Deriving symbolic representations from stochastic process algebras*

Matthias Kuntz, Markus Siegle

Friedrich-Alexander-Universität Erlangen-Nürnberg, Institut für Informatik
{mskuntz,siegle}@informatik.uni-erlangen.de

Abstract. A new denotational semantics for a variant of the stochastic process algebra TIPP is presented, which maps process terms to Multi-terminal binary decision diagrams. It is shown that the new semantics is Markovian bisimulation equivalent to the standard SOS semantics. The paper also addresses the difficult question of keeping the underlying state space minimal at every construction step.

1 Introduction

Motivation: Binary decision diagrams (BDD) enjoy great success for the compact representation, manipulation and analysis of very large state spaces. They have proved to be an efficient vehicle for alleviating the notorious problem of state space explosion. Recently, stochastic models have been represented symbolically with the help of Multi-terminal BDDs (MTBDD), and it has been shown that in addition to functional analysis, performance analysis and the verification of performability properties can also be carried out on such symbolic representations [7, 14, 19, 21, 25, 27].

We employ stochastic process algebras (SPA) for model specification and wish to generate symbolic representations directly from the high-level model, instead of generating transition systems as an intermediate representation. For this purpose, we develop a denotational semantics which maps a given SPA specification directly to its underlying MTBDD. The semantics proceeds in a compositional fashion, according to the structure of the process term at hand, as it has been observed before [7, 14, 27] that structure exploitation is the key to achieving compact representations. The process algebra which we use is a restricted version of TIPP [15] which guarantees finiteness of the underlying state space. To our knowledge, this is the first complete BDD-based semantics for SPA which completely avoids the construction of transition systems.

The transition system encoded by the MTBDD resulting from our compositional semantics is not necessarily minimal with respect to Markovian bisimulation. Since minimality is a desirable feature, we address the difficult problem of keeping the underlying state space minimal at every step of the MTBDD construction. While it turns out that certain reductions can be performed with the

* This work is supported by the DFG-funded project BDDANA (HE 1408/8) and by the DFG/NWO-funded project VOSS (SI 710/2).

help of some relatively simple heuristic algorithms, we cannot, in general, achieve the maximal reduction without applying a standard bisimulation algorithm.

Related Work: The following is a short summary of related research. We are aware of three approaches to a BDD-based semantics for process algebras, but all of these only cover the functional case, whereas our approach covers stochastic process algebras (note, however, that with minor changes our method can also be applied to purely functional process algebras).

The earliest approach is that of [9], where the authors describe BDD-based procedures for parallel composition, relabelling and restriction of BDDs generated from CCS terms. These procedures assume that the operands are already available as BDDs, and it is shown that under certain conditions the size of the symbolic representation of a given term is proportional to the sum of the sizes of its components. The paper [28] considers the process algebra LOTOS and proposes to exploit the compositional nature of process algebras for building BDD representations of the underlying state space. The overall system specification is decomposed into its basic building blocks. For these, small transition systems are generated and converted into their corresponding BDD representations. The BDD representation of the overall system is then obtained by combining the BDDs for the components in an appropriate way. The approach presented in [8] is the most general, since this method is applicable to a large class of process algebras whose rule system is of GSOS format. A given process algebra term is interpreted as a Boolean formula, i.e. a minterm, and each operator that appears in the term is encoded by one or more Boolean variables, depending on its arity. The minterm is associated with a Boolean vector, and the application of the leading operator causes the changing of some values of the Boolean vector according to specific rules. In terms of transition systems, the original minterm corresponds to the source state and the modified minterm corresponds to the target state of a transition. The length of the encoding is kept constant, although the encoded term can become shorter if its leading operator is of dynamic nature. Since each operator is represented by at least one Boolean variable, the depth of the BDD can become larger than necessary. For the parallel composition operator, the BDD is not constructed from the BDDs representing the two operands, but in a monolithic way, which is not in accordance with the findings of [9, 7, 25–27].

In [24, 3] a denotational semantics is presented which maps terms of a restricted stochastic process algebra to real-valued matrices (which could be represented by MTBDDs). This work is interesting for us since it also aims at constructing representations which are minimal with respect to Markovian bisimulation.

Outline: This paper is organised as follows: In Sect. 2 we define the SPA language and study some properties of Markovian bisimulation. Sect. 3 briefly reviews MTBDDs and provides definitions related to the encoding of finite sets. In Sect. 4 the MTBDD semantics is presented together with an example. The correctness of the new semantics is shown in Sect. 5, the problem of minimality is addressed in Sect. 6, and Sect. 7 concludes the paper.

2 Stochastic process algebras and bisimulation

2.1 Definition of the language

In this paper, a restricted version of the stochastic process algebra TIPP [15] is considered which will be referred to as restricted TIPP or R-TIPP for short.

Definition 1. (Language R-TIPP) *For a set of actions Act (including the internal action τ), let $a \in Act$ and $b \in Act\backslash\{\tau\}$. Let $L \subseteq Act\backslash\{\tau\}$ be a set of visible actions, let $\lambda \in \mathbb{R}^{>0}$ be a rate, and let $X \in Var$ be a process variable. R-TIPP is the language whose terms are given by the following grammar:*

$$\begin{aligned} P &:= P|[L]|P \mid \mathit{hide}\ b\ \mathit{in}\ P \mid Q \\ Q &:= \mathit{stop} \mid X \mid (a,\lambda);Q \mid Q+Q \mid \mathit{rec}X : Q \end{aligned}$$

All occurrences of process variables must be guarded[1].

We have restricted the language in order to keep the state space finite, since we will only be able to represent finite state spaces symbolically. Therefore, recursion is not allowed over static operators.

As underlying semantics we assume the standard SOS semantics given by the semantic rules in Fig. 1[2]. The semantic model is a multi-transition system, i.e. a transition system where the number of instances of a transition is recognised. This multi-transition system is defined as the tuple $(S, Act, \Longrightarrow, s^0)$, where S is the set of derivable process terms, Act is the set of actions, $s^0 \in S$ is the initial process and $\Longrightarrow = \{\!|(s,a,\lambda,t) \,|\, s,t \in S,\ a \in Act,\ \lambda \in \mathbb{R}^{>0}|\!\}$ is a multi-relation ($\{\!|$ and $|\!\}$ denote multiset brackets). The multiplicity of a certain transition is defined as the number of its distinct derivations according to the semantic rules in Fig. 1. For more details see for instance [12, 17]. It is possible to flatten the multi-relation to an ordinary transition relation as follows: Transitions with multiplicity greater than one can be amalgamated into a single transition whose rate is the sum of the individual rates, and such a cumulation (which is actually a special case of Lemma 2 below) preserves the behaviour with respect to Markovian bisimulation[3]. Therefore, from now on, we can safely assume that multiple transitions are already cumulated and that the semantic model is a stochastic labelled transition system (with an ordinary transition relation), defined as follows:

Definition 2. (Stochastic Labelled Transition System (SLTS)) *Let S be a finite set of states, let $s^0 \in S$ be the initial state and let Act be a finite set of action labels. Let $\longrightarrow \subseteq S \times Act \times \mathbb{R}^{>0} \times S$. We call $\mathcal{T} = (S, Act, \longrightarrow, s^0)$ a stochastic labelled transition system. If $(s,a,\lambda,t) \in \longrightarrow$ we write $s \xrightarrow{a,\lambda} t$.*

[1] For a definition of guardedness see e.g. [22]

[2] In the rule for synchronisation, we adopt TIPP's concept of multiplying the rates of the two partner transitions. However, the MTBDD-based semantics described in this paper could also realise other synchronisation schemes, as long as the resulting rate is a function of the two partner rates.

[3] In Sect. 4 we shall see that our MTBDD-based semantics automatically cumulates multiple transitions into a single one.

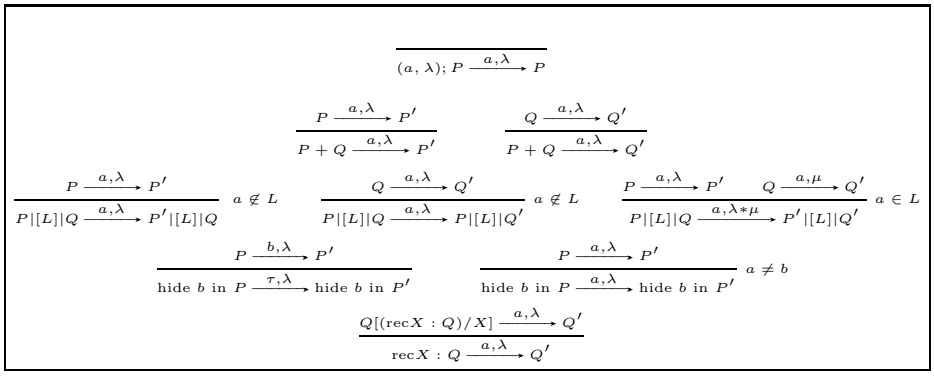

Fig. 1. Semantic rules for the language R-TIPP

2.2 Markovian bisimulation

Bisimulation relations characterise equivalent behaviour at the level of the labelled transition system. Examples are strong and weak bisimulation [22], strong and weak Markovian bisimulation [13] and extended Markovian bisimulation [2] (strong Markovian bisimulation agrees with Hillston's strong equivalence [17]). In this paper, we do not consider weak bisimulations since in R-TIPP all internal transitions have an exponentially distributed delay, which can neither be ignored nor "merged" with another such delay.

Informally, two states are Markovian bisimilar if from both states the same equivalence classes can be reached in one step by the same actions and with the same "cumulative rate". Markovian bisimulation can be seen as a refinement of Markov chain lumpability [6], by distinguishing between different action names. Formally:

Definition 3. (Cumulative rate γ) *The cumulative rate from a state $s \in S$ by action $a \in Act$ to a set of states $C \subseteq S$ is defined by the function $\gamma(s,a,C) = \sum_{\lambda \in E(s,a,C)} \lambda$, where $E(s,a,C) = \{\!|\, \lambda \mid s \xrightarrow{a,\lambda} s' \wedge s' \in C \,|\!\}$. ($\{\!|$ and $|\!\}$ denote multiset brackets.)*

Definition 4. (Markovian bisimulation) *An equivalence relation $\mathcal{B}$ on the set of states S of an SLTS is a (strong) Markovian bisimulation, if $(s_1,s_2) \in \mathcal{B}$ implies that for all equivalence classes C of $\mathcal{B}$ and all actions a it holds that*

$$\gamma(s_1,a,C) = \gamma(s_2,a,C)$$

Two states s_1 and s_2 are Markovian bisimilar (written $s_1 \sim_M s_2$) if they are contained in a Markovian bisimulation.

If $C_1 = \{s_1, s_2, \ldots\}$ and C_2 are equivalence classes of a Markovian bisimulation $\mathcal{B}$ we sometimes write $\gamma(C_1,a,C_2)$ instead of $\gamma(s_i,a,C_2)$, knowing that the

cumulative rate is the same for all $s_i \in C_1$. Given all Markovian bisimulations $\mathcal{B}_1$, $\mathcal{B}_2, \ldots$ on the same SLTS, one is typically interested in the largest (i.e. coarsest) one, namely $\mathcal{B} = \bigcup_i \mathcal{B}_i$.

Bisimulations are useful for reducing the state space of a given transition system, by replacing each class of equivalent states by a single macro state. In that case, the carrier set of the bisimulation relation is the state space S of the transition system to be reduced. From a slightly different perspective, in order to show that two transition systems are equivalent (written $\mathcal{T}_1 \sim_M \mathcal{T}_2$), we may show that their initial states s_1^0 and s_2^0 are bisimulation equivalent. In this case, the union of the two state spaces $S_1 \cup S_2$ can be used as the carrier set of the bisimulation relation. A third viewpoint, which arises when we consider parallel composition, is discussed in the sequel.

Equivalence class structure under parallel composition: We now discuss how equivalence classes of process $P = Q|[L]|R$ can be constructed from the equivalence classes of Q and R (we will use Lemma 1 in Sect. 5).

Lemma 1. *Let $Part^Q = \{C_1^Q, \ldots, C_{m_Q}^Q\}$ ($Part^R = \{C_1^R, \ldots, C_{m_R}^R\}$) be a partition of the state space of process Q (R) which corresponds to the equivalence classes of a Markovian bisimulation. The Cartesian product $C_{i,j}^P = C_i^Q \times C_j^R$ yields a partition $Part^P$ of the state space of process $P = Q|[L]|R$ (with equivalence classes $\{C_{i,j}^P | i = 1, \ldots, m_Q, j = 1, \ldots, m_R\}$) which again corresponds to a Markovian bisimulation.*

Lemma 1 can be proven by showing the equivalence of the cumulative rates [20], details are omitted here (Lemma 1 can also be seen as a consequence of the fact that Markovian bisimulation is a congruence [15, 13]). Note that some equivalence classes of the combined process P may not be reachable due to synchronisation conditions. Note further that Lemma 1 does not assume that $Part^Q$ and $Part^R$ correspond to the largest bisimulation relations, nor does it claim that the equivalence classes $C_{i,j}^P$ are maximal.

Parallel Transitions: Two transitions $s_1 \xrightarrow{a_1,\lambda_1} t_1$ and $s_2 \xrightarrow{a_2,\lambda_2} t_2$ are called parallel if $s_1 = s_2$ and $a_1 = a_2$ and $t_1 = t_2$ (note that, in principle, both $\lambda_1 \neq \lambda_2$ and $\lambda_1 = \lambda_2$ is possible, although the latter case is ruled out if we only consider ordinary transition systems, as opposed to multi-transition systems). Parallel transitions can be created by applying the choice or hiding operators, or by applying the recursion operator in combination with choice. As we will see in Sect. 4, our MTBDD semantics does not represent parallel transitions separately, but cumulates their rates, which is correct by the following lemma:

Lemma 2. *Let $\mathcal{T}$ be an SLTS and let transition system $\mathcal{T}'$ be constructed from $\mathcal{T}$ by cumulating parallel transitions, i.e. by replacing each set of parallel transitions $\{s \xrightarrow{a,\lambda_i} t \mid i = 1, \ldots, n\}$ by a single transition $s \xrightarrow{a,\lambda} t$, where $\lambda = \sum_{i=1}^n \lambda_i$. Then $\mathcal{T} \sim_M \mathcal{T}'$.*

Lemma 2 can be shown by comparing cumulative rates [20].

3 Basis for symbolic representation

3.1 Multi-Terminal Binary Decision Diagrams

MTBDDs [11] (also called ADDs [1]) are an extension of BDDs [5] for the graph-based representation of pseudo-Boolean functions, i.e. functions of type $\mathbb{B}^n \mapsto \mathbb{R}$. An MTBDD is a collapsed binary decision tree whose isomorphic subtrees have been merged and whose don't care vertices are skipped. We consider ordered MTBDDs where on every path from the root to a terminal vertex the variable labelling of the nonterminal vertices obeys a fixed ordering.

In the sequel we assume that the MTBDD variables have the following ordering, denoted by $\prec$. At the first $n_a \geq \lceil \log_2 |Act| \rceil$ levels from the root are the variables a_i encoding the action. On the remaining levels we have $2{*}n_s \geq 2{*}\lceil \log_2 |S| \rceil$ variables encoding the source and target state of a transition. The source state variables (s_i) and the target states variables (t_i) are ordered in an interleaved fashion, which yields the following overall variable ordering[4]:

$$\mathsf{a}_{n_a-1} \prec \ldots \prec \mathsf{a}_0 \prec \mathsf{s}_{n_s-1} \prec \mathsf{t}_{n_s-1} \prec \ldots \prec \mathsf{s}_0 \prec \mathsf{t}_0$$

The function represented by MTBDD M is denoted f_{M}. Given an MTBDD M, we use $\mathsf{M}\big|_{\mathsf{s}=0}$ or $\mathsf{M}\big|_{\mathsf{s}=1}$ to denote its restriction to the case $\mathsf{s} = 0$ or $\mathsf{s} = 1$. Note that the MTBDD resulting from such a restriction does no longer depend on Boolean variable s. Given two MTBDDs M_1 and M_2 and an arithmetic operator $\star \in \{+, -, *, \ldots\}$, we simply write $\mathsf{M} := \mathsf{M}_1 \star \mathsf{M}_2$ to obtain the MTBDD which represents $f_{\mathsf{M}_1} \star f_{\mathsf{M}_2}$. These standard arithmetic (and Boolean) operators can be implemented efficiently on the MTBDD data structure with the help of the so-called APPLY algorithm [11].

3.2 Encodings

Definition 5. (Encoding function) *Let M be an arbitrary finite set. $Enc_M(m)$ denotes the injective encoding function that maps $m \in M$ to its binary encoding (a Boolean vector) of length n, i.e. $Enc_M : M \mapsto \mathbb{B}^n$, $n \geq \lceil \log_2 |M| \rceil$. If M is obvious from the context, the index of the encoding function can be omitted. We write $Enc_M(m) = \vec{m} = (m_{n-1}, \ldots, m_0)$.*

Definition 6. (Encoding sets) *Let the length n of an encoding be given. PC is the set of all possible binary encodings, i.e. $PC := \mathbb{B}^n$. The set of used encodings UC contains those elements of PC that were already used to encode elements of a given set M, i.e. $UC := \{\vec{c} \mid \vec{c} \in PC \wedge \exists m \in M : (Enc_M(m) = \vec{c})\}$. The set of free encodings FC contains those elements of PC that are not in UC, i.e. $FC := PC \setminus UC$.*

[4] This interleaved ordering is the commonly accepted heuristics for obtaining small MTBDD sizes, see for instance [9, 11, 27].

Definition 7. (Extension of a set of encodings by a leading binary digit) *Let C be a set of Boolean vectors of length n. $Ext_0(C)$ is obtained by adding a leading zero to the elements of C, i.e.:*

$$Ext_0(C) = \{\vec{c'} \mid \vec{c'} = 0 \circ \vec{c} \ \wedge \ \vec{c} \in C\}$$

Analogously we obtain $Ext_1(C)$ from C by adding a leading one. The function $Ext(C)$ adds an arbitrary leading digit to the vectors in C, i.e. $Ext(C) = Ext_0(C) \cup Ext_1(C)$.

Definition 8. (Choice of encoding) *An element $\vec{c}$ of a given set of encodings C is chosen with respect to a total ordering relation $\bowtie$ by the function $Ch(C, \bowtie) := \vec{c} \in C$ such that $\forall \vec{c'} \in C : (\vec{c} \bowtie \vec{c'})$.*

Definition 9. (Literal, normal term, minterm) *A literal is a Boolean variable (a) or its complement ($1 - \mathsf{a}$). A normal term is a term in which no variable occurs more than once. A minterm in n variables is a normal multiplication term in n literals.*

Note that since we are working with MTBDDs we use $(1 - \mathsf{a})$ instead of $\overline{\mathsf{a}}$ and multiplication $*$ instead of conjunction $\wedge$.

Definition 10. (Minterm function) *Given n distinct Boolean variables $\mathsf{a}_1, \ldots, \mathsf{a}_n$ and a Boolean vector $(b_1, \ldots, b_n)$ of length n, $MT(\mathsf{a}_1, \ldots, \mathsf{a}_n, b_1 \ldots, b_n)$ denotes the minterm consisting of n literals, i.e.*

$$MT(\mathsf{a}_1, \ldots, \mathsf{a}_n, b_1 \ldots, b_n) \ := \ \mathsf{a}_1^* * \ldots * \mathsf{a}_n^*$$

where $\mathsf{a}_i^* = \mathsf{a}_i$ if $b_i = 1$ and $\mathsf{a}_i^* = (1 - \mathsf{a}_i)$ if $b_i = 0$.

Definition 11. (Transition encoding function) *A transition $x \xrightarrow{a,\lambda} y$ of an SLTS can be encoded using the minterm function:*

$$TR(x \xrightarrow{a,\lambda} y) \ := \ MT(\vec{\mathsf{s}}, Enc_S(x)) * MT(\vec{\mathsf{a}}, Enc_{Act}(a)) * MT(\vec{\mathsf{t}}, Enc_S(y)) * \lambda$$

where $\vec{\mathsf{a}}$ denotes the vector of Boolean variables encoding the action, and $\vec{\mathsf{s}}$ and $\vec{\mathsf{t}}$ denote the vectors of Boolean variables encoding the source and target state of the transition. In the sequel $TR(x \xrightarrow{a,\lambda} y)$ will be written as $TR(x, a, \lambda, y)$.

4 MTBDD semantics for R-TIPP

4.1 General idea

The general idea behind our MTBDD semantics is to encode the transitions of a given process algebraic term P symbolically by an MTBDD. The symbolic representation $[\![P]\!]$ is constructed from the parse tree of P which is processed in a depth first manner, thereby constructing $[\![P]\!]$ inductively from simpler terms. Finally we have the pure MTBDD-based representation of the transitional behaviour of P.

Definition 12. (Symbolic representation of process algebra terms) *The symbolic representation of a process algebra term P is denoted $[\![P]\!]$. It consists of the following parts:*

- *an MTBDD $B(P)$ which encodes the transition relation,*
- *a list of encodings of process variables X that appear in P, denoted $Enc_S(X)$*[5]*,*
- *the encoding of the initial state $Enc_S(s_P^{DS})$.*

The list of action encodings $Enc_{Act}(a)$ is globally valid for all processes and therefore not included in $[\![P]\!]$.

In the following sections we describe how to obtain $[\![P]\!]$ from the symbolic representations of its constituents.

4.2 Process variables and stop process

Verbal description: A (guarded) process variable X specifies a reference state within a surrounding rec$X$ operator[6]. Therefore, process variables are encoded in a similar fashion as states, i.e. their encodings are taken from PC (the set of possible encodings). Within each sequential component[7] (within the scope of the same rec$X$ operator) process variables having the same name get the same encoding. Upon first appearance of a process variable X, the MTBDD associated with it is the 0-MTBDD (the MTBDD consisting of only the terminal vertex 0).

Formal description:

if not first appearance of X within present sequential component **then**
  skip /* do nothing */
else if $FC = \emptyset$ **then** /* need to extend the set of possible encodings */
    $PC := Ext(PC)$; $UC := Ext_0(UC)$; $FC := PC \backslash UC$
  endif
  $Enc_S(X) := Ch(FC, <)$
  $B(X) := 0$
endif

The stop process is a special case of a process variable (a process constant). It has no emanating behaviour, i.e. it remains inactive forever. Therefore, the stop process is associated with the 0-MTBDD.

[5] As explained in Sect. 4.2, process variables correspond to states, therefore we use the encoding function Enc_S for both states and process variables.

[6] Note that R-TIPP does not allow defining equations where the behaviour originating in a process variable is specified in another equation.

[7] A sequential component is a process term which does not include the parallel composition operator.

4.3 Prefix $P = (a, \lambda); Q$

Verbal description: To generate $[\![P]\!] = [\![(a, \lambda); Q]\!]$ from $[\![Q]\!]$ an additional transition has to be inserted into $B(Q)$, leading from the encoding of a new initial state to the encoding of the initial state of Q. A free encoding is chosen and used as the new initial state of the overall process. The path that encodes the new transition is added to the existing MTBDD. (If the set FC of free encodings is empty the set of possible encodings PC has to be extended first, and in the existing MTBDD $B(Q)$ a new source- and target-variable (s_n and t_n) have to be introduced, whose values remain constant.)

Formal description:

if $FC = \emptyset$ **then**
$PC := Ext(PC)$; $UC := Ext_0(UC)$; $FC := PC \backslash UC$
$B(Q) := B(Q) * (1 - \mathsf{s}_n) * (1 - \mathsf{t}_n)$
endif
$Enc_S(s_P^{DS}) := Ch(FC, <)$
$B(P) := B(Q) + TR(s_P^{DS}, a, \lambda, s_Q^{DS})$

4.4 Choice $P = Q + R$

Verbal description: When deriving the symbolic representation $[\![P]\!] = [\![Q + R]\!]$ from $[\![Q]\!]$ and $[\![R]\!]$, a new initial state is introduced for $Q + R$. All transitions emanating from the initial states of the subprocesses Q and R have to be copied, as they may also take place in the initial state of the overall process. (If the set FC of free encodings is empty the set of possible encodings PC has to be extended first, and the existing MTBDDs $B(Q)$ and $B(R)$ have to be adjusted accordingly.)

Formal description:

if $FC = \emptyset$ **then**
$PC := Ext(PC)$; $UC := Ext_0(UC)$; $FC := PC \backslash UC$
$B(Q) := B(Q) * (1 - \mathsf{s}_n) * (1 - \mathsf{t}_n)$
$B(R) := B(R) * (1 - \mathsf{s}_n) * (1 - \mathsf{t}_n)$
endif
$Enc_S(s_P^{DS}) := Ch(FC, <)$
/* copy initial transitions from s_Q^{DS} and s_R^{DS}: */
$B(Q') := B(Q)\Big|_{\vec{\mathsf{s}}=Enc_S(s_Q^{DS})} * MT(\vec{\mathsf{s}}, Enc_S(s_P^{DS}))$
$B(R') := B(R)\Big|_{\vec{\mathsf{s}}=Enc_S(s_R^{DS})} * MT(\vec{\mathsf{s}}, Enc_S(s_P^{DS}))$
$B(P) := B(Q) + B(R) + B(Q') + B(R')$ /* put it all together */

At this point we observe that this procedure will cumulate parallel or multiple transitions correctly: In case Q contains a transition $s_Q^{DS} \xrightarrow{a,\lambda_1} t$ and R contains a transition $s_R^{DS} \xrightarrow{a,\lambda_2} t$ (for any common target state t, and for either $\lambda_1 \neq \lambda_2$ or $\lambda_1 = \lambda_2$), these two transitions will be cumulated, since they are represented in $B(Q')$ and $B(R')$ as parallel transitions emanating from s_P^{DS} and leading to t, and since the MTBDD addition on the last line realises the addition of rates.

As an optimisation of the above procedure, if Q does not possess a cyclic transition sequence of the form $s_0 \xrightarrow{a_1,\lambda_1} s_1 \xrightarrow{a_2,\lambda_2} s_2 \ldots s_{l-1} \xrightarrow{a_l,\lambda_l} s_0$, the initial state of Q can be re-used as initial state of P (and similar for R).

4.5 Parallel composition $P = Q|[L]|R$

Verbal description: For symbolic parallel composition we follow the same basic strategy as described e.g. in [9, 7, 25–27], where it had been found that this scheme ensures that the size of the symbolic representation of the composed process is linear in the size of its components. $[\![P]\!] = [\![Q|[L]|R]\!]$ can be constructed from $[\![Q]\!]$ and $[\![R]\!]$ as follows[8]: The MTBDD which represents the transitions in which both processes participate is constructed by combining those parts of $B(Q)$ and $B(R)$ which correspond to transitions labelled by actions from L (we use L to denote the BDD which encodes the actions in L). The MTBDD which represents the transitions which Q (R) performs independently of R (Q) is constructed by multiplying the part of $B(Q)$ ($B(R)$) with a BDD Id_R (Id_Q) which denotes stability[9] of proces R (Q).

Formal description:

$$Enc_S(s_P^{DS}) := Enc_S(s_Q^{DS}) \circ Enc_S(s_R^{DS})$$
$$B(P) := (B(Q) * \mathsf{L}) * (B(R) * \mathsf{L}) + B(Q) * (1 - \mathsf{L}) * \mathsf{Id}_R + B(R) * (1 - \mathsf{L}) * \mathsf{Id}_Q$$

4.6 Recursion $P = \mathbf{rec}X : Q$

Verbal description: When constructing $[\![P]\!] = [\![\mathrm{rec}X : Q]\!]$ from $[\![Q]\!]$ we can distinguish two cases:

1. X does not appear (unbound) in Q: In this case we simply identify the symbolic representation of $\mathrm{rec}X : Q$ with that of Q.
2. X appears in Q: In this case the process variable X is identified with the encoding of the initial state of Q.

[8] It is assumed that $B(Q)$ depends on the vectors of Boolean variables $\vec{\mathsf{a}}$, $\vec{\mathsf{s}}^Q$, $\vec{\mathsf{t}}^Q$ and $B(R)$ depends on $\vec{\mathsf{a}}$, $\vec{\mathsf{s}}^R$, $\vec{\mathsf{t}}^R$, i.e. their sets of state variables are disjoint.

[9] BDD Id_Q, depending on the vectors of Boolean variables $\vec{\mathsf{s}}^Q$ and $\vec{\mathsf{t}}^Q$, encodes the identity matrix of appropriate size, and has a very compact representation under the interleaved variable ordering. Similar for Id_R.

Formal description (case 2 only):

$$Enc_S(s_P^{DS}) := Enc_S(s_Q^{DS})$$
$$B(P) := B(Q)*(1-MT(\vec{\mathsf{t}}, Enc_S(X)))+B(Q)\Big|_{\vec{\mathsf{t}}=Enc_S(X)} * MT(\vec{\mathsf{t}}, Enc_S(s_Q^{DS}))$$

Note that recursion (in combination with the choice operator appearing within the scope of the recursion) may lead to parallel transitions which are cumulated correctly by the above procedure: In case process Q contains two transitions $s \xrightarrow{b,\lambda_1} s_Q^{DS}$ and $s \xrightarrow{b,\lambda_2} X$ (for any source state s and any action b), the latter of them will be redirected to the target state s_Q^{DS} and the two transitions will be cumulated into the single transition $s \xrightarrow{b,\lambda_1+\lambda_2} s_Q^{DS}$ by the addition of the two MTBDDs.

4.7 Hiding P = hide b in Q

Verbal description: For constructing $[\![P]\!] = [\![\text{hide } b \text{ in } Q]\!]$ from $[\![Q]\!]$, the MTBDD $B(Q)$ is first cofactorised with respect to the encoding of action b. The result is multiplied with the minterm encoding the internal action τ. Finally, the part of the original MTBDD $B(Q)$ that does not correspond to action b is added.

Formal description:

$$Enc_S(s_P^{DS}) := Enc_S(s_Q^{DS})$$
$$B(P) := B(Q)\Big|_{\vec{\mathsf{a}}=Enc_{Act}(b)} * MT(\vec{\mathsf{a}}, Enc_{Act}(\tau))+B(Q)*(1-MT(\vec{\mathsf{a}}, Enc_{Act}(b)))$$

Again, this procedure cumulates parallel transitions correctly. For any pair of states s and t, a transition $s \xrightarrow{b,\lambda_1} t$ (which will be turned into an internal τ-transition) and an existing τ-transition $s \xrightarrow{\tau,\lambda_2} t$ will be cumulated by the addition of the two MTBDDs, leading to a single transition $s \xrightarrow{\tau,\lambda_1+\lambda_2} t$.

4.8 Example

In this section we show how to build the symbolic representation for the process term $P := \text{rec}X : ((a,\lambda);(b,\mu);X \ + \ (c,\gamma);\text{stop})$.

1. First, we generate the parse tree of P which is shown in Fig. 2 (a)
2. We set the action encoding as $Enc_{Act}(\tau) = 00$, $Enc_{Act}(a) = 01$, $Enc_{Act}(b) = 10$ and $Enc_{Act}(c) = 11$.
3. First, we generate the symbolic representation of the left hand side of P.
   (a) $[\![X]\!]$: For process variable X we allocate one state bit, $\mathsf{s_0}$, and encode X as $\mathsf{s_0} = \mathsf{0}$.
   (b) $[\![(b,\mu);X]\!]$: This yields a new state which we encode as $\mathsf{s_0} = \mathsf{1}$. The MTBDD representation is shown in Fig. 2 (b). (In the graphical depiction of an MTBDD, dashed lines indicate zero-edges and solid lines indicate one-edges.)

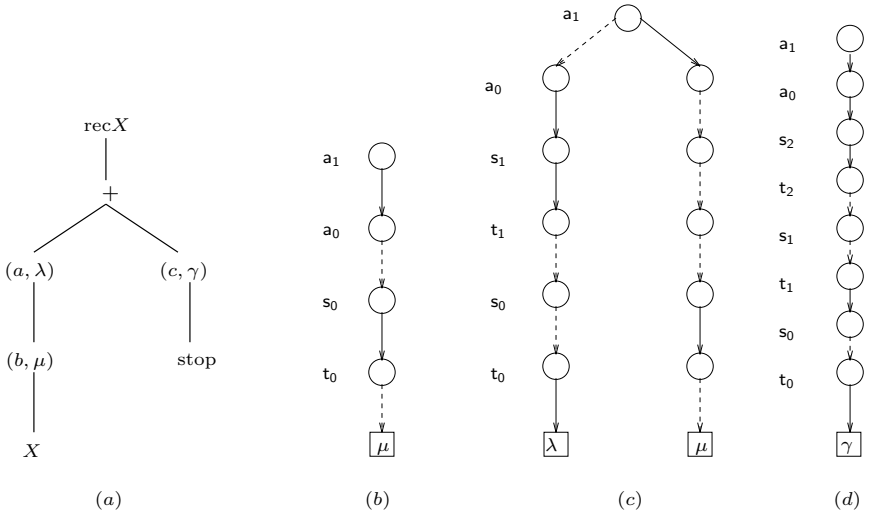

Fig. 2. (a) Parse tree of P, (b) MTBDD representing $(b, \mu); X$, (c)(d) MTBDD representations of the left hand side and right hand side of P

(c) $[\![(a, \lambda); (b, \mu); X]\!]$: This yields a new state. We need to introduce an additional state variable, s_1, and we encode the new state as $\mathsf{s}_1\mathsf{s}_0 = \mathsf{10}$. Now the encoding of the left hand side of P is complete, we store its initial state and the encoding of the process variable X: $(Enc_S(s^{DS}_{(a,\lambda);(b,\mu);X}) = 10, Enc_S(X) = 00)$. The MTBDD representation is depicted in Fig. 2 (c).

4. Next we generate the symbolic representation of the right hand side of P:
   (a) $[\![stop]\!]$: We use the fresh encoding $\mathsf{s}_1\mathsf{s}_0 = \mathsf{11}$ to encode the process stop.
   (b) $[\![(c, \gamma); stop]\!]$: Since the set of free encodings is now empty, we have to add a third variable to encode this new state: $\mathsf{s}_2\mathsf{s}_1\mathsf{s}_0 = \mathsf{100}$. We store the following information: $(Enc_S(s^{DS}_{(c,\gamma);stop}) = 100, Enc_S(\text{stop}) = 011)$. The MTBDD generated for the right hand side of P is depicted in Fig. 2 (d).
5. The third state variable (s_2 and t_2) must also be introduced into the MTBDD corresponding to the left hand side of P (not shown in the figure).
6. The left hand side and the right hand side of P are now composed according to the semantics of the choice operator. As no subprocess has transitions leading back to its initial state, we can choose one of the subprocesses' initial states as the initial state of the overall process. In this case we choose the initial state of the left hand side: $Enc(s^{DS}_{(a,\lambda);(b,\mu);X+(c,\gamma);\text{stop}}) = 010$, yielding the MTBDD representation as shown in Fig. 3 (a).
7. In the last step we add recursion. The encoding of X is now identified with the encoding of the initial state, leading to the final MTBDD depicted in Fig. 3 (b).

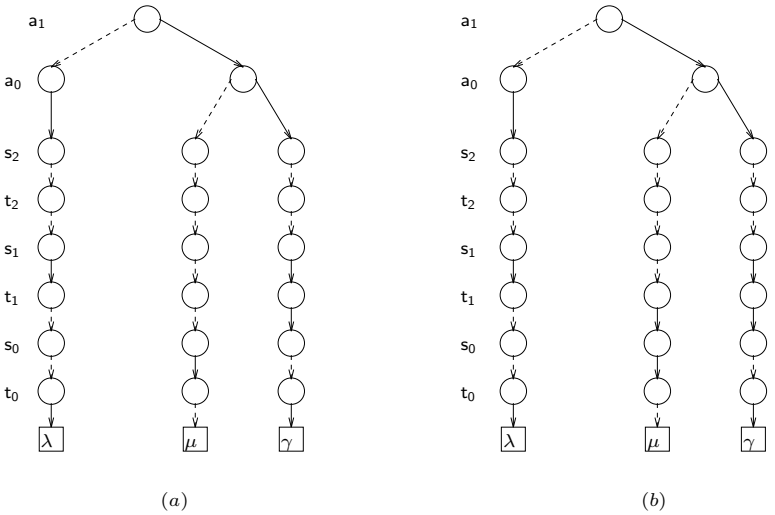

Fig. 3. (a): Left hand side and right hand side joined together using choice, (b) Recursion operator added

5 Correctness of the semantics

In this section we show that our MTBDD semantics is bisimulation equivalent to the standard SOS-semantics. Roughly speaking, this is shown along the following steps:

- From the MTBDD-representation $[\![P]\!]$ of a given term P we derive an SLTS which we denote by $Tr([\![P]\!])$ and whose initial state we denote by s_P^{DS}. The SLTS $Tr([\![P]\!]) = (S_P^{DS}, Act, \longrightarrow, s_P^{DS})$ is obtained from $[\![P]\!]$ by a straightforward algorithm [20] which extracts the encoded transitions one by one from the MTBDD. The details are omitted here.
- Using induction on the term's structure and exploiting the congruence property of Markovian bisimulation we show that for an arbitrary term P the SLTS obtained by applying the SOS-rules, in the following denoted by $SOS(P)$ (with state space S_P^{SOS} and initial state s_P^{SOS}), is bisimulation equivalent to $Tr([\![P]\!])$[10]. (Induction on the term's structure means that for each operator we show that its addition preserves the bisimulation equivalence relation established for the shorter term.)

Theorem 1. *For any process term P from the language R-TIPP it holds that $Tr([\![P]\!]) \sim_M SOS(P)$.*

Proof. The complete proof can be found in [20], here we only present part of it. We distinguish the following cases:

[10] Note that $SOS(P)$ and $Tr([\![P]\!])$ are bisimulation equivalent but not necessarily isomorphic, since $SOS(P)$ may contain parallel transitions, while $Tr([\![P]\!])$ cannot represent these separately.

1. $P = \text{stop}$: This case constitutes the start of the induction. For the stop process, our MTBDD semantics generates a state encoding and the 0-MTBDD. The SLTS $Tr([\![\text{stop}]\!])$ derived from this symbolic representation consists of a single state with no outgoing transitions, which is isomorphic (and therefore Markovian bisimulation equivalent) with the SLTS $SOS(\text{stop})$. The case $P = \text{X}$ is similar.

2. $P = (a, \lambda); Q$: By construction, in $Tr([\![(a, \lambda); Q]\!])$ there is only a single transition emanating from the initial state, namely $s_P^{DS} \xrightarrow{a,\lambda} s_Q^{DS}$. Similarly, in $SOS((a, \lambda); Q)$ there is only a single transition emanating from the initial state, namely $s_P^{SOS} \xrightarrow{a,\lambda} s_Q^{SOS}$. Since by the induction hypothesis we have $s_Q^{DS} \sim_M s_Q^{SOS}$ it follows that $s_P^{DS} \sim_M s_P^{SOS}$ and thus $Tr([\![P]\!]) \sim_M SOS(P)$.

3. $P = Q + R$: We need to show that $s_P^{DS} \sim_M s_P^{SOS}$ which means that we have to show that for all actions $a \in Act$ and for all equivalence classes $C \subseteq S_P^{DS} \cup S_P^{SOS}$ we have $\gamma(s_P^{DS}, a, C) = \gamma(s_P^{SOS}, a, C)$. (Note that the bisimulation relation $\mathcal{B}$ is now defined on the union of the state spaces of $Tr([\![Q + R]\!])$ and $SOS(Q + R)$, i.e. each equivalence class C is a subset of that union.)

$$\begin{aligned}
\gamma(s_P^{DS}, a, C) &\overset{\text{(by construction)}}{=} \gamma(s_P^{DS}, a^Q, C) + \gamma(s_P^{DS}, a^R, C) \\
&\overset{\text{(by construction)}}{=} \gamma(s_Q^{DS}, a^Q, C) + \gamma(s_R^{DS}, a^R, C) \\
&\overset{\text{(by ind. hypothesis)}}{=} \gamma(s_Q^{SOS}, a^Q, C) + \gamma(s_R^{SOS}, a^R, C) \\
&\overset{\text{(by SOS semantics)}}{=} \gamma(s_P^{SOS}, a, C)
\end{aligned}$$

   Here $\gamma(s_P^{DS}, a^Q, C)$ denotes the cumulation of those a-transitions which are due to an a-transition in subprocess Q (and similarly for R). We have used the induction hypothesis, namely that $s_Q^{DS} \sim_M s_Q^{SOS}$ and $s_R^{DS} \sim_M s_R^{SOS}$.

4. $P = Q|[L]|R$: We prove $s_P^{DS} \sim_M s_P^{SOS}$ by showing that for every action a and for every equivalence class $C_{k,l}^P$ we have $\gamma(s_P^{DS}, a, C_{k,l}^P) = \gamma(s_P^{SOS}, a, C_{k,l}^P)$ (note that the equivalence relation is now defined on the union of the state spaces $S = S_P^{DS} \cup S_P^{SOS}$ where $S_P^{DS} = S_Q^{DS} \times S_R^{DS}$ and $S_P^{SOS} = S_Q^{SOS} \times S_R^{SOS}$). We need to distinguish two cases:
   (a) $a \notin L$, i.e. a is a non-synchronising action.

$$\begin{aligned}
\gamma(s_P^{DS}, a, C_{k,l}^P) &\overset{\text{(by construction)}}{=} \gamma(s_P^{DS}, a^Q, C_{k,l}^P) + \gamma(s_P^{DS}, a^R, C_{k,l}^P) \\
&\overset{\text{(by constr. and Lemma 1)}}{=} \gamma(s_Q^{DS}, a^Q, C_k^Q) + \gamma(s_R^{DS}, a^R, C_l^R) \\
&\overset{\text{(by ind. hypothesis)}}{=} \gamma(s_Q^{SOS}, a^Q, C_k^Q) + \gamma(s_R^{SOS}, a^R, C_l^R) \\
&\overset{\text{(by SOS sem. and Lemma 1)}}{=} \gamma(s_P^{SOS}, a^Q, C_{k,l}^P) + \gamma(s_P^{SOS}, a^R, C_{k,l}^P)
\end{aligned}$$

$$\stackrel{\text{(by SOS semantics)}}{=} \gamma(s_P^{SOS}, a, C_{k,l}^P)$$

Here $\gamma(., a^Q, .)$ ($\gamma(., a^R, .)$) denotes those transitions which are due to an a-transition of process Q (R). Note that (assuming that $s_P^{DS} \in C_{i,j}^P$), in this non-synchronising case, target class $C_{k,l}^P$ is either $C_{i,l}^P$ or $C_{k,j}^P$ since only one partner makes a move, and thus one of the summed cumulative rates in the above sequence of equations is always equal to zero.

(b) $a \in L$, i.e. a is a synchronising action. Details omitted for lack of space.

5.-6. The proof for recursion and hiding proceeds essentially along the same line, by establishing the equivalence of the cumulative rates. Details are omitted.

6 Towards minimal semantics

We have shown that our MTBDD semantics is correct, but so far we have not made any considerations on its minimality. It would, of course, be desirable that the MTBDD constructed by our denotational semantics encodes an SLTS which is minimal with respect to Markovian bisimulation, where minimality means that every class of bisimilar states is represented by a single macro state. Trivially, this goal could be achieved by performing bisimulation minimisation [23, 10, 18] after every construction step, thereby ensuring that all intermediate representations are minimal. BDD-based bisimulation algorithms are available [4, 16], they follow the usual iterative refinement scheme, but such a strategy is impracticable since the overhead for running the bisimulation algorithm would be prohibitive.

Ideally, we wish to perform bisimulation on-the-fly, keeping the encoded state space minimal at every step of the construction, by exploiting information about the operator at hand and the structure of the operand processes. For that purpose we investigate a set of heuristic algorithms. As a simple example, we now briefly discuss such an algorithm for the case of recursion. Fig. 4 shows an algorithm which merges known equivalence classes of process Q in order to obtain equivalence classes for $P = \text{rec}X : Q$. Starting from the newly formed class C'_{ini} which contains both X and the initial state, the algorithm looks for pairs of predecessor classes (denoted $Pred(C'_{ini})$) which can be merged. If two classes are merged, the search also considers the further predecessor classes in a chained fashion, and this aim-driven procedure makes the algorithm quite efficient. In the example shown in Fig. 5, the predecessor of X and the predecessor of the initial state are merged first, and in the subsequent step the two dark states are merged. This is a case where our heuristic algorithm finds the largest Markovian bisimulation, leading to a minimal state space. However, there exist situations like the one shown in Fig. 6 where the algorithm of Fig. 4 does not find the coarsest partition, since it is not possible to merge the two dark states without merging the three lightly shaded states at the same time. Altogether, one can say that the merging of two classes at a time is not sufficient, as our algorithm is correct but not complete.

(1) $Part := Part^Q$
(2) $C'_{ini} := C_{ini} \cup C_X$ /* the initial class and the class containing state X are merged */
(3) $Part := Part \backslash \{C_{ini}, C_X\} \cup \{C'_{ini}\}$ /* the partition is updated */
(4) $Mergers := \{C'_{ini}\}$
(5) **while** $Mergers \neq \emptyset$ **do**
(6) *choose* $C_{mrg} \in Mergers$
(7) **forall** $C_i, C_j \in Pred(C_{mrg})$ **do** /* consider pairs of predecessor classes of C_{mrg} */
(8) **if** $\forall a : \gamma(C_i, a, C_i \cup C_j) = \gamma(C_j, a, C_i \cup C_j)$ **then** /* compare mutual rates */
(9) **if** $\forall C_k : \forall a : \gamma(C_i, a, C_k) = \gamma(C_j, a, C_k)$ **then** /* compare rates to third party */
(10) $C'_i := C_i \cup C_j$ /* two classes are merged */
(11) $Part := Part \backslash \{C_i, C_j\} \cup \{C'_i\}$ /* the partition is updated */
(12) $Mergers := Mergers \cup \{C'_i\}$ /* a new merger is added */
(13) **endif**
(14) **endif**
(15) **endfor**
(16) $Mergers := Mergers \setminus \{C_{mrg}\}$ /* the processed merger is removed */
(17) **endwhile**
(18) **return** $Part$

Fig. 4. Determining equivalence classes of rec$X : Q$ from the classes of Q

For prefix, finding the optimal partition is trivial: If Q contains a state s' with a single outgoing transition $s' \xrightarrow{a,\lambda} s_Q^{DS}$ then s' is equivalent to the new initial state of $P = (a, \lambda); Q$. For choice, as already observed in [24], the key to minimality lies in the ability to detect common behaviour within the operands Q and R. This can be achieved by identifying and comparing the strongly connected components (SCC) of Q and R. SCCs can be determined symbolically in an efficient way [29][11]. For parallel composition, the resulting SLTS is not minimal if the two partners contain identical behaviour which leads to symmetries in the state space (but symmetry is not a necessary precondition for non-minimality). [24] describes state space reduction for replicated processes. Although this can yield a large reduction of the state space, it is shown in [3] that the resulting SLTS is not necessarily minimal. In fact, it is minimal only if all states of the replicated process are "relatively prime", which condition is difficult to verify in practice[12]. For hiding, the situation is essentially the same as for recursion: We have an algorithm which calculates equivalence classes of (hide b in Q) from the equivalence classes of Q, but again the resulting partition is not necessarily optimal.

[11] In addition, since only reachable behaviour should be represented, in case the initial state s_Q^{DS} (s_R^{DS}) is unreachable after the application of the choice operator, transitions emanating from this state can be deleted.

[12] Furthermore, since some states of the combined process may be unreachable due to synchronisation conditions, (symbolic) reachability analysis may be necessary in order to determine the set of reachable states.

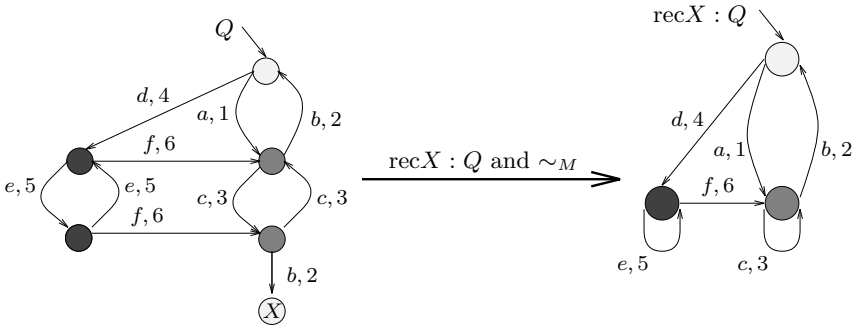

Fig. 5. Example where the algorithm of Fig. 4 finds the largest bisimulation

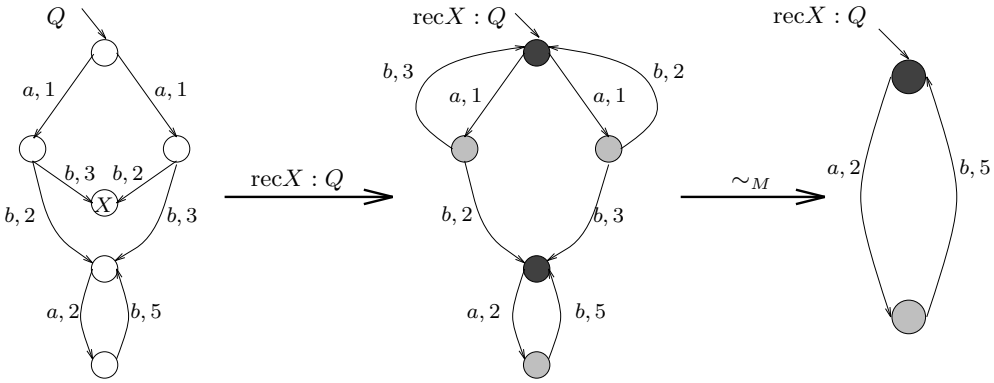

Fig. 6. Example (after [3]) that demonstrates non-optimality of the algorithm in Fig. 4

7 Conclusion and future work

Conclusions: We presented a straight-forward approach to deriving a symbolic representation of the transition system underlying a given SPA term. The major contribution of this work is to provide a fully MTBDD-based semantics for a stochastic process algebra, in order to avoid the state space explosion problem by exploiting compositionality. Furthermore we discussed some heuristic algorithms that help to reduce the encoded state space at each stage of its construction.

Future work: Next, we plan to implement our denotational semantics, in order to compare the size of the resulting MTBDDs to that of existing MTBDD generation methods. In the implementation, some optimisations are possible (e.g. reuse of the encodings of states that became unreachable). It is planned that our implentation works with an extended version of R-TIPP, for instance allowing parallel composition not only at the top level (as long as it is not in the scope of a recursion operator), and adding an immediate prefix. Finally, we intend to

further investigate the questions addressed in Sect. 6, e.g.: Is it possible to classify the situations in which our heuristic algorithms lead to optimal/non-optimal results? What is the benefit of using our algorithms compared to standard bisimulation algorithms? Is it even possible to extend our algorithms in such a way that they guarantee minimality?

Acknowledgements: The authors would like to thank the anonymous referees whose comments helped to fix some technical flaws and improve the presentation of the paper.

References

1. 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.
2. M. Bernardo and R. Gorrieri. A Tutorial on EMPA: A Theory of Concurrent Processes with Nondeterminism, Priorities, Probabilities and Time. *Theoretical Computer Science*, 202:1–54, 1998.
3. H. Bohnenkamp. Kompositionelle Semantiken stochastischer Prozeßalgebren zur Erzeugung reduzierter Transitionssysteme. Master's thesis, University of Erlangen–Nürnberg, Informatik 7, 1995 (in German).
4. A. Bouali and R. de Simone. Symbolic Bisimulation Minimisation. In *Computer Aided Verification*, pages 96–108, 1992. LNCS 663.
5. R.E. Bryant. Graph-based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.
6. P. Buchholz. Exact and Ordinary Lumpability in Finite Markov Chains. *Journal of Applied Probability*, (31):59–75, 1994.
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 S. Graf and M. Schwartzbach, editors, *TACAS'2000, LNCS 1785*, pages 395–410, Berlin, 2000.
8. A. Dsouza and B. Bloom. Generating BDD models for process algebra terms. In *Computer Aided Verification*, pages 16–30, 1995. LNCS 939.
9. R. Enders, T. Filkorn, and D. Taubner. Generating BDDs for symbolic model checking in CCS. *Distributed Computing*, (6):155–164, 1993.
10. J.C. Fernandez. An Implementation of an Efficient Algorithm for Bisimulation Equivalence. *Science of Computer Programming*, 13:219–236, 1989.
11. 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.
12. N. Götz. Stochastische Prozeßalgebren – Integration von funktionalem Entwurf und Leistungsbewertung Verteilter Systeme. Ph.D. thesis, Universität Erlangen–Nürnberg, 1994 (in German).
13. H. Hermanns, U. Herzog, and V. Mertsiotakis. Stochastic Process Algebras — Between LOTOS and Markov Chains. *Computer Networks and ISDN (CNIS)*, 30(9-10):901–924, 1998.
14. H. Hermanns, J. Meyer-Kayser, and M. Siegle. Multi Terminal Binary Decision Diagrams to Represent and Analyse Continuous Time Markov Chains. In B. Plateau,

W.J. Stewart, and M. Silva, editors, *3rd Int. Workshop on the Numerical Solution of Markov Chains*, pages 188–207. Prensas Universitarias de Zaragoza, 1999.
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.
16. H. Hermanns and M. Siegle. Bisimulation Algorithms for Stochastic Process Algebras and their BDD-based Implementation. In J.-P. Katoen, editor, *ARTS'99, 5th Int. AMAST Workshop on Real-Time and Probabilistic Systems*, pages 144–264. Springer, LNCS 1601, 1999.
17. J. Hillston. *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.
18. P. Kanellakis and S. Smolka. CCS Expressions, Finite State Processes, and Three Problems of Equivalence. *Information and Computation*, 86:43–68, 1990.
19. 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.
20. M. Kuntz and M. Siegle. Deriving symbolic representations from stochastic process algebras. Tech. Rep. Informatik 7 03/02, Universität Erlangen-Nürnberg, 2002.
21. M. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic Symbolic Model Checker. In P. Kemper, editor, *MMB-PNPM-PAPM-PROBMIV Tool Proceedings*, pages 7–12. Univ. Dortmund, Informatik IV, Bericht 760/2001, 2001.
22. R. Milner. *Communication and Concurrency*. Prentice Hall, London, 1989.
23. R. Paige and R. Tarjan. Three Partition Refinement Algorithms. *SIAM Journal of Computing*, 16(6):973–989, 1987.
24. M. Rettelbach and M. Siegle. Compositional Minimal Semantics for the Stochastic Process Algebra TIPP. In *Proc. of PAPM'94*, pages 89–106. Arbeitsberichte des IMMD 27 (4), Universität Erlangen-Nürnberg, 1994.
25. M. Siegle. Compact representation of large performability models based on extended BDDs. In *Fourth Int. Workshop on Performability Modeling of Computer and Communication Systems (PMCCS4)*, pages 77–80, Williamsburg, Sept. 1998.
26. M. Siegle. Compositional Representation and Reduction of Stochastic Labelled Transition Systems based on Decision Node BDDs. In D. Baum, N. Müller, and R. Rödler, editors, *MMB'99*, pages 173–185, Trier, September 1999. VDE Verlag.
27. M. Siegle. Advances in model representation. In L. de Alfaro and S. Gilmore, editors, *Process Algebra and Probabilistic Methods, Joint Int. Workshop PAPM-PROBMIV 2001*, pages 1–22. Springer, LNCS 2165, September 2001.
28. R. Sisto. A method to build symbolic representations of LOTOS specifications. In *Protocol Specification, Testing and Verification*, pages 323–338, 1995.
29. A. Xie and P.A. Beerel. Implicit Enumeration of Strongly Connected Components. In *ICCAD'99*, pages 37–40. IEEE, 1999.