An Efficient Symbolic Elimination Algorithm for the Stochastic Process Algebra tool CASPA

Jens Bachmann, Martin Riedl, Johann Schuster, Markus Siegle

University of the German Federal Armed Forces Munich
Department of Computer Science
{martin.riedl, johann.schuster, markus.siegle}@unibw.de

Abstract. CASPA is a stochastic process algebra tool for performance and dependability modelling, analysis and verification. It is based entirely on the symbolic data structure MTBDD (multi-terminal binary decision diagram) which enables the tool to handle models with very large state space. This paper focuses on an extension of CASPA's modelling language by weighted immediate actions. We discuss the pertaining semantics and present an efficient symbolic algorithm for the elimination of vanishing states. A non-trivial case study illustrates the usage features of CASPA, from graphical model specification to numerical analysis.

Keywords: stochastic process algebra, MTBDD, elimination of immediate transitions

1 Introduction

CASPA is a tool for performance and dependability modelling, based on a stochastic process algebra. Its development began in 2003 [16], and it has since been the environment in which its developers have experimented extensively with symbolic (i.e. MTBDD-based) techniques that enable the tool to generate and analyse Markov chains with very large state spaces in a highly efficient manner [10]. In addition to classical analysis methods, model checking algorithms have been realised in CASPA, employing stochastic temporal logics with novel features for characterising accepting paths [9]. However, up to now, acceptance of CASPA by a broader community has been limited, due to its lacking of two features: (1) CASPA's process algebraic specification language had been purely textual, which made it difficult to use for inexperienced users. (2) So far, the modelling language of CASPA supported only timed, i.e. Markovian actions, whereas immediate actions, which are often very handy for modelling instantaneous process interaction, had been missing. As a consequence, when synchronisation of processes was modelled by Markovian actions with "fast" rates, state spaces often became unnecessarily large.

The first problem has recently been dealt with by implementing a new user interface, realised with state-of-the-art software development techniques (Eclipse Graphical Modelling Framework), offering graphical model specification and easy-to-use interaction with the tool [2].

The present paper focuses on the second issue: We define an extended specification language, offering both Markovian and weighted immediate actions, and give the

```
(1)   int max      = 12;
(2)   rate alpha   = 5; rate gamma = 5.4; rate fi = 3.2;
(3)   weight slow  = 2; weight fast = 3;
(4)   System:= hide s in (P(max) |[s]| Q(0))
(5)   P(n [max]) :=[*]      -> (a,alpha); P(n) + (b,7.0); stop
(6)                 [n>0]    -> (c,n*gamma); P(n-1)
(7)                 [n<max]  -> (*s,1*); (d,0.3); P(n+1) + (*t,1*); (d,0.4); P(n+1)
(8)   Q(m [10]) := [m>1]    -> (e,1.38); ((*s,slow*); Q(m-1) + (*s,fast*); Q(m-2))
(9)                 [m=1]    -> (e,1.38); Q(0)
(10)                [m=0]    -> (f,fi); Q(5)
(11)  statemeasure nonsense (P(n>1) & !Q(m=4))
(12)  meanvalue Pmean P(n)
(13)  throughputmeasure tput_c c
```

Fig. 1: Specification toy example

pertaining semantics. Most importantly, we propose an efficient algorithm for the elimination of vanishing states (which are caused by the immediate actions and need to be removed before numerical analysis can take place). The elimination algorithm consists of a very efficient fully symbolic main phase, followed by a semi-symbolic post-processing phase which is responsible for the elimination of cycles of immediate transitions – if such cycles exist.

The paper is organised as follows: Section 2 introduces the syntax and semantics of CASPA's extended textual specification language. Section 3 is devoted to the symbolic algorithm for the elimination of vanishing states. Section 4 gives a sketch of the new graphical model specification features of CASPA and presents a non-trivial application case study that illustrates the efficiency of the implemented symbolic data structures and algorithms, and Section 5 concludes the paper.

2 Modelling language and its semantics

2.1 Informal overview

The specification language of CASPA is a stochastic process algebra (SPA), originally based on the stochastic process algebra TIPP [6]. The syntax for the purely Markovian version is described in [10]. The current extensions to the input language are:

- Actions can be specified both as Markovian actions ($\langle$ACTION$\rangle$, $\langle$RATE$\rangle$) or as immediate actions (* $\langle$ACTION$\rangle$, $\langle$WEIGHT$\rangle$ *).
- Processes can synchronise by both immediate and Markovian actions.
- Corresponding to the two different action types, constants can be specified either as *rates* (for Markovian actions) or *weights* (for immediate actions).

A toy example of the specification language is given in Fig. 1 (the example has no real meaning, it is purely for demonstrating the use of the language elements): In line (1) a global parameter constant is defined, line (2) and (3) define constants for rates and weights. The entire system is defined in line (4), where the parametrised processes $P(max)$ and $Q(0)$ are composed in parallel and synchronised by the common action *s*, i.e. action *s* has to be performed by $P(max)$ and $Q(0)$ at the same time. The system specification also shows the hide operator, that replaces action *s* after the parallel composition by the internal immediate action `tau`. In line (5) a guard with parameter * is

given which means that this branch is possible for any process parameter value. This line also shows the prefix operator ; that separates sequential actions and the choice operator + that chooses either the branch starting with the Markovian action *a* leading to P with the same parameter n or the one starting with the Markovian action *b* leading to the `stop` keyword that disables any further action of P (; binds more tightly than +). The action in line (6) can only be taken if $n > 0$ holds. Its rate gamma is scaled by the current value of the process parameter n. Line (7) shows the specification of an immediate action with weight 1. The specification of $Q(m[10])$ can be read in a similar way. Some examples of measures are defined in lines (11) to (13) of the example. The first measure is the steady state probability of states fulfilling the given condition on the process parameter values. The second one calculates the mean value of process parameter n of P. The last measure calculates the throughput of action c.

The resulting model has tangible and vanishing states, where a state is called vanishing if and only if it has at least one outgoing immediate transition (non-vanishing states are called tangible). In CASPA, a model with both immediate and Markovian actions has to satisfy the following constraints:

- The initial state of the system must not be vanishing.
- Immediate and Markovian actions with the same name are not allowed.
- As there is no bisimulation equivalence relation for internal Markovian transitions, hiding of Markovian actions is not supported and the internal action `tau` has to be immediate.
- Throughput measures are only allowed for Markovian actions.
- As in the previous releases of CASPA, recursion over static operators (e.g. parallel composition and hiding) is not allowed (to keep the state space finite).

2.2 Formal semantics

As described in [10], CASPA directly derives an MTBDD representation from a given SPA without generating an explicit Stochastic Labelled Transition System (SLTS) first. This is done by means of a denotational semantics [8]. When a model with both immediate and Markovian actions is given as input, CASPA generates two MTBDDs: One for the immediate and one for the Markovian transitions, again without building the corresponding ESLTS (extended SLTS) first. To describe the semantics, we give the following

Definition 1 (weighted ESLTS) *Let S be a finite set of states, $s_0 \in S$ the initial state, L_I and L_M finite sets of action labels with $L_I \cap L_M = \emptyset$ and finally $\rightarrow \subseteq S \times L_M \times \mathbb{R}^{>0} \times S$ and $\dashrightarrow \subseteq S \times L_I \times \mathbb{R}^{>0} \times S$ be transition relations. We call $T = (S, L_M, L_I, \rightarrow, \dashrightarrow, s_0)$ a weighted ESLTS. For $(s, a, w, t) \in \dashrightarrow$ we write $s \overset{a,w}{\dashrightarrow} t$ and call it an immediate a-transition from state s to state t with weight w. Analogously we write $s \overset{b,\lambda}{\rightarrow} t$ for $(s, b, \lambda, t) \in \rightarrow$ and call it a Markovian b-transition from state s to state t with rate λ.*

Let $\mathcal{S}_M \subseteq L_M$ and $\mathcal{S}_I \subseteq L_I$ be sets of synchronising action labels. The semantics for Markovian transitions is as usual: Concurring Markovian actions correspond to a

race condition [14]. Parallel composition of Markovian transitions leads to the multiplication of rates as shown in Eq. 1 (left). This approach includes the possibility to set one of the partner rates to 1, modelling passive cooperation (which is the concept realised in EMPA [3]). There also exist other synchronisation policies [7], but in our experience, the multiplication of rates is general, flexible and very useful for practical modelling.

$$\frac{P \overset{a,\lambda}{\rightarrow} P' \qquad Q \overset{a,\mu}{\rightarrow} Q'}{P|[\mathcal{S}_M]|Q \overset{a,\lambda\cdot\mu}{\rightarrow} P'|[\mathcal{S}_M]|Q'} \quad a \in \mathcal{S}_M \qquad \frac{P \overset{a,w}{\dashrightarrow} P' \qquad Q \overset{a,v}{\dashrightarrow} Q'}{P|[\mathcal{S}_I]|Q \overset{a,w\cdot v}{\dashrightarrow} P'|[\mathcal{S}_I]|Q'} \quad a \in \mathcal{S}_I \quad (1)$$

Immediate transitions are treated in a similar way, as shown in Eq. 1 (right). The intuition behind our weighted approach is that a parallel step $(*a,w_1*);P|[a]|(*a,w_2*);Q$ should get a weight $w_1 \cdot w_2$. Thereby the weights of unsynchronised actions remain the same. In a post-processing step, when the overall system has been composed and no further parallel composition takes place (a closed system), all outgoing weights of a state are normalised to a sum of 1. So the concurring weights are mapped to a discrete probability distribution. Our model should exhibit asynchronous behaviour, i.e. our synchronising processes should be allowed to act independently without assuming explicit idle-step of processes resting in their states (so we did not follow the synchronous approach described in [15]). The drawback of our weighted approach is that it is not scale-free. That means that multiplying all weights within a subprocess by the same factor may lead to a different overall system. As an example, take the synchronisation of $(P(n)|[s]|Q(m))$ with P and Q as defined in Fig. 1. Consider the case where $n < max$, $m > 1$ and Q has just performed the transition $(e, 1.38)$. So Q is waiting for the synchronisation with P. There are three possibilities for the next immediate transition:

1. $((d, 0.3); P(n+1))|[s]|Q(m-1)$, slow s transition (synchronised)
2. $((d, 0.3); P(n+1))|[s]|Q(m-2)$, fast s transition (synchronised)
3. $((d, 0.4); P(n+1))|[s]|((*s, slow*); Q(m-1) + (*s, fast*); Q(m-2))$, t transition (only P)

with the probabilities $\frac{1}{6}(1 \cdot 2, 1 \cdot 3, 1) = (\frac{1}{3}, \frac{1}{2}, \frac{1}{6})$. When changing the values of fast and slow in Q but not their ratio, our approach leads to a different overall behaviour. Multiplying the weights of Q by 2 leads to the transition probabilities $\frac{1}{11}(1 \cdot 4, 1 \cdot 6, 1) = (\frac{4}{11}, \frac{6}{11}, \frac{1}{11})$ which are different from the probabilities of the first configuration.

3 Elimination algorithm for immediate transitions

The description of the algorithm is built upon a standard MTBDD representation of a labelled transition system. Thus, the algorithm is applicable not only to CASPA models, but also to finite-state models generated from other specification languages (e.g. elimination of vanishing markings for GSPNs [11]). Enumerating all the states and all the actions by positive integers, a Markovian transition $s \overset{a,\lambda}{\rightarrow} t$ can be seen as a function $f : \mathbb{N}^3 \rightarrow \mathbb{R}; (a, s, t) \mapsto \lambda$. The same holds for immediate transitions where the value of the function denotes the weight instead of the rate. The integers can subsequently be encoded in binary form thus leading to a MTBDD representation

of f. For the details consider e.g. [14]. In the description of the algorithm the following name conventions for MTBDD variables are used: $\boldsymbol{a}$ (action labels), $\boldsymbol{s}$ (source states), $\boldsymbol{t}$ (target states), $\boldsymbol{u}$ (temporary states). The variable ordering in the MTBDD is $a_1 \prec \ldots \prec a_n \prec s_1 \prec t_1 \prec u_1 \ldots \prec s_m \prec t_m \prec u_m$ according to commonly accepted heuristics. The following MTBDD operations are used to describe the algorithm (see e.g. [14] for a more detailed explanation):

- $\langle$MTBDD1$\rangle\langle$OP$\rangle\langle$MTBDD2$\rangle$, the general apply operator
- $\langle$MTBDD$\rangle_{\langle VAR\rangle=\langle VAL\rangle}$, returns $\langle$MTBDD$\rangle$ with $\langle$VAR$\rangle$ set to $\langle$VAL$\rangle$ (Restriction)
- $ABSTRACT(\langle$MTBDD$\rangle, \langle$VAR$\rangle, \langle$OP$\rangle)$:= $\langle$MTBDD$\rangle_{\langle VAR\rangle=0}\langle$OP$\rangle\langle$MTBDD$\rangle_{\langle VAR\rangle=1}$
- $THRESHOLD(\langle$MTBDD$\rangle, \langle$VAL$\rangle)$ generates a 0-1-MTBDD with value 1 where the function represented by the MTBDD is $>\langle$VAL$\rangle$, value 0 elsewhere

Abstraction and restriction of more than one variable is defined in the canonical recursive way. The elimination algorithm used in CASPA is a combination of the fully symbolic approach sketched in [14] and an adaptation of a semi-symbolic algorithm which was first presented in [4]. The fully symbolic algorithm is a fast method to eliminate all the vanishing states without vanishing predecessors and the semi-symbolic algorithm eliminates the remaining (i.e. loop or cycle) immediate transitions, that are usually quite few (a loop is a cycle of length one).

The next subsections describe the steps to eliminate all the immediate transitions in the given ESLTS. In the following, let M^M be the MTBDD encoding the Markovian transitions and M^I the MTBDD encoding the immediate transitions.

3.1 Precomputations

In the overall, closed system model no further parallel composition may take place, so no immediate transitions are rendered inactive due to synchronisation constraints. By the assumption of maximal progress [5], Markovian transitions that emanate from states with outgoing immediate transitions can be disregarded. The following lines show how this is done symbolically:

- Source states of immediate transitions are determined by $S^I = THRESHOLD(ABSTRACT(M^I, (\boldsymbol{a}, \boldsymbol{t}), +), 0)$
- Markovian transitions concurring with immediate transitions are removed by $M^M = M^M \cdot (1 - S^I)$, where $(1 - S^I)$ denotes the complement of S^I

Now, for every vanishing state, the weights of all outgoing immediate transitions are normalised to probabilities. This is achieved by the following symbolic operations:

- Abstract from the immediate action labels (*ST* indicates that this MTBDD only depends on s and t variables): $M^{I_{ST}} = ABSTRACT(M^I, (\boldsymbol{a}), +)$
- Sum up the exit weights for every state (*S* indicates that this MTBDD only depends on s variables): $M^{I_S} = ABSTRACT(M^{I_{ST}}, (\boldsymbol{t}), +)$
- Divide the outgoing weights by the sum of exit weights to get the new immediate transition system: $M^{I_{new}} = M^{I_{ST}} / M^{I_S}$

1: $S^{I_{exit}}$=THRESHOLD(ABSTRACT($M^{I_{new}}$, ($\boldsymbol{t}$), +), 0)
2: $T^{I_{target}}$=THRESHOLD(ABSTRACT($S^{I_{exit}} \cdot M^{I_{new}}$, ($\boldsymbol{s}$), +), 0)
3: T^{I}=swapSourceAndTargetVariables($S^{I_{exit}}$)
4: $T^{el} = T^{I} \cdot (1 - T^{I_{target}})$
5: S^{el} = swapSourceAndTargetVariables(T^{el})
6: $M^{M_{left}}$=$M^{M} \cdot (1 - T^{el})$
7: $M^{I_{left}}$=$M^{I_{new}} \cdot (1 - S^{el})$
8: $M^{I_{temp}}$=changeVariables($M^{I_{new}}$, ($\boldsymbol{t} \mapsto \boldsymbol{u}$), ($\boldsymbol{s} \mapsto \boldsymbol{t}$))
9: $M^{M_{redir}}$=changeVariables(ABSTRACT($T^{el} \cdot M^{M} \cdot M^{I_{temp}}$, ($\boldsymbol{t}$), +), ($\boldsymbol{u} \mapsto \boldsymbol{t}$))
10: $M^{M} = M^{M_{redir}} + M^{M_{left}}$
11: $M^{I_{new}} = M^{I_{left}}$

Fig. 2: Fully-symbolic elimination step

3.2 Fully-symbolic elimination step

The algorithm is a round-based scheme where in every round all immediate transitions without immediate predecessor transition are eliminated at once. The number of rounds is determined by the maximum length of sequences of immediate transitions (without cycles), and *not* by the number of vanishing states. Such a round-based scheme is typical for symbolic algorithms and works very efficiently. The algorithm is performed until only immediate transitions with at least one immediate predecessor remain. The algorithm in Fig. 2 shows one elimination round, the loop that repeats the procedure is omitted. In line (1) the source states of immediate transitions $S^{I_{exit}}$ are calculated from $M^{I_{new}}$ by abstracting over the target variables, disregarding the actual weights. Line (2) calculates the target states of immediate transitions by looking at the possible transitions emanating from $S^{I_{exit}}$, abstracting over the source states and again disregarding the actual weights. $T^{I_{target}}$ depends only on $\boldsymbol{t}$ variables. The MTBDD $S^{I_{exit}}$ that depends only on $\boldsymbol{s}$ variables is swapped to $\boldsymbol{t}$ variables in line (3). An important immediate result is calculated in line (4): vanishing states without incoming immediate transitions. Just a variable swapping from target to source states is performed in line (5). Lines (6) and (7) calculate the Markovian and immediate transitions that are left unchanged in the current round. The most important steps are line (8) where the immediate transitions are swapped to an alternative variable set and line (9) that performs all the redirections and rescalings for this round via the $\boldsymbol{t}$ variables. After abstraction of the $\boldsymbol{t}$ variables, the $\boldsymbol{u}$ variables are swapped back to $\boldsymbol{t}$ and the resulting MTBDD only depends on $\boldsymbol{s}$ and $\boldsymbol{t}$ variables Finally in line (10) the new set of Markovian transitions is calculated and in line (11) the immediate transitions are reduced to the set that has not been eliminated. This procedure is repeated until T^{el} is equal to zero.

3.3 Semi-symbolic elimination step

It remains to eliminate the loops and cycles of immediate transitions. The elimination has to be done in a semi-symbolic way as vanishing states within a cycle have to be eliminated one after another. If a vanishing state has an immediate self-loop, the loop first has to be resolved before the elimination is possible. Self-loops are removed by a geometric series argument.

The semi-symbolic algorithm uses an MTBDD containing the source states of the remaining immediate transitions as a trigger. This MTBDD is calculated symbolically by $S^I = ABSTRACT(M^{I_{new}}, (\boldsymbol{t}), +)$. The initial call for the elimination algorithm is eliminate(0, S^I, 1), where 1 means a MTBDD representing the constant 1. The corresponding subroutine is shown in Fig. 3 (top). The parameter *bit* encodes a source state

```
 1: eliminate(bit, Trigger, S)
 2: if Trigger is not constant node then
 3:    eliminate(bit + 2, Trigger_{v_bit=0}, S · v̄_bit)
 4:    eliminate(bit + 2, Trigger_{v_bit=1}, S · v_bit)
 5: else
 6:    if NOT value(Trigger) == 0 then
 7:       if bit <NumStateVariables then
 8:          eliminate(bit + 2, Trigger, S · v̄_bit)
 9:          eliminate(bit + 2, Trigger, S · v_bit)
10:       else
11:          S^swap=swapSourceAndTargetVariables(S)
12:          M^{I_S}_curr=M^{I_new} · S
13:          M^{I_S}_curr = ABSTRACT(M^{I_S}_curr, (s), +)
14:          Loop = M^{I_S}_curr · S^swap
15:          M^{I_S}_curr = M^{I_S}_curr·(1-THRESHOLD(Loop, 0))
16:          Loop = ABSTRACT(Loop, (t), +)
17:          if NOT value(Loop)==0 then
18:             M^{I_S}_curr = M^{I_S}_curr · 1/(1 − value(Loop))
19:          end if
20:          redirect(M^M, S^swap, M^{I_S}_curr)
21:          redirect(M^{I_new}, S^swap, M^{I_S}_curr)
22:       end if
23:    end if
24: end if
```

```
1: redirect(M, T, I)
2: M_changed = M_{t=T}
3: if NOT M_changed == 0 then
4:    M = M·(1-THRESHOLD(M_changed · T, 0))
5:    M = M + M_changed · I
6: end if
```

Fig. 3: Semi-symbolic elimination step

variable or constant level in the MTBDD, *Trigger* is the node in the trigger MTBDD that is currently processed and *S* is a MTBDD used to encode source states of immediate transitions. In the following the notation v_i means the MTBDD variable corresponding to the i-th bit of the state variables in the MTBDD, $\overline{v}_i$ denotes its negation. Line (2) checks whether *Trigger* is a non-constant node. As long as no constant node of *Trigger* is reached, the function eliminate is called recursively with the restrictions of *Trigger* and *S* to the possible assignments of v_{bit} in lines (3) and (4). When a non-zero constant node is reached but not all state variable bits are already processed, these *don't care*

levels are resolved recursively in lines (8) and (9). Once the source state of a certain immediate transition is encoded in S and the corresponding terminal node of the trigger MTBDD is not 0, the elimination of S takes place in lines (11)-(22). In line (11) S is encoded as target state. Line (12) calculates the immediate transitions $M^{I_S}_{curr}$ emanating from S. Immediate loops may occur during the elimination process. They are calculated in lines (13)-(14), eliminated from $M^{I_S}_{curr}$ in line (15) and finally $M^{I_S}_{curr}$ is rescaled in lines (16)-(19) by the limit of the corresponding infinite geometric series. Lines (20) and (21) redirect the Markovian and immediate transition systems according to $M^{I_S}_{curr}$. The redirect subroutine is given in Fig. 3 (bottom). Line (2) determines the transitions in *M* that end at the currently processed state and therefore have to be redirected. Here the restriction sets all $\boldsymbol{t}$-variables in M to the assignment given by state T. If there are transitions to change, line (4) removes the transitions leading to state *T* from *M* and line (5) adds the redirected transitions to *M*. Note that I depends only on t variables whereas $M_{changed}$ only depends on s variables. Therefore the product of both again depends on s and t variables.

4 Tool and Application case study

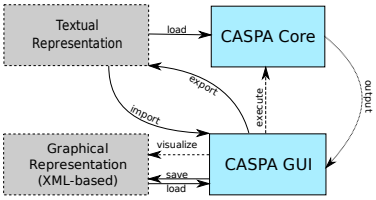


Fig. 4: CASPA Architecture

The CASPA Core component includes the parser for the SPA language, the symbolic state space generator and manipulator (including the elimination of vanishing states), as well as the analysis components, most notably the numerical analysis methods for computing steady-state and transient solutions. Recently, the CASPA Core component has been extended by a graphical user interface (GUI) [2], such that the overall architecture is as shown in Fig. 4.

CASPA and its GUI extension now allow the user to

- specify and edit models graphically,
- import/export textual representations of CASPA models,
- serialise the GUI-model including its layout information,
- and to call the CASPA Core component to perform the model analysis.

The graphical specification of a system takes place on two layers:

- On **System-Layer** the overall model is represented as a rooted directed graph with sequential processes as its leaves.
- On **Process-Layer** the sequential processes are represented as labelled transition diagrams.

Figure 5 and 6 are exemplary screenshots of parts of our case study model as one can visualize or specify them using CASPA GUI. The details of the case study follow in the next sections.

4.1 An accident and emergency department

As a case study, we consider the accident and emergency department (AED) of a large hospital. Our model is based on the one described in [1], i.e. the basic structure and

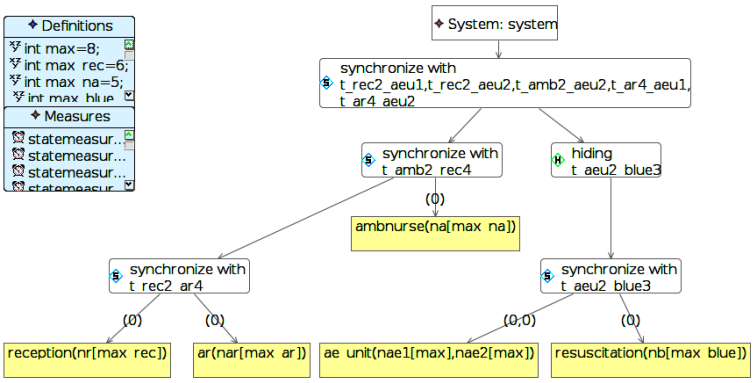


Fig. 5: CASPA GUI System Layer Example

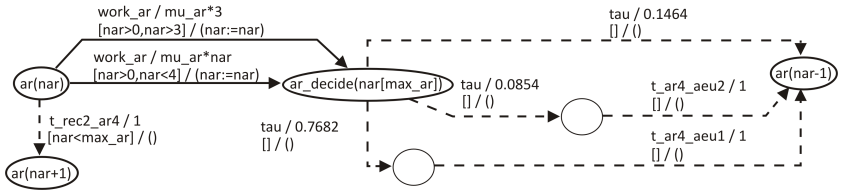
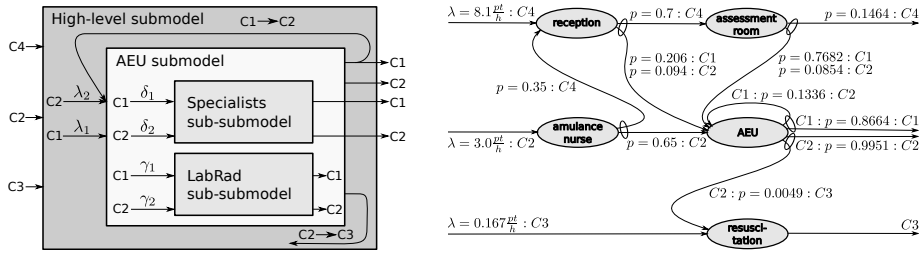


Fig. 6: CASPA GUI Process Layer Example

behaviour, as well as the numerical parameters, are taken from there. Contrary to [1], we model the AED as a three-level hierarchical SPA model as shown in Fig. 7a. This is done in order to keep the state space at a manageable size, i.e. to avoid the construction of the complete state space of the overall model.

In the high-level model (see Fig. 7b, which corresponds to Fig. 5, seen here as a queueing network), the flow of patients (belonging to four different classes) through the AED is shown. Patients with minor (major) injuries or illnesses belong to class 1 (2), emergency patients requiring resuscitation belong to class 3 and patients yet to be classified belong to class 4. Patients enter the AED either through the reception, by (ordinary) ambulance or by (emergency) ambulance blue call. While in the AED, it is possible for patients to switch their class, e.g. moving from class 4 to class 1 or 2, from class 1 to 2 or from class 2 to 3. The high-level model contains the nodes “reception”, “assessment room”, “ambulance nurse”, “accident and emergency unit (AEU)” and “resuscitation”, each of which is modelled as a multi-server queueing station for one or more patient classes. A node’s behaviour is modelled by a multi-parameter CASPA process, where a particular parameter is used to keep track of the current number of patients of a particular class. The transition of patients from one node to another is modelled by synchronising immediate actions, e.g. the action `t_rec2_ar4` models the transit of a class 2 patient from the reception (`rec`) to the assessment room (`ar`) which he or she enters as a class 4 patient. As an example, consider the “assessment room” process shown in Fig. 8 (which is the textual version of Fig. 6). Since only class 4 patients enter



(a) Hierarchical structure (b) High-level model

Fig. 7: AED model

```
ar(nar[max_ar]) :=
/* arriving from the reception as class 4 patient: */
   [nar<max_ar] -> (*t_rec2_ar4,1*); ar(nar+1)
/* the assessment nurses perform their work, */
/* where a maximum of 3 assessment nurses are working concurrently */
   [nar>0, nar<4] -> (work_ar,mu_ar*nar); ar_decide(nar)
   [nar>0, nar>3] -> (work_ar,mu_ar*3); ar_decide(nar)
ar_decide(nar[max_ar]) :=
/* leaving the AED: */
   [*] -> (*tau,0.1464*);ar(nar-1)
/* transfering to AEU as class 1 patient: */
   [*] -> (*tau,0.7682*);(*t_ar4_aeu1,1*);ar(nar-1)
/* transfering to AEU as class 2 patient: */
   [*] -> (*tau,0.0854*);(*t_ar4_aeu2,1*);ar(nar-1)
```

Fig. 8: CASPA code describing the activities in the assessment room

the assessment room, a single process parameter (`nar`) suffices to keep track of the number of patients. Transitions into the assessment room are modelled by immediate action `t_rec2_ar4`, and transition from the assessment room to the AEU as class 1/2 patients are modelled by actions `t_ar4_aeu1` and `t_ar4_aeu2`, respectively.

4.2 Hierarchical modelling approach

The AEU node of the high-level model is special in that it is the flow-equivalent of the AEU submodel. The `AEU` submodel, similar to the high-level model, consists of a number of interacting nodes, of which `LabRad` (lab test and radiology unit) and `Specialists` (three different kinds of medical specialists) in turn are flow-equivalent servers of two further sub-submodels.

For the analysis, we proceed as follows: In a first (top-down) step the effective arrival rates at the `AEU` submodel are calculated by hand from the external patient arrival rates at the high-level model, taking into account the routing probabilities and class switching probabilities in the high-level model (this can be seen as solving the network's traffic equations). Similarly, at the next lower level of the hierarchy, from the arrival rates at the AEU submodel, considering the routing and class switching probabilities in the `AEU` model, the effective arrival rates at the `LabRad` and `Spec` sub-submodels are obtained.

In a second (bottom-up) step, submodels are analysed and the analysis results used to compute the class-dependent service rates of the corresponding node at the next

Model	s_{total}	s_{Markov}	t_{gen}	t_{reach}	t_{elim}	n_{iter}	t_{sol}
`Specialists` sub-submodel	14,641	14,641	0.019	0.020	–	340	0.637
`LabRad` sub-submodel	156,849	38,283	0.171	0.212	0.039	948	5.331
`AEU` submodel	10,936,950	1,409,286	0.038	1.644	1.097	859	181.378
high-level model	3,812,364	697,160	0.075	0.566	1.299	1013	101.901

Table 1: Empirical results for the AED model (all times given in seconds)

higher level of the hierarchy. This hierarchical procedure has the advantage that the huge state space of the flattened overall model never has to be generated.

4.3 Empirical results

In this section, we present some empirical results which we obtained when analysing the AED model on an Intel Xeon 3.06 GHz machine with 2 GB of main memory running SUSE Linux version 9.1. For the steady-state analysis of the high-level model and the submodels, the pseudo Gauss-Seidel method was used. This method is a compromise between the methods of Jacobi and Gauss-Seidel and lends itself well to an MTBDD-based (or hybrid symbolic-explicit) implementation, as first described in [12].

Table 1 shows the results. Column s_{total} gives the size of the reachable state space including the vanishing states. Column s_{Markov} gives the size of the state space after the vanishing states have been eliminated. Column t_{gen} gives the time for generating the symbolic representation from the process algebraic description, and columns t_{reach} and t_{elim} show the times for reachability analysis and for the elimination of the vanishing states (where a "–" in column t_{elim} indicates that the model does not contain any immediate transitions). The last two columns provide information about the number of iterations for the numerical method to converge (n_{iter}) and the total solution time (t_{sol}). All timing information is given in seconds. For the experiments, the maximum values of process parameters, which determine the size of the state space, were chosen such that the probability of stations being full (and thus blocked) is kept sufficiently small (up to around 10 percent).

The dominating times are the ones for numerical analysis, but we think that a solution time of around three minutes for a model with more than 10 million reachable states and 1.4 million tangible states is quite acceptable. We point out that the time consumed by the other steps, in particular the time for the new elimination algorithm, are extremely small and almost negligible when compared to the numerical solution time. This is a notable fact, as the major part of the models (except Specialists sub-submodel) consists of immediate actions.

5 Conclusion

This paper described the current state-of-the-art of the SPA tool CASPA. The focus was on the increased expressiveness of the tool's modelling language which is gained through the introduction of weighted immediate actions. A symbolic algorithm for the elimination of vanishing states was described in detail, and empirical results showed its high efficiency. The paper also touched on the improved usability of CASPA for users not familiar with the syntax of stochastic process algebras, by shortly describing the recently added features for graphical model specification. As future work, we

are currently developing an advanced model debugging feature, and we plan to integrate additional numerical analysis algorithms, such as the one described in [13], into CASPA.

A copy of CASPA may be requested by sending an email to one of the authors.

Acknowledgements: We would like to thank the authors of [1] for clarifying some details of their AED model. We would further like to thank Deutsche Forschungsgemeinschaft (DFG) who supported this work under grants SI 710/2 and SI 710/3.

References

1. S.W.M. Au-Yeung, P.G. Harrison, and W.J. Knottenbelt. A Queueing Network Model of Patient Flow in an Accident and Emergency Department. In *Proc. 20th Ann. European Simulation and Modelling Conf.*, pages 60–67, Toulouse, France, 2006.
2. J. Bachmann. Entwurf und Implementierung eines graphischen Modelleditors und einer Benutzerschnittstelle für das Werkzeug CASPA. Master's thesis, Universität der Bundeswehr München, Dept. of Computer Science 4 (in German), 2007.
3. M. Bernardo and R. Gorrieri. A tutorial on EMPA: A theory of concurrent processes with nondeterminism, priorities, probabilities and time. *Theoret. Comp. Science*, 202:1–54, 1998.
4. E. Frank. Erweiterung eines MTBDD-basierten Werkzeugs für die Analyse stocchastischer Transitionssysteme. Int. Report, Univ. of Erlangen, Computer Science 7 (in German), 2000.
5. H. Hermanns. *Interactive Markov Chains : The Quest for Quantified Quality*. Springer LNCS 2428, 2002.
6. H. Hermanns and M. Rettelbach. Syntax, Semantics, Equivalences, and Axioms for MTIPP. *Proc. of PAPM'94, Arbeitsberichte des IMMD*, 27(4):71–88, 1994.
7. J. Hillston. *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.
8. M. Kuntz and M. Siegle. Deriving symbolic representations from stochastic process algebras. In *Proc. Process Algebra and Probabilistic Methods (PAPM-PROBMIV'02)*, pages 188–206. Springer, LNCS 2399, 2002.
9. M. Kuntz and M. Siegle. Symbolic Model Checking of Stochastic Systems: Theory and Implementation. In *Proc. of 13th Int. SPIN Workshop*, pages 89–107. Springer, LNCS 3925, 2006.
10. M. Kuntz, M. Siegle, and E. Werner. Symbolic Performance and Dependability Evaluation with the Tool CASPA. In *Proc. of First European Performance Engineering Workshop (EPEW), FORTE'04 Workshop*, pages 293–307. Springer LNCS 3236, 2004.
11. M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with generalized stochastic Petri nets*. Wiley, 1995.
12. D. Parker. *Implementation of symbolic model checking for probabilistic systems*. PhD thesis, School of Computer Science, Faculty of Science, University of Birmingham, 2002.
13. J. Schuster and M. Siegle. A symbolic multilevel method with sparse submatrix representation for memory-speed tradeoff. In *14. GI/ITG Conf. Measurement, Modelling and Evaluation of Comp. and Communic. Systems (MMB08)*, pages 191–205. VDE Verlag, 2008.
14. M. Siegle. *Behaviour analysis of communication systems: Compositional modelling, compact representation and analysis of performability properties*. Shaker-Verlag, 2002.
15. C. Tofts. Processes with probabilities, priority and time. *Formal Aspects of Computing*, 6(5):536–564, September 1994.
16. E. Werner. Leistungsbewertung mit Multi-terminalen Binären Entscheidungsdiagrammen. Master's thesis, Univ. Erlangen, Computer Science 7 (in German), 2003.