbers, whereas rates are real numbers.

2.2 Example

We now discuss a small example (see Fig. 1). This specification has no special meaning, its only purpose is to introduce the language elements of CASPA. In lines (2) to (4) we find the definition of specific rate values, in line (5) a global parameter constant is defined. Lines (7) to (12) contain the system specification. Line (7) shows both possibilities to define parallel processes: The two `P(0)` processes are composed in parallel without interaction, i.e. all their actions are performed independently of each other. In contrast, `Q(10)` is composed in parallel with the two former processes in a synchronised way, i.e. action `b` must be performed by one of the `P`-processes and the `Q`-process at the same time. The synchronisation semantics is the same as for TIPP [15, 12]. In line (7) we also find the hiding operator: Action `a` in process `Q` is hidden from the environment and replaced by the special silent action `tau`. In line (8) we see an example of guarded choice: Action `a` can be performed if the value of parameter `m` is greater than zero. In line (10) we see a guarded choice whose test consists of $*$, which means the branch can be taken regardless of the actual parameter value. In lines (8) and (10) the maximum value of the respective parameters is given: For process `P` we chose a global constant `max`, for process `Q` the maximum value 10 is given explicitly. As for every process parameter such a maximum value has to be defined, the finiteness of the underlying state space is guaranteed. In

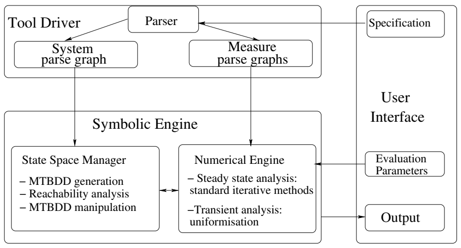

Fig. 2. Tool architecture

line (10) a choice between `(b,gamma); P(n)` and `(c,gamma); stop` is given. As all actions have exponential delay the choice of which action is actually taken corresponds to a race condition (as for stochastic Petri nets). In line (11) we see that rates can be arithmetic expressions, which makes it possible to define rates that are dependent on actual parameter values, similar to marking dependent rates in stochastic Petri nets. Finally, in lines (14) to (16) we find examples of measure specifications. We see that state measures can contain Boolean expressions with the usual connectives conjunction (&), disjunction (|) and negation (!). The clause `(P{1}(n > 0) & !P{2}(n = max))` characterises states in which parameter `n` of process `P{1}` is greater than zero, and parameter `n` of process `P{2}` is smaller than the maximum value, where `P{i}` expresses that we are interested in the `i`-th of the two `P` processes. A mean value will return the expected value of the specified process parameter (in line (15) it will be the mean value of parameter `n` of process `P{2}`), and a throughput measure will compute the throughput of the given action (in line (16) this is action `a`).

2.3 Tool Architecture

CASPA is written entirely in C. The lexical analyser was realised using the tool flex, the parser was written in bison. The symbolic engine was implemented using the BDD-library CUDD [25] and the hybrid numerical solution methods developed by Dave Parker [22] within the context of the tool PRISM. The tool architecture consists of three major parts [26], as shown in Fig. 2.

User Interface. Up to now, CASPA has only a textual user interface. A typical call of the tool consists of indicating the file name that contains the system and measure specification, the analysis method, parameters for the numerical analysis and information about the verbosity level. An example call looks as follows:

```
    caspa -v 1 -r -T -a TRANSIENT 100 ftcs.cas
```

The textual user interface is also used to present the results of the tool, i.e. number of states, information about the size of the symbolic representation, computation times, results of numerical analysis, etc. Additionally, CASPA can generate output that makes it possible to visualise the state space using the tool davinci [6], or for any graphical tool that can handle the .dot [1] format.

Tool Driver. From the command line the specification file is passed to the tool driver. It parses the system and the measure specification, translates them into their parse graphs and passes the results to the state space manager.

State Space Manager. The state space manager generates (from the parse graphs of the system and measure specification) the MTBDD representation of the labelled CTMC, resp. of the measures. It can perform reachability analysis, and manipulates the MTBDD data structure to allow for efficient numerical analysis (cf. section 3).

Numerical Engine. The numerical engine computes the vector of state probabilities. Several well-known numerical algorithms for both steady-state and transient analysis are implemented. The algorithms and their implementations are taken from PRISM, for their detailed description see [22]. The user can set the parameters of the algorithms, such as accuracy or maximum number of iterations.

3 Markov Chain Generation, Representation and Numerical Analysis

In this section, the approach of CASPA for directly mapping the process terms to MTBDDs is presented. Note that a CTMC is never constructed explicitly, only its symbolic encoding. A more detailed exposition of this translation can be found in [18]. We also briefly describe how the specified measures are related to the Markov chain representation and how the measures are computed.

3.1 Basis for Symbolic Representations

In this subsection we briefly introduce the basics of symbolic state space representation. An exhaustive account of this can be found in [24].

Multi-terminal Binary Decision Diagrams. MTBDDs [10] (also called algebraic decision diagrams (ADDs) [2]) are an extension of binary decision diagrams (BDDs) [3] for the canonical graph-based representation of functions of type $\mathbb{B}^n \mapsto \mathbb{R}$. We consider ordered reduced MTBDDs where on every path from the root to a terminal vertex the variable labelling of the non-terminal vertices obeys a fixed ordering.

Representation of CTMC. MTBDDs can be employed to compactly represent labelled CTMCs. Let $s \xrightarrow{a,\lambda} t$ be a transition of a labelled CTMC, where s is the source state, a is the action label, λ is the rate and t the target state of the transition, then this transition is encoded by a bit string, $a_1, ..., a_{n_L}, s_1, t_1, ...s_{n_S}, t_{n_S}$ where

- $a_1, ..., a_{n_L}$ encode the action label a
- $s_1, ..., s_{n_S}$ encode the source state s and
- $t_1, ..., t_{n_S}$ encode the target state t

In the MTBDD, there is a Boolean variable for each of these $n_L + 2 \cdot n_S$ bits, and the rate λ will be stored in a terminal vertex. One of the main issues in obtaining a compact MTBDD representation is the choice of an appropriate variable ordering. A commonly accepted heuristics is an interleaved ordering for the variables encoding source resp. target states, i.e. the ordering of the MTBDD will be: $a_1 \prec a_2 \prec ... \prec a_{n_L} \prec s_1 \prec t_1 \prec s_2 \prec t_2 ... \prec s_{n_S} \prec t_{n_S}$. This ordering, together with a proper treatment of the parallel composition operator, ensures the compactness of the resulting MTBDD [13, 24]

Translating the CASPA-Specification to MTBDDs. The basic procedure is as described in [18]. Here we only discuss the translation of parameterised processes, i.e. we describe our approach of how to represent parameterised processes symbolically. The parse graph structure describes the transitions depending on the parameter values, thereby also describing the possible changes of the parameter values. In CASPA the definition of the transitions is separated from the change of parameters. Let X be a parameterised process, then there is in X's parse graph exactly one node, called $PARAM$ node, which describes the possible transitions. The condition list of a guarded choice is stored in this node. Furthermore, the parse graph may contain several nodes that store the possible changes of the parameter values, called $PARAMDEF$ nodes. In order to generate from this information the actual MTBDD representation of a parameterised process, it is necessary that the generation algorithm keeps track of the current parameter value, which information is taken from the $PARAMDEF$ nodes. The $PARAM$ node serves to determine which transitions are possible in view of the current parameter values. For every satisfied condition, the successor process is determined, and the overall representation of a parameterised process is then a choice over all possible successor processes.

3.2 Data Structures for Measure Representation

For state measures and mean values the main task is to identify the states, resp. their binary encodings, that are relevant for the measure. As many states may contribute to a particular measure, we employ BDDs for a compact representation of the state sets.

```
(1) /* Rate and constant definitions */
(2)  ...
(3)  /* System specification */
(4)  Process      :=  Queue(0)
(5)  Queue(n [3]) :=  [n >= 1] -> (serve,mu); Queue(n-1)
(6)                   [n < 3] -> (arrival,lambda); Queue(n+1)
(7)                   [*] -> (fail,gamma); Repair
(8)  Repair       :=  (repair,rho); Queue(0)
(9)  /* Measure specification */
(10) statemeasure Queuenotfull Queue(n < 3)
(11) meanvalue Fill Queue(n)
(12) throughputmeasure Service serve
```

Fig. 3. Example specification

State Measures. For a given state measure, we first generate its parse graph. Since the state measure is related to one or several process names, each node of the system's parse graph that contains a process which is referenced in the state measure will get a pointer to the respective node in the measure's parse graph. On generation of the MTBDD for the system, the encoding of each process that contains such a pointer is written to the correspondig measure's sub-BDD. After the complete generation of the system's MTBDD representation, the measure's overall BDD is generated by applying the Boolean operators in the measure's parse graph.

Mean Values. For mean values we have to generate for each possible parameter value a BDD that encodes the states in which the parameter has exactly that value. Since processes can have several parameters (and since processes are composed in parallel with other processes), there may be many states in which the parameter of interest has the same value (whereas the values of the remaining parameters, or the states of the other processes, may change). After the generation of the system's MTBDD representation, the measure BDDs are added up, thereby weighing each BDD with the associated parameter value. The result is an MTBDD in which every state encoding is related to its respective parameter value.

Throughput Measures. Throughput measures are not related to specific processes. Therefore no extra BDD for them needs to be generated. The system's MTBDD representation is restricted to the action label whose throughput is to be determined, and the target states are abstracted away. The result is then an MTBDD consisting of the states in which the relevant action is enabled, weighed with the respective transition rates.

3.3 Example

We will clarify the concepts of generating an MTBDD representation for parameterised processes and relating encodings and measures by means of the

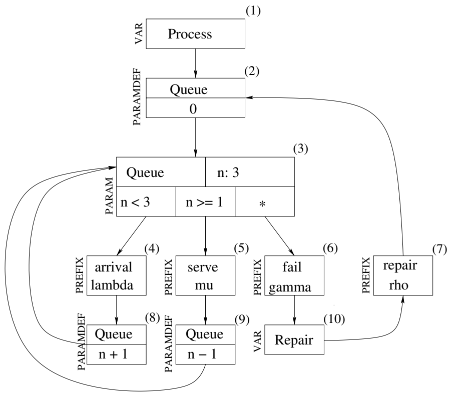

Fig. 4. Parse graph for the specification in Fig. 3

example shown in Fig. 3. The parse graph of the specification can be found in Fig. 4. In the *PARAMDEF* node (2) the parameter value is initialised to zero. In the *PARAM* node (3), when it is visited for the first time, the conditions of the first and the third field are fulfilled, therefore we can generate the MTBDD for their respective successor nodes. To generate the successor node we use the information about the actual parameter value and the change of parameter values of the *PARAMDEF* nodes. In the initial case the successor processes are `(arrival,lambda);Queue(1)` and `(fail,gamma);Repair`. For `Queue(1)` and `Repair` we then compute again the successor processes, and so on. For `Queue(1)` all three conditions are fulfilled and we have three successor processes, namely `(arrival,lambda);Queue(2)`, `(serve,mu);Queue(0)` and `(fail,gamma);Repair`. For the latter two, the successor nodes are already known, whereas for `Queue(2)` the successor processes still have to be computed. The overall MTBDD representation is obtained as a choice between the MTBDD representations of all successor processes which were found.

For the state measure `Queuenotfull` of Fig. 3 the states for `Queue(0)`, `Queue(1)` and `Queue(2)` are relevant. Therefore, on generation of the respective MTBDDs the encodings of these states are copied to the measure BDD. For the mean value measure `Fill` an MTBDD is constructed where the encoding of each reachable state leads to the corresponding value of parameter `n`. Assuming that states are encoded as shown in Fig. 5 (left), the resulting MTBDD for this mean value measure looks as shown in Fig. 5 (right).

3.4 Numerical Analysis and Computation of Measures

Our experience shows that MTBDDs are a suitable data structure for the compact and efficient storing of extremely large state spaces [24]. However, it is known that purely MTBDD-based numerical analysis is very slow [9]. In [22], it was shown that it is possible to combine the advantages of sparse data structures (efficient matrix-vector multiplication) with those of MTBDDs (compact model

State	Encoding
`Queue[0]`	000
`Queue[1]`	001
`Queue[2]`	011
`Queue[3]`	100
`Repair`	010

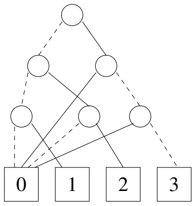

Fig. 5. State encoding (left) and MTBDD for measure `Fill` (right)

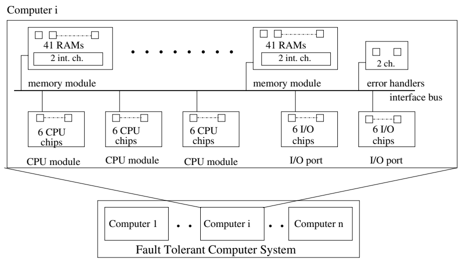

Fig. 6. Configuration of a single computer for the fault-tolerant multi computer system according to [23]

representation), leading to so-called hybrid offset-labelled MTBDDs. These data structures and the associated numerical algorithms were implemented in the stochastic model checker PRISM [20, 21]. Several case studies using PRISM proved the efficiency of the data structures and algorithms, therefore we decided to adopt this approach for CASPA.

In PRISM, and therefore also in CASPA, numerical algorithms for both steady-state and transient analysis are implemented. For steady-state analysis Power, Jacobi, Pseudo-Gauss-Seidel and their overrelaxed versions can be used. For transient analysis uniformisation is employed.

4 Case Studies

In this section we show the applicability of CASPA by means of several case studies: All results were computed on an Intel Pentium IV 3 GHz CPU with 1024 MB RAM, running SuSe 9.0 Linux.

4.1 Fault Tolerant Multi Computer System

This example is based on a case study described originally in [23] and used again in [8]. Due to the different modelling formalisms (stochastic activity networks versus stochastic process algebra), some re-modelling effort was required. The original model consists of n computers each of which has the following components:

- 3 memory modules, of which 2 must be operational
- 3 CPU units, of which 2 must be operational
- 2 I/O ports, of which 1 must be operational
- 2 error-handling chips, which are not redundant.

Each CPU and I/O-port consists of 6 non-redundant chips. Each memory module possesses 41 RAM chips, of which at most 2 may fail, and 2 interface chips that all must be operational. A computer fails, if one of its components fails. The overall system is operational if at least one computer is operational. A diagramatic overview can be found in Fig. 6.

Results. The measure we are interested in is the survival probability of the system. All results were computed using uniformisation with relative precision 10^{-6}. The computation times, including model construction, numerical analysis and measure computation, range from less than one second for the smallest configuration (889 reachable states) to about 30 sec for the largest configuration (750,000 reachable states). In Fig. 7 we see the survival probability for different system configurations: C1 is the configuration of the original system (i.e. consisting of two computers with three memory modules each) which has about 750,000 reachable states. C2 consists of two computers with only one memory module each, and has 2152 reachable states. C3, which is identical to C2 but possesses no redundant I/0 port, has only 889 reachable states. Finally C4 has the same configuration as C2, but consists of 3 computers instead of 2, having 120,000 reachable states. The survival probability, dependent on the mission time, is shown in Fig. 7.

Our results are not directly comparable to the ones reported in [8] (where multi-valued decision diagrams and matrix diagrams are employed as the underlying data structures): Firstly, we consider slightly different system configurations, and secondly, we do not exploit lumpability (which is due to replicated components). However, it is interesting that [8] reports a computation time of 15.5 sec per iteration for a 463,000 state model, while we measured only 0.12 sec per iteration for the 750,000 state model (the machine speeds are almost identical, and ours has only one third of the memory).

4.2 Kanban System

For this case study we computed steady-state probabilities for the well-known Kanban example (originally described in [5]). We model a Kanban system with four cells, a single type of Kanban cards, and the possibility that some workpiece may need to be reworked. The performance measures we compute are the average number of cards in each cell and the throughput of parts with and without rework for each cell.

Results. In Fig. 8 (left) we see the average fill of cells 1 to 4, depending on the number of cards N. Note, that due to the symmetric nature of the model, the

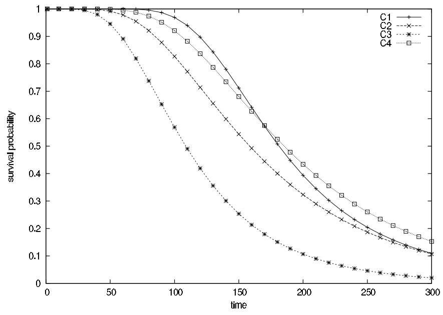

Fig. 7. Survival probability of fault-tolerant multi computer system

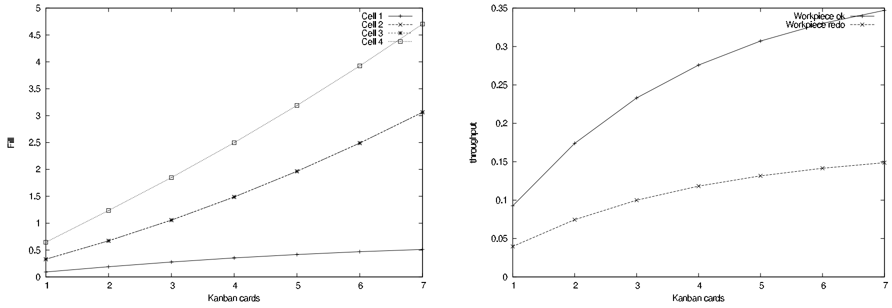

Fig. 8. Left: Average fill of cells 1 to 4. Right: Throughput of workpieces for cell 1

fill of cell 2 and 3 is identical, therefore the two curves are superposed. Fig. 8 (right) shows the throughput measures for cell 1 dependent on the number of cards.

Details about state space size, MTBDD size and time needed for computing the measures can be found in Table 1 (top). The analysis method used here was Pseudo-Gauss-Seidel (which was the most efficient one), with relative precision 10^{-6}.

As a comparison, TIPPtool [12] takes more than 4 hours just to generate the state space of the Kanban model with $N = 4$ (on the same machine), not including numerical analysis and measure computation. This huge difference can be explained by the fact that TIPPtool needs to traverse all transitions of the underlying labelled Markov chain explicitly in a sequential fashion, while CASPA performs a symbolic product space construction followed by a symbolic reachability analysis (where the latter works on sets of states and not on individual states).

We now compare the experimental results obtained with CASPA to those obtained with PRISM, which are shown in Table 1 (bottom). We observe, that the number of MTBDD nodes (column "final") is not the same for both tools:

Table 1. Results for Kanban system

N	Reach. States	MTBDD Nodes peak	MTBDD Nodes final	MTBDD Generation[a]	Iterations	Num. Analysis[b]
CASPA:						
3	58,400	5,739	2,241	0.04 sec.	213	1.63 sec.
4	454,475	14,055	3,990	0.15 sec.	319	20.39 sec.
5	2,546,432	25,514	5,392	0.42 sec.	492	184.44 sec.
6	11,261,376	47,395	8,086	0.94 sec.	625	1191.46 sec.
7	41,644,800	76,230	10,389	1.59 sec.	950	21h
8	133,865,325	116,785	13,998	3.02 sec.	-	-
9	384,392,800	168,694	17,762	4.87 sec.	-	-
10	1,005,927,208	248,461	23,231	8.37 sec.	-	-
11	2,435,541,472	323,115	27,411	12.90 sec.	-	-
12	5,519,907,575	414,719	32,324	17.22 sec.	-	-
PRISM:						
3	58,400	?	2,474	0.12 sec.	230	2.36 sec.
4	454,475	?	4,900	0.49 sec.	370	25.11 sec.
5	2,546,432	?	6,308	1.02 sec.	528	177.60 sec.
6	11,261,376	?	7,876	1.8 sec.	891	1140.48 sec.
7	41,644,800	?	9,521	3.11 sec.	-	-
8	133,865,325	?	14,702	6.10 sec.	-	-
9	384,392,800	?	17,196	8.62 sec.	-	-
10	$1,005,927,208$	?	19,877	12.43 sec.	-	-
11	2,435,541,472	?	22,666	17.79 sec.	-	-
12	5,519,907,575	?	25,710	24.20 sec.	-	-

[a] including reachability analysis
[b] including measure computation (in the case of CASPA)

While CASPA constructs smaller MTBDDs for small values of N, the converse is the case for larger values of N. We attribute this phenomenon partly to the different state space ordering as caused by the state space generation algorithms of the two tools, and partly to the fact that CASPA's symbolic representation includes the encoding of the action labels which is not the case for PRISM. In the case of CASPA, the peak number of MTBDD nodes during the construction process (column "peak") is much higher than the final number of nodes, but it is still extremely small compared to the size of the state space. Since PRISM does not give peak numbers, the corresponding positions are marked with a "?". State space construction (column "MTBDD Generation") is faster in CASPA, but this is not really significant since state space construction is not the bottleneck. The number of iterations performed is always smaller in CASPA, which again seems to be due to the state space ordering (we emphasize at this point, that finding the optimal state space ordering is an NP-complete problem, therefore one can only resort to heuristics as described e.g. in [13]). A comparison of the total times for numerical analysis is not of interest for two reasons: Firstly, CASPA-times include measure computation while PRISM-times do not. Secondly, the numerical engine of CASPA is taken from PRISM, so the implementations are practically identical.

Table 2. Results for Polling system

N	Reach. States	MTBDD Nodes peak	MTBDD Nodes final	MTBDD Generation[a]	Iterations	Num. Analysis[b]
CASPA:						
10	15,360	3,748	1,193	0.01 sec.	185	0.29 sec.
15	737,280	9,417	2,370	0.05 sec.	150	13.02 sec.
20	31,457,280	19,311	4,556	0.17 sec.	161	741.27 sec.
PRISM:						
10	15,360	?	931	0.02 sec.	74	0.29 sec.
15	737,280	?	1,942	0.05 sec.	97	8.27 sec.
20	31,457,280	?	3,346	0.15 sec.	112	663.65 sec.

[a] including reachability analysis

[b] including measure computation (in the case of CASPA)

4.3 Polling System

As a third case study we consider a polling system, consisting of a server and N stations which are served in a cyclic fashion. This system was originally described in [16] and has since been frequently used as a standard benchmark. The results are shown in Table 2 (where the numerical method was again Pseudo-Gauss-Seidel). We observe that the MTBDDs generated by CASPA are up to 36% larger than the ones generated by PRISM, and that CASPA always needs more iterations. However, quite surprisingly, the time per iteration is always smaller in the case of CASPA.

5 Conclusions

In this paper we have presented CASPA, a stochastic process algebra tool which realises a completely symbolic approach from model construction to the computation of performance and dependability measures. CASPA implements an MTBDD-based state space generation scheme and allows the computation of transient and steady-state measures on the basis of well-established numerical algorithms.

We carried out systematic tests on many case studies: In addition to the ones given here, several queueing models, a mainframe system with software failures and a wireless communication network were analysed using CASPA. The results for all case studies are very positive, both state space generation and numerical analysis have shown to be highly efficient. CASPA is clearly superior to sparse-matrix based tools such as TIPPtool, and it compares well to other symbolic tools such as PRISM.

We are currently working on extending CASPA with a symbolic stochastic model checking engine. We plan to support the powerful action-based logic sPDL [19] which enables the user to define complex performance and dependability requirements in a formal and concise way.

Last but not least, in order to further increase its usability, in the future CASPA shall be equipped with a graphical user interface.

Acknowledgement

The authors would like to cordially thank Dave Parker for making the numerical solution code of PRISM available to the scientific community.

References

[1] AT&T. Drawing graphs with dot http://www.research.att.com. 297

[2] R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic Decision Diagrams and their Applications. *Formal Methods in System Design*, 10(2/3):171–206, April/May 1997. 297

[3] R. E. Bryant. Graph-based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986. 293, 297

[4] G. Ciardo, R. L. Jones, A. S. Miner, and R. Siminiceanu. SMART: Stochastic Model Analyzer for Reliability and Timing. In *Tools of Aachen 2001 Int. Multiconference on Measurement, Modelling and Evaluation of Computer Communication Systems*, pages 29–34, 2001. 294

[5] G. Ciardo and M. Tilgner. On the use of Kronecker operators for the solution of generalized stochastic Petri nets. Technical Report 96-35, ICASE, 1996. 302

[6] daVinci V2.1. http://www.informatik.uni-bremen.de/davinci/. 297

[7] L. de Alfaro, M. Kwiatkowska, G. Norman, D. Parker, and R. Segala. Symbolic Model Checking for Probabilistic Processes using MTBDDs and the Kronecker Representation. In *TACAS'2000*, pages 395–410. Springer LNCS 1785, 2000. 293

[8] S. Derisavi, P. Kemper, and W. H. Sanders. Symbolic State-Space Exploration and Numerical Analysis of State-Sharing Composed Models. In *Fourth Int. Conf. on the Numerical Solution of Markov Chains*, pages 167–18167–189, 2003. 294, 301, 302

[9] E. Frank. Codierung und numerische Analyse von Transitionssystemen unter Verwendung von MTBDDs. Student's thesis, Universität Erlangen–Nürnberg, IMMD VII, 1999 (in German). 300

[10] M. Fujita, P. McGeer, and J. C.-Y. Yang. Multi-terminal Binary Decision Diagrams: An efficient data structure for matrix representation. *Formal Methods in System Design*, 10(2/3):149–169, April/May 1997. 297

[11] S. Gilmore and L. Kloul. A unified tool for performance modelling and prediction. In *Proc. 22nd Int. Conf. on Computer Safety, Reliability and Security (SAFECOMP 2003)*, pages 179–192, 2003. LNCS 2788. 294

[12] H. Hermanns, U. Herzog, U. Klehmet, V. Mertsiotakis, and M. Siegle. Compositional performance modelling with the TIPPtool. *Performance Evaluation*, 39(1-4):5–35, January 2000. 294, 295, 303

[13] H. Hermanns, M. Kwiatkowska, G. Norman, D. Parker, and M. Siegle. On the use of MTBDDs for performability analysis and verification of stochastic systems. *Journal of Logic and Algebraic Programming*, 56(1-2):23–67, 2003. 293, 298, 304

[14] H. Hermanns, J. Meyer-Kayser, and M. Siegle. Multi Terminal Binary Decision Diagrams to Represent and Analyse Continuous Time Markov Chains. In *3rd Int. Workshop on the Numerical Solution of Markov Chains*, pages 188–207. Prensas Universitarias de Zaragoza, 1999. 293

[15] H. Hermanns and M. Rettelbach. Syntax, Semantics, Equivalences, and Axioms for MTIPP. In *Proc. of PAPM'94*, pages 71–88. Arbeitsberichte des IMMD 27 (4), Universität Erlangen-Nürnberg, 1994. 294, 295

[16] O. C. Ibe and K. S. Trivedi. Stochastic Petri Net Models of Polling Systems. *IEEE Journal on Selected Areas in Communications*, 8(9):1649–1657, December 1990. 305

[17] J.-P. Katoen, M. Kwiatkowska, G. Norman, and D. Parker. Faster and Symbolic CTMC Model Checking. In *PAPM-PROBMIV'01*, pages 23–38. Springer, LNCS 2165, 2001. 293

[18] M. Kuntz and M. Siegle. Deriving symbolic representations from stochastic process algebras. In *Process Algebra and Probabilistic Methods, Proc. PAPM-PROBMIV'02*, pages 188–206. Springer, LNCS 2399, 2002. 297, 298

[19] M. Kuntz and M. Siegle. A stochastic extension of the logic PDL. In *Sixth Int. Workshop on Performability Modeling of Computer and Communication Systems (PMCCS6)*, pages 58–61, Monticello, Illinois, 2003. 305

[20] M. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic Symbolic Model Checker. In *MMB-PNPM-PAPM-PROBMIV Tool Proceedings*, pages 7–12. Univ. Dortmund, Informatik IV, Bericht 760/2001, 2001. 294, 301

[21] M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic Symbolic Model Checking with PRISM: A Hybrid Approach. In *TACAS'2002*, pages 52–66. Springer LNCS 2280, April 2002. 294, 301

[22] D. Parker. *Implementation of symbolic model checking for probabilistic systems.* PhD thesis, School of Computer Science, Faculty of Science, University of Birmingham, 2002. 296, 297, 300

[23] W. H. Sanders and L. M. Malhis. Dependability evaluation using composed SAN-based reward models. *Journal of Parallel and Distributed Computing*, 15(3):238–254, 1992. 301

[24] M. Siegle. *Behaviour analysis of communication systems: Compositional modelling, compact representation and analysis of performability properties.* Shaker Verlag, Aachen, 2002. 293, 297, 298, 300

[25] F. Somenzi. CUDD: Colorado University Decision Diagram Package, Release 2.3.1. User's Manual and Programmer's Manual, February 2001. 296

[26] E. Werner. Leistungsbewertung mit Multi-Terminalen Binären Entscheidungsdiagrammen. Master's thesis, Universität Erlangen–Nürnberg, Informatik 7, 2003 (in German). 296