Distrib Comput (1993) 6:155–164

DISTRIBUTED COMPUTING

Generating BDDs for symbolic model checking in CCS

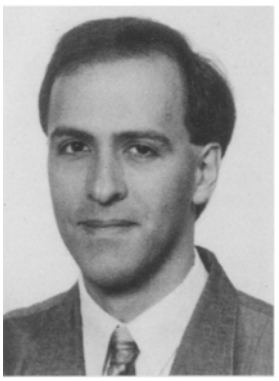
Reinhard Enders [1], **Thomas Filkorn** [1], **and Dirk Taubner** [2]

[1] Siemens AG, Corporate Research and Development (ZFE BT SE 1), Otto-Hahn-Ring 6, W-8000 München 83, Federal Republic of Germany
[2] sd & m GmbH, Thomas-Dehler-Strasse 27, W-8000 München 83, Federal Republic of Germany

Received September 1991 / Accepted July 1992

Reinhard Enders graduated from the Technical University in Munich with a Diploma in mathematics and computer science in 1978. From 1977 to 1984 he was employed by Siemens, working in computer linguistics and expert systems. From 1984 to 1988 he worked at ECRC on Prolog extensions. In Autmn 1988 he joined Siemens and is developping the constraint extension of a new Prolog product.

Thomas Filkorn received the computer science degree and the Ph.D. degree, both from the Technical University of Munich. Since 1992 he works at Siemens' Corporate Research and Development on symbolic algorithms and methods for the verification of finite state systems.



Dirk Taubner received his Ph.D. in informatics at the Technical University of Munich in 1988. He investigated which sublanguages of process algebra could be represented finitely by automata and Petri nets.

From 1989 through 91 he worked at Siemens' Corporate Research and Development where he led a project on computer-aided verification of parallel processes. This paper presents part of the work of that project. Currently he works on commercial software engineering for a software consulting company.

Correspondence to: T. Filkorn

Summary. Finite transition systems can easily be represented by binary decision diagrams (BDDs) through the characteristic function of the transition relation. Burch et al. have shown how model checking of a powerful version of the μ-calculus can be performed on such BDDs. In this paper we show how a BDD can be generated from elementary finite transition systems given as BDDs by applying the CCS operations of parallel composition, restriction, and relabelling. The resulting BDDs only grow linearly in the number of parallel components. This way bisimilarity checking can be performed for processes out of the reach of conventional process algebra tools.

Key words: Binary decision diagram (BDD) – Process algebra – CCS – Transition system – (Symbolic) model checking – Bisimilarity

1 Introduction

A binary decision diagram as described by Bryant [1] is a normal form representation of boolean functions $f: \mathbb{B}^n \to \mathbb{B}$, where $\mathbb{B} = \{0, 1\}$. It is often much smaller than other normal form representations. Moreover boolean operations can be applied efficiently.

Transition systems have states and transitions leading from one state via some action to another state. They underly many semantics for concurrent systems, e.g. bisimulation equivalence for CCS [11]. They also underly semantics of modal logics, which allow to reason about concurrent systems.

Burch et al. [2] have shown how model checking of a powerful version of the μ-calculus including existential quantification and λ-abstraction can be performed for a boolean domain using BDDs for internal representation. They call the approach *symbolic model checking*. They also indicate how other finite domains can be treated through binary encoding.

In particular for a finite transition system the set of states S and the set Act of actions can be encoded as subsets of $\mathbb{B}^{\lceil \log_2 |S| \rceil}$ and $\mathbb{B}^{\lceil \log_2 |Act| \rceil}$ respectively. The transition relation $D \subseteq S \times Act \times S$ can be encoded as a function $\chi: \mathbb{B}^{2 \cdot \lceil \log_2 |S| \rceil + \lceil \log_2 |Act| \rceil} \to \mathbb{B}$ which in turn can be represented as a BDD. Burch et al. show how weak and strong bisimilarity can be expressed as formulas in their μ-calculus and hence can be checked for transition systems given as BDDs.

However they do not indicate how the BDDs are generated. If they were generated from the list of state-action-state triples of the transition relation (which is straightforward) the approach would immediately suffer the well-known explosion problem, namely that a system of N parallel components, with n states each, may have n^N transitions.

The question arises whether the BDD which represents the transition system of the compound system could be generated more efficiently directly from the N components given as BDDs without enumerating all resulting transitions. This question is the topic of this paper. It is answered in the following way: We give a certain encoding of transition systems as BDDs. We show how the CCS [11] operators of parallel composition, restriction, and relabelling can be applied to BDDs. We prove that for a fixed set of actions the BDD for representing N parallel communicating processes grows only linearly in N. Worst case boundaries for the size of BDDs are rare in literature, hence this bound is interesting. However it has to be pointed out that it concerns only the resulting BDD which represents the system of N parallel component processes. It does not concern intermediate BDDs which are used during the generation or during the model checking. Nevertheless a benchmark example shows that bisimilarity checking can be performed for systems out of the reach of conventional tools such as [3, 4, 7, 10, 6].

Note that the μ-calculus of [2] is more powerful than that of e.g. [5, 12]. Hence all formulas checked with the latter approaches can be checked in the framework of this paper. We expect to be able to handle larger transition systems.

On the other hand it should be noted that BDDs are efficient only in a heuristic sense, the worst case may still be catastrophic. For circuit verification experience has shown that indeed BDDs can serve as an efficient representation in many practical cases. This paper gives evidence that this is also the case for verification of parallel processes. Possible applications are communication protocols, operating system tasks, and distributed control systems, see e.g. [9].

Before we start with the technical part of this paper let us explain with an example the basic idea, i.e. how to represent relations by BDDs and why this is promising for the parallel composition of transition systems. Consider the relation D of Fig. 1 where $D \subseteq S \times S$ for $S = \{0, 1, 2, 3\}$. We omit the actions for the moment. An obvious boolean encoding of S is $0 \mapsto 00$, $1 \mapsto 01$, $2 \mapsto 10$, $3 \mapsto 11$. The relation D may then be represented by a function: $\chi: \mathbb{B}^4 \to \mathbb{B}$, such that $\chi(r_1, r_2, s_1, s_2) = 1$ if and only if the state encoded by $r_1 r_2$ has an edge to the state encoded by $s_1 s_2$. This boolean function χ can in turn be represented as a BDD.

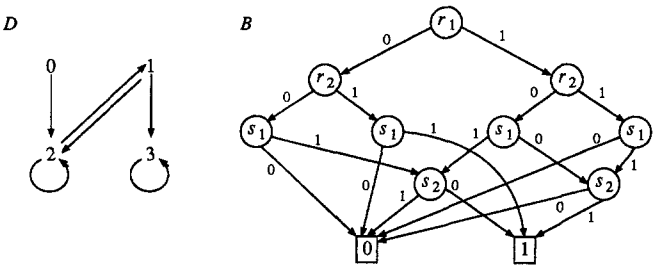


Fig. 1. A relation (left) and its BDD representation (right)

For a given set of boolean variables which are totally ordered by $\prec$ a BDD is a rooted directed acyclic graph, each node is either terminal, then it is labelled by a truth-value and has no successor, or it is nonterminal, then it is labelled by some variable and has two successors which are terminal or labelled by a larger ($\prec$) boolean variable. One successor corresponds to 0, the other to 1.

Taking the ordering $r_1 \prec r_2 \prec s_1 \prec s_2$ the function χ is represented by the BDD B given in Fig. 1.

Now assume we want to represent the relation [1]

$$\{\langle\langle r, r'\rangle, \langle s, s'\rangle\rangle \in S \times S \mid \langle r, s\rangle \in D \wedge \langle r', s'\rangle \in D\} \qquad (*)$$

as a BDD C. We can take a copy B' of B with fresh variables r_1', r_2', s_1', s_2' and calculate C as $B \wedge B'$ where the conjunction of BDDs is as described in [1].

However the size of C depends on the chosen variable ordering. If $r_1 \prec r_2 \prec s_1 \prec s_2 \prec r_1' \prec r_2' \prec s_1' \prec s_2'$ is chosen it has 20 nodes, approximately $2 \cdot |B|$, where $|B|$ denotes the number of nodes in B. It is formed simply by attaching B' to the 1-exit of B. If, on the other hand, $r_1 \prec r_2 \prec r_1' \prec r_2' \prec s_1 \prec s_2 \prec s_1' \prec s_2'$ is chosen it has 47 nodes (approximately $|B|^2/2$).

It is not hard to see that for an arbitrary binary relation D which is encoded in the style of Fig. 1 the number of nodes of the BDD representing the relation (*) is bounded by $O(|B|)$ and $O(|B|^2)$ respectively depending on the choosen ordering.

This ends our introductory example. We wanted to indicate that if the ordering of variables is chosen carefully the BDD only grows additively while the number of elements in the represented relation grows multiplicatively. However be aware that the encoding as presented in Sect. 4 uses a different ordering in order to also serve the asynchronous case well.

2 Operators on transition systems

Our language for composing parallel processes is taken from the process algebra CCS [11]. Let Act be the finite set of actions which contains the invisible action τ and

[1] This relation resembles the synchronization part of parallel composition, cf. the next section. In general, of course, one wants to combine two relations and not twice the same

all other actions in two copies, a and $\bar{a}$, which are complementary, i.e. $\bar{\bar{a}}=a$.

A transition system $T=\langle S, D, z\rangle$ has states S, initial state $z\in S$, and transitions $D\subseteq S\times Act\times S$ leading from one state with some action to another state. Throughout we assume S to be finite.

Next we give operators on transition systems which correspond to CCS parallel composition, restriction, and relabelling. In [13] it is shown that they are correct (consistent) in the following sense. Given two closed CCS terms P_1 and P_2 and let T_1, T_2 be their respective transition systems according to the transitional semantics [11], then $T_1|T_2$ as defined below is strongly bisimilar to the transition system of $P_1|P_2$ according to the transitional semantics. Similar results hold for restriction and relabelling.

The *CCS parallel composition* for given $T_1=\langle S_1, D_1, z_1\rangle$ and $T_2=\langle S_2, D_2, z_2\rangle$ is defined as $T_1|T_2=\langle S_1\times S_2, D, \langle z_1, z_2\rangle\rangle$ where

$$D=\{\langle\langle r_1, r_2\rangle, \alpha, \langle s_1, s_2\rangle\rangle \mid \langle r_1, \alpha, s_1\rangle\in D_1\wedge r_2=s_2 \vee \langle r_2, \alpha, s_2\rangle\in D_2\wedge r_1=s_1 \vee \alpha=\tau\wedge\exists a\neq\tau\colon \langle r_1, a, s_1\rangle\in D_1\wedge\langle r_2, \bar{a}, s_2\rangle\in D_2\}.$$

The condition $\langle r_1, \alpha, s_1\rangle\in D_1\wedge r_2=s_2$ and its symmetric version represent an *asynchronous* move in which only one component proceeds. On the other hand the condition $\langle r_1, a, s_1\rangle\in D_1\wedge\langle r_2, \bar{a}, s_2\rangle\in D_2$ represents a *synchronous* move. The latter in a simplified version has already been discussed in the introduction.

Let $T=\langle S, D, z\rangle$ be given. *Restriction* of a subset $A\subseteq Act-\{\tau\}$ is defined as

$$T\backslash A=\langle S, \{\langle r, \alpha, s\rangle\in D \mid \alpha\notin A\wedge\bar{\alpha}\notin A\}, z\rangle.$$

Relabelling of a visible action a $(a\neq\tau)$ into the action b is defined as $T[b/a]=\langle S, D', z\rangle$ where

$$D'=\{\langle r, \beta, s\rangle \mid \exists\alpha\colon \langle r, \alpha, s\rangle\in D\wedge(\beta=\alpha\notin\{a, \bar{a}\}\not\in\vee (\beta=b\wedge\alpha=a)\vee(\beta=\bar{b}\wedge\alpha=\bar{a}))\}.$$

3 Symbolic model checking using BDDs

Burch, Clarke, McMillan, Dill, and Hwang [2] show how model checking can be performed using BDDs. They call their approach 'symbolic' as it uses BDDs to represent relations internally with the intention that this is more efficient than the corresponding lists of tuples.

The version of the μ-calculus used in [2] includes individual variables, n-ary relational variables $(n>0)$, existential quantification, abstraction (λ-binding) of individual variables, and fixpoint binding of relational variables. Syntax and semantics are summarized in Figs. 2 and 3. The propositional μ-calculus as used e.g. in [5, 12] can easily be embedded by defining a suitable relational term for the modal operator $\langle a\rangle$. It is based on relational variables representing the transition relation of the structure which underlies the interpretation of the propositional μ-calculus, see Fig. 4.

Assume a finite set of individual variables (denoted z, $z_1, \ldots$ in the following syntactic clauses) and a set of n-ary relational variables for $n>0$ (denoted Z^n, $Z_1^n, \ldots$). The *syntax* has syntactic categories F for formulas and P^n or n-ary relational terms.

$$F::=\mathrm{True}\mid z_1=z_2\mid\neg F\mid F\vee F\mid\exists z[F]\mid P^n(z_1, \ldots, z_n)$$

where in the last clause all individual variables $z_1, \ldots, z_n$ are not free in P^n.

$$P^n::=Z^n\mid\mu Z^n[P^n]\mid\lambda z_1, \ldots, z_n[F]$$

where in the second clause P^n is syntactically monotone in Z^n and where in the last clause the individual variables are distinct.
Abbreviations are as usual, i.e. $\mathrm{False}\equiv\neg\mathrm{True}$, $F_1\Rightarrow F_2\equiv\neg F_1\vee F_2$, $F_1\wedge F_2\equiv\neg(\neg F_1\vee\neg F_2)$, $\forall z[F]\equiv\neg\exists z[\neg F]$, and $\nu Z^n[P^n]\equiv\neg\mu Z^n[\neg P^n\langle Z^n\leftarrow\neg Z^n\rangle]$.

Fig. 2. Syntax of μ-calculus used in [2]

Assume a domain $\mathbb{D}$, a relational variable interpretation I_P which maps every n-ary relational variable to a subset of $\mathbb{D}^n$, and an individual variable interpretation I_D which maps every individual variable to an element of $\mathbb{D}$.
The *semantics* maps a formula F to a truthvalue, i.e.

$$[\![F]\!](I_P)(I_D)\in\mathbb{B}$$

and it maps an n-ary relational term P^n to a subset of $\mathbb{D}^n$, i.e.

$$[\![P^n]\!](I_P)(I_D)\subseteq\mathbb{D}^n$$

Semantics of formulas

$$\begin{aligned}
&[\![\mathrm{True}]\!](I_P)(I_D) &&:=1\\
&[\![z_1=z_2]\!](I_P)(I_D) &&:=I_D(z_1)=I_D(z_2)\\
&[\![\neg F]\!](I_P)(I_D) &&:=\neg[\![F]\!](I_P)(I_D)\\
&[\![F_1\vee F_2]\!](I_P)(I_D) &&:=[\![F_1]\!](I_P)(I_D)\vee[\![F_2]\!](I_P)(I_D)\\
&[\![\exists z[F]]\!](I_P)(I_D) &&:=\exists d\in\mathbb{D}\colon[\![F]\!](I_P)(I_D\langle z\leftarrow d\rangle)\\
&[\![P^n(z_1, \ldots, z_n)]\!](I_P)(I_D) &&:=\langle I_D(z_1), \ldots, I_D(z_n)\rangle\in[\![P^n]\!](I_P)(I_D)
\end{aligned}$$

Semantics of n-ary relational terms

$$\begin{aligned}
&[\![Z^n]\!](I_P)(I_D) &&:=I_P(Z^n)\\
&[\![\lambda z_1, \ldots, z_n[F]]\!](I_P)(I_D) &&:=\{\langle d_1, \ldots, d_n\rangle \mid [\![F]\!](I_P)(I_D\cdot\langle z_1\leftarrow d_1, \ldots, z_n\leftarrow d_n\rangle)\}\\
&[\![\mu Z^n[P^n]]\!](I_P)(I_D) &&:=\mathrm{lfp}(\lambda Q\subseteq\mathbb{D}^n\cdot[\![P^n]\!](I_P\langle Z^n\leftarrow Q\rangle)(I_D))
\end{aligned}$$

where $I\langle Var\leftarrow Value\rangle$ is the same function as I except that Var is mapped to $Value$, and where lfp denotes the least fixpoint in $\mathbb{D}^n$ with respect to set inclusion.

Fig. 3. Semantics of μ-calculus used in [2]

In general, given a structure of a domain and interpretations I_P and I_D of relational and individual variables, model checking performs the task of checking, whether a formula F is true in this structure (i.e. $[\![F]\!](I_P)(I_D)=1$), in other words, whether the structure is a model for F. However the interpretation of the relational variables has to be supplied before the symbolic model checker can start. It has to be supplied as one BDD for each free relational variable.

In particular for model checking CCS-terms [2, Sect. 8] the transition relations of the transition sys-

The propositional μ-calculus with syntax

$$G ::= \text{True} \mid \neg G \mid G \vee G \mid \langle\alpha\rangle G \mid Z \mid \mu Z \cdot G$$

(where $\alpha \in Act$ and Z ranges over propositional variables) is interpreted relative to a transition system with states S, transitions D, and a valuation V of the propositional variables, see e.g. [12] for details.

It can be embedded into the first order μ-calculus of [2] using the following syntactic transformation $\hat{\cdot}$. This transformation directly reflects the definition of the semantics of the propositional μ-calculus.

$$\begin{aligned}
\widehat{\text{True}} &:= \lambda z[\text{True}] \\
\widehat{\neg G} &:= \lambda z \neg(\hat{G}(z))] \\
\widehat{G_1 \vee G_2} &:= \lambda z[\hat{G}_1(z) \vee \hat{G}_2(z)] \\
\widehat{\langle\alpha\rangle G} &:= \lambda z[\exists z_2[Z_\alpha^2(z, z_2) \wedge \hat{G}(z_2)]] \\
\hat{Z} &:= Z^1 \\
\widehat{\mu Z \cdot G} &:= \mu Z^1[\hat{G}]
\end{aligned}$$

The relational variables (Z_α^2) are used to represent for each action the corresponding portion of the transition relation (other embeddings are conceivable, cf. Sect. 7). If one chooses $\mathbb{D} = S$ and I_P such that $I_P(Z^1) = V(Z)$ for all propositional variables the following holds for each propositional μ-calculus formula G.

$$[\![G]\!]_V = [\![\hat{G}]\!](I_P)(I_D)$$

Fig. 4. Embedding the propositional μ-calculus into the first order μ-calculus of [2]

tems to be checked have to be supplied by the relational-variable interpretation, i.e. as BDDs. Burch et al. leave open where the BDDs for the transition relations come from. The following two sections propose an approach for generating these BDDs.

4 Encoding of transition systems as BDDs

In principle given a certain enumeration $e_0, e_1, \ldots, e_{|S|-1}$ of a finite set S the boolean encoding is obvious, one needs $\#S := \lceil \log_2 |S| \rceil$ boolean variables and lets $s_1, \ldots, s_{\#S}$ denote e_i where i is the number one gets when interpreting $s_1 \ldots s_{\#S}$ as a binary digit.

We will use this encoding for the state sets of elementary [2] transition systems and for the set of actions.

[2] We call a transition system elementary if it is not formed by parallel composition, restriction, or relabelling but is given as a list of transitions

For the latter we additionally assume that Act is enumerated $\tau, a, b, \ldots, \bar{a}, \bar{b}, \ldots$, i.e. that τ is encoded as all zeros and any visible action a as $0x_2 \ldots x_{\#A}$ and its complement $\bar{a}$ as $1y_2 \ldots y_{\#A}$ such that for all $i \in \{2, \ldots, \#A\}$ we have $x_i = y_i$. Here $\#A := \lceil \log_2 |Act| \rceil$.

In order to work with BDDs one has to fix a global ordering on the boolean variables used to encode the information. This ordering has great influence on the size of the BDDs. If nothing is known about the structure of a relation to be represented as a BDD not much can be done. However in our case we can use knowledge about the transition relation and in particular about the operations performed on them, most importantly the parallel compostion.

For explanation let us consider a binary relation $D \subseteq S \times S$ as in the introduction. If we need $\#S$ bits to encode S we need $2 \cdot \#S$ bits to encode D, say $r_1, \ldots, r_{\#S}$ to encode the first element of a pair in D and $s_1, \ldots, s_{\#S}$ for the second.

In the introduction we demonstrated for the synchronization case of parallel composition of two such relations that we get small BDDs if all variables of one relation are ordered before those of the other.

However in the asynchronous case where only one component proceeds while the other (say the first) stays in its state we have to check for $\langle\langle r, r'\rangle, \langle s, s'\rangle\rangle$ whether $r = s \wedge \langle r', s'\rangle \in D$, i.e. we have to check whether $r_1 = s_1 \wedge r_2 = s_2 \wedge \ldots \wedge r_{\#S} = s_{\#S}$. But with the variable ordering $r_1 \prec r_2 \prec \ldots r_{\#S} \prec s_1 \prec \ldots \prec s_{\#S}$ for the bits of a relation the BDD for this check explodes. Figure 5 gives an example for $\#S = 3$. In general it has $\Omega(2^{\#S})$ nodes. Therefore we choose a different ordering, namely

$$r_1 \prec s_1 \prec r_2 \prec s_2 \prec \ldots \prec r_{\#S} \prec s_{\#S}.$$

With this improved ordering the BDD for the above example is as given on the right of Fig. 5, i.e. it grows only linearly.

We are now ready to present our encoding for a transition relation $D \subseteq S \times Act \times S$. We choose the variable ordering

$$a_1 \prec a_2 \prec \ldots a_{\#A} \prec r_1 \prec s_1 \prec r_2 \prec s_2 \prec \ldots \prec r_{\#S} \prec s_{\#S}$$

where $a_1 \ldots a_{\#A}$ encodes the action of the transition, $r_1 \ldots r_{\#S}$ encodes the source state of the transition, and $s_1 \ldots s_{\#S}$ encodes the target state. Figure 6 gives an exam-

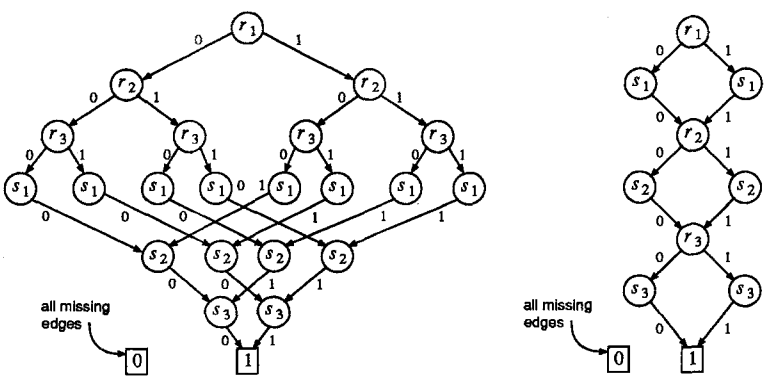
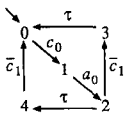


Fig. 5. BDD for $(r_1 = s_1) \wedge (r_2 = s_2) \wedge (r_3 = s_3)$ with poor (left) and good (right) variable ordering



States			Actions		
0	↦	000	τ	↦	000
1	↦	001	a_0	↦	001
2	↦	010	$\bar{a}_0$	↦	101
3	↦	011	c_0	↦	010
4	↦	100	$\bar{c}_0$	↦	110
			c_1	↦	011
			$\bar{c}_1$	↦	111

Fig. 6. A transition system and the encoding of its states and actions

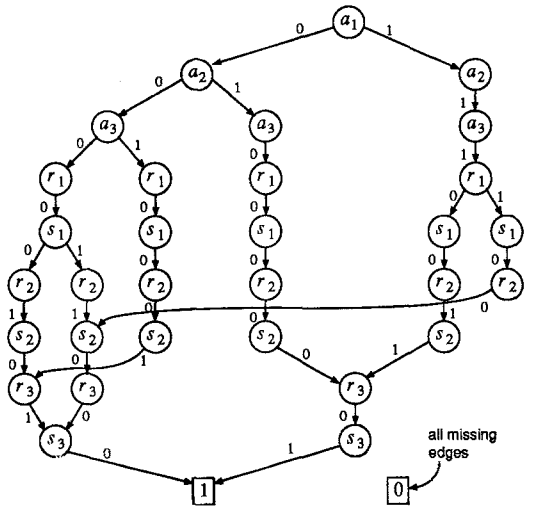


Fig. 7. BDD encoding of the transition relation of the system of Fig. 6

ple transition system and Fig. 7 shows the representation of its transition relation as a BDD.

The reason to put the bits for the actions above those for the states is purely intuitive: actions naturally partition the transition relation. Also this way the proof given in the appendix went through smoothly. However, we did not try to carry out the proof with a different ordering.

We always understand that 00...0 is the initial state.

5 Operators on BDDs

In this section we describe the operators of CCS parallel composition, restriction, and relabelling on BDDs. Note that this comprises the practically most important case of a parallel composition of N processes

$$(P_1 | P_2 | \ldots | P_N) \backslash A$$

where the restriction imposes the wanted synchronization.

The operators on BDDs we present here are new. However they are based on the well-known operators on BDDs for disjunction ($\cdot \vee \cdot$), conjunction ($\cdot \wedge \cdot$), negation ($\neg \cdot$), and restriction of a boolean variable to a certain value ($\cdot|_{var=val}$). For an introduction of the latter operators the reader is referred to Bryant's excellent paper [1].

We assume that the elementary processes (transition systems) are given as BDDs. The BDD for an elementary transition system may easily be formed from the list of triples in the transition relation as follows. Each transition $\langle r, a, s\rangle$ which has encodings $br_1, \ldots, br_{\#S}$, $ba_1, \ldots, ba_{\#A}$, and $bs_1, \ldots, bs_{\#S}$, is considered as one BDD with only the path $a_1 \xrightarrow{ba_1} a_2 \xrightarrow{ba_2} \ldots a_{\#A} \xrightarrow{ba_{\#A}} r_1 \xrightarrow{br_1} s_1 \xrightarrow{bs_1} \ldots r_{\#S} \xrightarrow{br_{\#S}} s_{\#S} \xrightarrow{bs_{\#S}} 1$ leading to 1. All other branches directly lead to 0. The BDD for the transition relation is calculated by performing the disjunction of all these BDDs. In the worst case this yields a BDD with approximately $2^{2\cdot\#S+\#A}$ nodes.

CCS parallel composition

Let B_1 and B_2 be two BDDs representing transition relations D_1 and D_2 over the same set of actions but over disjoint sets of states encoded as described in the previous section. The BDD $B_1 | B_2$ representing the transition relation of the parallel composition is calculated as follows.

Let $a_1, \ldots, a_{\#A}$ be the boolean variables for the actions in both, B_1 and B_2. Let r_i, s_i ($i \in \{1, \ldots, \#S_1\}$) and r'_j, s'_j ($j \in \{1, \ldots, \#S_2\}$) be the boolean variables for the source and target states of D_1 and D_2 respectively. According to the previous section we have the following ordering dependencies

$$a_1 \prec \ldots \prec a_{\#A} \prec r_1 \prec s_1 \prec \ldots \prec r_{\#S_1} \prec s_{\#S_1}$$

and

$$a_1 \prec \ldots \prec a_{\#A} \prec r'_1 \prec s'_1 \prec \ldots \prec r'_{\#S_2} \prec s'_{\#S_2}.$$

We additionally impose that

$$s_{\#S_1} \prec r'_1$$

this leaves the previous orderings and hence B_1, B_2 unchanged.

Note that this choice of combined ordering ensures that the BDD for $B_1 | B_2$ again fulfills our encoding convention for transition systems, i.e. we use the first $\#A$ variables for the actions and than alternatingly one bit for the source and one for the target state of the transition.

We now may calculate $B_1 | B_2$ as

$$(B_1 \wedge Stab_2) \vee (B_2 \wedge Stab_1) \vee C$$

where the missing components are explained below. The $\wedge$ and $\vee$ are operations on BDDs as described in [1]. $Stab_1$ is the BDD for $r_1 = s_1 \wedge \ldots \wedge r_{\#S_1} = s_{\#S_1}$ (cf. Fig. 5). It corresponds to the condition that the first component stays in its state ($Stab_2$ is analogous).

Let E be the BDD calculated as

$$(B_1|_{a_1=0} \wedge B_2|_{a_1=1}) \vee (B_1|_{a_1=1} \wedge B_2|_{a_1=0}).$$

$B_1|_{a_1=0}$ denotes the subgraph of B_1 one gets when restricting the variable a_1 to 0 (see [1] for details) and again $\wedge$ and $\vee$ are operations on BDDs. The BDD E expresses the condition that complementary actions match. The last component C of the parallel composition above

is calculated as

$$(a_1=0)\wedge\ldots\wedge(a_{\#A}=0)\wedge(\exists a_2\,\exists a_3\ldots\exists a_{\#A}\,E).$$

Here $\exists a_i\,G$ for some BDD G is short for $G|_{a_i=0}\vee G|_{a_i=1}$, i.e. it represents an existential quantification of the boolean variable a_i. The existential quantification of $a_2,\ldots,a_{\#A}$ can be implemented directly as a BDD operator. Applied to E it yields a BDD without nodes labelled by variables $a_1,\ldots,a_{\#A}$. The first part of C puts this BDD below the encoding of τ.

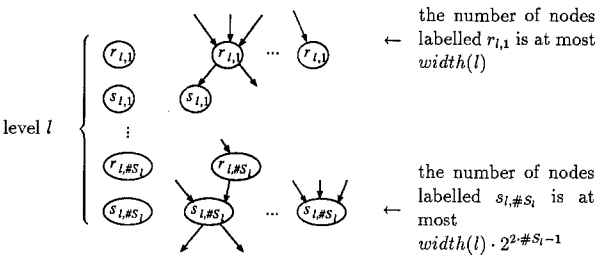


Fig. 8. Level l of a BDD

Restriction

Given a BDD B representing the transition relation of T and a BDD C for the set of actions A (it may be generated straightforwardly from the list of actions in A) the BDD for the transition relation of $T\backslash A$ is simply

$$B\wedge\neg(C|_{a_1=0}\vee C|_{a_1=1})$$

where $\wedge$, $\vee$, $\neg$, and $|_{a_1=b}$ on BDDs are as described in [1].

Relabelling

For B as above let the binary encoding of actions a and b be $ba_1,\ldots,ba_{\#A}$ and $bb_1,\ldots,bb_{\#A}$ respectively. The BDD for the transition relation of $T[b/a]$ is

$$\begin{aligned}&(B\wedge\neg(a_2=ba_2\wedge\ldots\wedge a_{\#A}=ba_{\#A}))\\ &\qquad\vee\\ &((a_2=bb_2\wedge\ldots\wedge a_{\#A}=bb_{\#A})\\ &\quad\wedge B|_{a_2=ba_2}|_{a_3=ba_3}|\ldots|_{a_{\#A}=ba_{\#A}})\end{aligned}$$

where again $\wedge$, $\vee$, $\neg$, and $|_{a_i=b}$ on BDDs are as described in [1].

6 Complexity of resulting BDDs

We have the following results on the size of the generated BDDs.

Theorem 6.1. *Let $T_i=\langle S_i, D_i, z_i\rangle$ for $i\in\{1,\ldots,N\}$ be transition systems where the transition relation is represented as* BDD B_i *according to Sect. 4. The number of nodes of the* BDD $B:=B_1|B_2|\ldots|B_N$ *is*

$$O\left(2^{|Act|}\cdot\sum_{i=1}^{N}|S_i|^2\right).$$

Proof. We will use $\vec{a}, \vec{r}_i, \vec{s}_i$ as abbreviations such that $\vec{a}=\langle a_1,\ldots,a_{\#A}\rangle$ is the variable vector encoding the actions of Act and such that $\vec{r}_i=\langle r_{i,1},\ldots,r_{i,\#S_i}\rangle$ and $\vec{s}_i=\langle s_{i,1},\ldots,s_{i,\#S_i}\rangle$ are the variable vectors encoding source and target state of the transition relation D_i. The BDD B_i ranges over the variables $\vec{a}, \vec{r}_i, \vec{s}_i$ and the BDD B over the variables $\vec{a}, \vec{r}_1, \vec{s}_1, \ldots, \vec{r}_N, \vec{s}_N$. The variable ordering is $\vec{a}\prec\vec{r}_1,\vec{s}_1\prec\ldots\prec\vec{r}_N,\vec{s}_N$ where the ordering within $\vec{r}_i, \vec{s}_i$ is as described in Sect. 4.

For the proof we have a layered view of B. Let us say that all variables for states of transition system T_l belong to level l. See Fig. 8.

For instances $\overrightarrow{ba}\in\mathbb{B}^{\#A}$ and $\overrightarrow{br_i}, \overrightarrow{bs_i}\in\mathbb{B}^{\#S_i}$ let σ_l denote the instantiation $\langle\overrightarrow{ba}, \overrightarrow{br_1}, \overrightarrow{bs_1},\ldots,\overrightarrow{br_{l-1}}, \overrightarrow{bs_{l-1}}\rangle$. Let $\sigma_l(B)$ denote the BDD one gets by instantiating B to σ_l, i.e.

$$\sigma_l(B):=B|_{\vec{a}=\overrightarrow{ba},\vec{r}_1=\overrightarrow{br_1},\vec{s}_1=\overrightarrow{bs_1},\ldots,\vec{r}_{l-1}=\overrightarrow{br_{l-1}},\vec{s}_{l-1}=\overrightarrow{bs_{l-1}}}.$$

This BDD $\sigma_l(B)$ denotes a boolean function of arity $2\cdot\sum_{i=l}^{N}\#S_i$.

To count the nodes of B on level l we determine the number of different such functions one gets by varying σ_l over all instances. This number is called $width(l)$. Since BDDs are a normal form for boolean functions and they are reduced (i.e. have no two isomorphic subgraphs) [1] this number immediately gives an upper bound for the number of nodes in B which are labeled with variables of level l. For $j\in\{1,\ldots,\#S_l\}$ this bound is $width(l)\cdot 2^{2\cdot(j-1)}$ for the label $r_{l,j}$ and $width(l)\cdot 2^{2\cdot(j-1)+1}$ for the label $s_{l,j}$. See Fig. 8. Note that we do not make any assumptions about the part of B within level l hence we have to allow the exponential growth in this part.

Knowing an upper bound W for $\max_l width(l)$ we get an upper bound for $|B|$, the number of nodes in B.

$$\begin{aligned}|B|&\le 2^{\#A}+\sum_{l=1}^{N}\sum_{j=1}^{\#S_l}width(l)\cdot(2^{2\cdot(j-1)}+2^{2\cdot(j-1)+1})\\ &\le 2^{\#A}+\sum_{l=1}^{N}width(l)\cdot 2^{2\cdot\#S_l}\\ &\le 2^{\#A}+W\cdot\sum_{l=1}^{N}2^{2\cdot\#S_l}\\ &\le 2\cdot|Act|+W\cdot\sum_{l=1}^{N}4\cdot|S_l|^2\\ &=O\left(|Act|+W\cdot\sum_{l=1}^{N}|S_l|^2\right)\end{aligned}$$

The rest of the proof is given in the appendix. It calculates W to be $4\cdot 2^{|Act|}$ which in turn proves the stated bound. □

For a fixed set of actions and $n := \max_i |S_i|$ this bound is simply $O(N \cdot n^2)$ which compares favourably to the straightforward worst case bound $O(n^{N+2} \cdot N^2)$ for the number of transitions in $T_1 | \ldots | T_N$.

Worst case bounds for the size of BDDs are rare in literature, hence the above bound is very interesting. However it has to be pointed out that it concerns only the resulting BDD. Nothing is said about the intermediate BDDs which result from the basic operators $(\wedge, \vee, \neg, |_{var=val})$ on BDDs which are used in Sect. 5 to define the CCS parallel composition. In our practical experiments the size of the intermediate BDDs never exceeded the size of the resulting BDD.

If we know more about the structure of the transition systems we are able to give the following tighter bound. See the appendix for details.

Corollary 6.2. *For T_i and B as above let c be the number of sets of visible actions which occur in transitions between any two states of any component, i.e.*

$$c := |\{A \subseteq Act \mid \exists j \in \{1, \ldots, N\} : \exists r, s \in S_j : A = \{\alpha \mid \alpha \neq \tau \wedge \langle r, \alpha, s \rangle \in D_j\}\}|.$$

If no component contains a visible self-loop, i.e. if for all $j \in \{1, \ldots, N\}$ there exists no $s \in S_j$, $\alpha \in Act - \{r\}$ such that $\langle s, \alpha, s \rangle \in D_j$, then

$$|B| = O\left((c + |Act|) \cdot \sum_{i=1}^{N} |S_i|^2\right).$$

According to our experience with elementary transition systems stemming from practical examples, there is seldomly more than one transition between any pair of states. Therefore we expect c to be close to $|Act|$ in typical cases.

Let us remark that the same bounds are true for any relabelling of the B_i's and for arbitrary restrictions (as for example in $((B_1 | B_2) \backslash A_1 | (B_3 | B_4) \backslash A_2) \backslash A_3$). In particular this comprises the most important practical case given at the beginning of the previous section.

7 Implementation

We have implemented the described generation of BDDs for elementary transition systems and the operators of CCS parallel composition, restriction, and relabelling. Furthermore we implemented the symbolic model checker of Burch et al. [2]. The implementations are based on a Prolog system extended by unification in finite algebras [8]. It captures the powerful version of the μ-calculus as used in [2]. In particular the μ-calculus formulas for strong and weak bisimilarity can be checked. The interface consists of two Prolog predicates, *mu_formula* and *mu_relation*, where the former checks μ-calculus formulas, whereas the latter allows the computation of relations from relational terms.

Controlling the ordering of variables

Special care is needed for the ordering of boolean variables encoding the variables in the μ-calculus. For example for computing the transitive closure of a relation R by the term

$$\mu T[\lambda x, y[R(x, y) \vee \exists z[T(x, z) \wedge T(z, y)]]]$$

we want to preserve the original variable ordering, i.e. if the BDD representation of the relation R has the ordering $x_1 \prec y_1 \prec x_2 \prec y_2 \ldots \prec x_n \prec y_n$ we want to preserve that order for all intermediate results of the evaluation as well as for the representation of the transitive closure. We achieve this by the ordering $x_1 \prec z_1 \prec y_1 \prec x_2 \prec z_2 \prec y_2 \ldots \prec x_n \prec z_n \prec y_n$.

Note that a satisfactory variable ordering for all intermediate results cannot always be obtained. Assume for example a binary boolean relation R where the first argument is ordered before the second. If one requests $x \prec y$ in $\lambda x, y[R(x, y) \wedge R(y, x)]$ this would leave $R(x, y)$ unchanged but would force a new ordering in $R(y, x)$. Requesting $y \prec x$ leads to the dual problem.

In order to control the variable ordering we have implemented an annotation mechanism of the form

$$E\{A\}$$

where E is a formula or a relational term according to Fig. 2 and A is an annotation involving *all* bound individual variables in E. In order to allow a unique reference in A the individual variables have to be (re-)named such that no two bound occurrences equal.

The syntax for the annotations is

$$A := z \mid A; A \mid A, A, \ldots, A$$

where z ranges over the individual variables. The annotation $y; z$ means that all boolean variables encoding y are ordered before those encoding z, i.e. $\forall i, j: y_i \prec z_j$. The annotation, $z_1, z_2, \ldots, z_n$ means that the boolean variables encoding the z_i are interleaved, i.e. $\forall i, j, k, l: z_{ij} \prec z_{kl}$ iff $(j = l \wedge i < k) \vee j < l$. For example x, y, z with encodings by boolean variables x_1, x_2, x_3 and y_1 and z_1, z_2 respectively enforces the ordering $x_1 \prec y_1 \prec z_1 \prec x_2 \prec z_2 \prec x_3$.

In the following we show with the example of checking bisimilarity what can be expressed in the μ-calculus with our annotations.

Checking bisimilarity

For checking weak bisimilarity it is necessary to first compute the τ-closure (see e.g. [6]). Let Δ be the relational variable which is interpreted by the BDD which has been generated according to Sect. 4 and 5 for representing a transition relation. The τ-closure of the relation is computed by the following sequence of assignments to fresh relational variables:

$$\begin{aligned}
T &:= \lambda x, y[\Delta(x, \tau, y)] && \{x, y\} \\
T' &:= \lambda x, y[x = y \vee T(x, y)] && \{x, y\} \\
T^* &:= \mu Z[\lambda x, y[T'(x, y) \vee \exists z[Z(x, z) \wedge Z(z, y)]]] && \{x, z, y\} \\
\Delta^\tau &:= \lambda x, \sigma, y[\sigma = \tau \wedge T^*(x, y) \vee \exists z_1[T^*(x, z_1) \\
&\quad \wedge \exists z_2 \cdot [\Delta(z_1, \sigma, z_2) \wedge T^*(z_2, y)]]] && \{\sigma; (x, z_1, z_2, y)\}
\end{aligned}$$

The ordering annotation for T^* has been discussed above. All annotations are chosen to preserve our encoding convention for transition relations in a way that as few as possible reorderings appear.

Note that Δ^τ now stands for a BDD which represents the τ-closure of the transition relation. If Δ_1^τ and Δ_2^τ are such τ-closures of two transition systems then the relation of weak bisimilarity is expressed by

$$\begin{aligned} B := \nu R[\lambda p_1, p_2[&\forall \sigma_1, q_1[\Delta_1^\tau(p_1, \sigma_1, q_1) \\ &\Rightarrow \exists q_2[\Delta_2^\tau(p_2, \sigma_1, q_2) \wedge R(q_1, q_2)]] \\ &\wedge \forall \sigma_2, r_2[\Delta_2^\tau(p_2, \sigma_2, r_2) \\ &\Rightarrow \exists r_1[\Delta_1^\tau(p_1, \sigma_2, r_1) \wedge R(r_1, r_2)]]]] \\ &\{\sigma_1; \sigma_2; (p_1, q_1, r_1, p_2, q_2, r_2)\} \end{aligned}$$

If the individual variables z_1 and z_2 denote the initial states the μ-calculus formula

$$B(z_1, z_2)$$

yields the answer to the question whether the two transition systems are weakly bisimilar. Of course strong bisimilarity can be checked by using Δ_i instead of Δ_i^τ in the definition of B.

8 Example

Milner's example of a simple distributed scheduler has become a benchmark for process algebra tools [7, 10].

The scheduler consists of one starter process and N processes which are scheduled. The communication is organized in a ring. Expressed in CCS [11] the processes are as follows.

$$\begin{aligned} Starter &\stackrel{\text{def}}{=} \overline{c_0}.\mathbf{0} \\ C_0 &\stackrel{\text{def}}{=} c_0.a_0.(\tau.\overline{c_1}.C_0 + \overline{c_1}.\tau.C_0) \\ C_1 &\stackrel{\text{def}}{=} c_1.a_1.(\tau.\overline{c_2}.C_1 + \overline{c_2}.\tau.C_1) \\ &\vdots \\ C_{N-1} &\stackrel{\text{def}}{=} c_{N-1}.a_{N-1}.(\tau.\overline{c_0}.C_{N-1} + \overline{c_0}.\tau.C_{N-1}) \end{aligned}$$

Each cycler process C_i awaits the permit c_i to start, performs action a_i, and passes the permit to the next cycler either before or after some internal computation. The transition system for C_0 is given in Fig. 6. The compound process for N cyclers is

$$SCHED_N \stackrel{\text{def}}{=} (Starter | C_0 | C_1 | \ldots | C_{N-1}) \backslash \{c_0, \ldots, c_{N-1}\}.$$

The restriction enforces the synchronization of the cycler processes C_i via the actions c_i. $SCHED_N$ is weakly bisimilar to $SPEC_N \stackrel{\text{def}}{=} a_0.a_1.\ldots.a_{N-1}.SPEC_N$.

Table 1 shows the size of the BDD for $SCHED_N$ and the times needed to check the weak bisimilarity between $SCHED_N$ and $SPEC_N$ compared to conventional tools. The number of states and transitions for $N > 12$ as well as the times for the conventional tools for $N = 20$ are extrapolated estimates. Note that due to the fact that the BDD for a transition relation in general represents not just transitions reachable from the initial state the number of transitions cannot be calculated from the paths in the BDD leading to 1. The times for AUTO and BB are taken from [10], those for Aldébaran from [7]. Times for our system were obtained on a SUN 4/75 (Sparc2) workstation.

To be better comparable one would have to add the times needed for computing the transition relation of $SCHED_N$ to the columns for AUTO, Aldébaran, and BB. Additionally the time needed to compute the transitive closure with respect to τ-transitions has to be added for AUTO and Aldébaran, it is not needed for branching bisimulation. Only the column for this paper includes all these times. On the other hand we used a hardware which is approximately 2.5 times faster.

Applying Corollary 6.2 to $SCHED_N$ yields $c = O(|Act|)$ and hence the size of the BDD representing the transition system of $SCHED_N$ is $O\left(N \cdot \sum_{i=0}^{N-1} |S_i|^2\right) = O(N^2)$.

9 Conclusion

We have presented the missing link, namely the generation of transition relations as BDDs, for exploiting symbolic model checking as a tool for process algebras. The example shows that processes out of the reach of conventional tools can be checked for bisimilarity this way.

We have shown that the BDD for N parallel transition systems grows only linearly in N. Such a worst case bound for BDDs is rate in the literature. On the other

Table 1. Benchmarks for the scheduler

N	States	Transitions	Nodes in BDD	AUTO [4]	Aldébaran [7]	BB [10]	BDD
6	577	2017	427	3.3s	1.9s	0.2s	21s
8	3073	13825	651	57s	24s	1.2s	40s
10	15361	84481	907	–	–	7.4s	87s
12	73729	479233	1200	–	–	53s	145s
14	300000	2000000	1528	–	–	–	233s
16	1200000	8000000	1897	–	–	–	348s
18	4800000	32000000	2297	–	–	–	569s
20	18000000	128000000	2734	(10^9 s)	($7 \cdot 10^7$ s)	(50000 s)	850s

hand it is not clear how the BDDs grow during the symbolic model checking.

Another open problem is the question whether other process algebra operators, in particular recursion can be performed on BDDs similarly efficiently.

Acknowledgement. We thank Peter Warkentin, his fast implementation of operations on BDDs is used for the larger experiments reported in Table 1. Four anonymous referees have supplied helpful comments.

Appendix

This appendix shows how to calculate the bound W needed for the proof of Theorem 6.1. For the BDD B recall that $\sigma_l(B) = B|_{\vec{a}=\overrightarrow{ba}, \vec{r}_1=\overrightarrow{br}_1, \vec{s}_1=\overrightarrow{bs}_1, \ldots, \vec{r}_{l-1}=\overrightarrow{br}_{l-1}, \vec{s}_{l-1}=\overrightarrow{bs}_{l-1}}$. For arbitrary l we want W to be an upper bound for the number of different relations (i.e. characteristic functions) $\sigma_l(B)$ for any instantiation $\sigma_l = \langle \overrightarrow{ba}, \overrightarrow{br}_1, \overrightarrow{bs}_1, \ldots, \overrightarrow{br}_{l-1}, \overrightarrow{bs}_{l-1} \rangle$.

In the following the relations $\sigma_l(B)$ will be stated in terms of the relations $stab_l$, $move_l(A)$ and $SyncAsync_l$. The relation $stab_l$ expresses that all components T_i on level l or lower are stable, i.e. perform no change of state.

$$stab_l \stackrel{\text{def}}{=} \{\langle \overrightarrow{br}_l, \overrightarrow{bs}_l, \ldots, \overrightarrow{br}_N, \overrightarrow{bs}_N \rangle \mid \forall i, l \le i \le N: \overrightarrow{br}_i = \overrightarrow{bs}_i\}$$

For a set A of actions $move_l(A)$ expresses that one component T_j on level l or lower performs a transition with an action of A while all other processes are stable.

$$move_l(A) \stackrel{\text{def}}{=} \{\langle \overrightarrow{br}_l, \overrightarrow{bs}_l, \ldots, \overrightarrow{br}_N, \overrightarrow{bs}_N \rangle \mid \exists \alpha \in A, j, l \le j \le N: \langle \overrightarrow{br}_j, \alpha, \overrightarrow{bs}_j \rangle \in D_j \wedge \forall i, i \neq j, l \le i \le N: \overrightarrow{br}_i = \overrightarrow{bs}_i\}$$

The relation $SyncAsync_l$ states that some process T_j on level l or lower performs an asynchronous τ-transition or that two processes T_j, T_k on level l or lower perform synchronizing transitions. In both cases all other processes on level l or lower are stable.

$$SyncAsync_l \stackrel{\text{def}}{=} move_l(\{\tau\}) \cup \{\langle \overrightarrow{br}_l, \overrightarrow{bs}_l, \ldots, \overrightarrow{br}_N, \overrightarrow{bs}_N \rangle \mid \exists \alpha \in Act, j, k, j \neq k, l \le j, k \le N: \langle \overrightarrow{br}_j, \alpha, \overrightarrow{bs}_j \rangle \in D_j \wedge \langle \overrightarrow{br}_k, \overrightarrow{\alpha}, \overrightarrow{bs}_k \rangle \in D_k \wedge \forall i, k \neq i \neq j, l \le i \le N: \overrightarrow{br}_i = \overrightarrow{bs}_i\}$$

Instantiations σ_l can be divided into non-τ- and τ-transitions (cases **a** and **b** below). The former are always asynchronous. For the following by "the components of σ_l" we mean the components $T_1, \ldots, T_{l-1}$. Let us say that a component T_k is unstable if $\overrightarrow{br}_k \neq \overrightarrow{bs}_k$. Depending on the number of unstable components of σ_l we distinguish the cases **a0** (no unstable component), **a1** (one unstable component), and **a2** (two or more unstable components). Let $L := \{1, \ldots, l-1\}$.

Case a $\overrightarrow{ba} \neq \tau$

Case a0 $\forall i \in L: \overrightarrow{br}_i = \overrightarrow{bs}_i$

$$\sigma_l(B) = \begin{cases} move_l(\{\overrightarrow{ba}\}) \cup stab_l & \text{if } \exists j \in L: \langle \overrightarrow{br}_j, \overrightarrow{ba}, \overrightarrow{br}_j \rangle \in D_j \\ move_l(\{\overrightarrow{ba}\}) & \text{otherwise} \end{cases}$$

Case a1 $\exists j \in L: \overrightarrow{br}_j \neq \overrightarrow{bs}_j$ and $\forall i \in L, i \neq j: \overrightarrow{br}_i = \overrightarrow{bs}_i$

$$\sigma_l(B) = \begin{cases} stab_l & \text{if } \langle \overrightarrow{br}_j, \overrightarrow{ba}, \overrightarrow{bs}_j \rangle \in D_j \\ \emptyset & \text{otherwise} \end{cases}$$

Case a2 $\exists j, k \in L, j \neq k: \overrightarrow{br}_j \neq \overrightarrow{bs}_j \wedge \overrightarrow{br}_k \neq \overrightarrow{bs}_k$

$\sigma_l(B) = \emptyset$

The τ-transitions are more complicated because we have to consider transitions resulting from two synchronizing processes as well as asynchronous τ-transitions of a single process. Depending on the number of unstable components of σ_l we distinguish cases **b0**, **b1**, **b2** and **b3**. The case **b3** covers those σ_l which contain three or more unstable components.

The cases **b0**, **b1** and **b2** have to check whether for σ_l a process can proceed asynchronously and whether two processes can synchronize. For this purpose we define the predicates Async-τ and Sync-τ.

$$\text{Async-}\tau(j) \stackrel{\text{def}}{\Leftrightarrow} \langle \overrightarrow{br}_j, \tau, \overrightarrow{bs}_j \rangle \in D_j$$

$$\text{Sync-}\tau(j, k) \stackrel{\text{def}}{\Leftrightarrow} \exists \alpha \in Act: \langle \overrightarrow{br}_j, \alpha, \overrightarrow{bs}_j \rangle \in D_j \wedge \langle \overrightarrow{br}_k, \overrightarrow{\alpha}, \overrightarrow{bs}_k \rangle \in D_k$$

Case b $\overrightarrow{ba} = \tau$

Case b3 $\exists M \subseteq L, |M| > 2: \forall i \in M: \overrightarrow{br}_i \neq \overrightarrow{bs}_i$

This means that three or more processes perform a transition in B, which is impossible from the definition of the parallel composition operator.

$\sigma_l(B) = \emptyset$

Case b2 $\exists j, k \in L, j \neq k: \overrightarrow{br}_j \neq \overrightarrow{bs}_j, \overrightarrow{br}_k \neq \overrightarrow{bs}_k, \forall i \in L, k \neq i \neq j: \overrightarrow{br}_i = \overrightarrow{bs}_i$

If two processes are making a transition they are synchronizing, i.e. the remaining components $T_j, \ldots, T_N$ must be stable.

$$\sigma_l(B) = \begin{cases} stab_l & \text{if Sync-}\tau(j, k) \\ \emptyset & \text{otherwise} \end{cases}$$

Case b1 $\exists j \in L: \overrightarrow{br}_j \neq \overrightarrow{bs}_j \; \forall i \in L, i \neq j: \overrightarrow{r}_i = \overrightarrow{bs}_i$

If process T_j can perform an asynchronous τ-transition or synchronize with a process T_i, $i \in L$ the relation $stab_l$ is contained in $\sigma_l(B)$. The other part of $\sigma_l(B)$ stems from synchronisations of T_j with a process T_i, $i \ge l$. For the latter case we define the set $A_1 := \{\alpha \mid \langle \overrightarrow{br}_j, \alpha, \overrightarrow{bs}_j \rangle \in D_j \wedge \alpha \neq \tau\}$.

$$\sigma_l(B) = \begin{cases} move_l(A_1) \cup stab_l & \text{if Async-}\tau(j) \text{ or} \\ & \exists k \in L, j \neq k: \text{Sync-}\tau(j, k) \\ move_l(A_1) & \text{otherwise} \end{cases}$$

Case b0 $\forall i \in L: \overrightarrow{br}_i = \overrightarrow{bs}_i$

Again we have the same possibilities as in case **b1**, but the value of j is not fixed, it may be in the range of 1 to $l-1$. As components 1 to $l-1$ are stable we also have the possibility that processes $T_l, \ldots, T_N$ perform an asynchronous transition or a pair of synchronizing transitions.

Let $A_0 := \{\alpha \mid \exists j \in L: \langle \overrightarrow{br}_j, \alpha, \overrightarrow{br}_j \rangle \in D_j \wedge \alpha \neq \tau\}$.

$$\sigma_l(B) = \begin{cases} move_l(A_0) \cup SyncAsync_l \cup stab_l \\ \quad \text{if } \exists j \in L: \text{Async-}\tau(j) \text{ or } \exists j, k \in L, j \neq k: \text{Sync-}\tau(j, k) \\ move_l(A_0) \cup SyncAsync_l \quad \text{otherwise} \end{cases}$$

To get an upper bound for W we count the possible number of relations for each case allowing any subset of Act for A_0 and A_1. Cases **b0** and **b1** each may contribute $2 \cdot 2^{|Act|}$ different relations. Case **b2** may contribute two relations but these are already counted for case **b1** if $A_1 = \emptyset$. Cases **b3**, **a2**, **a1** are similar. Case **a0** contributes $2 \cdot |Act|$ different relations but these are already counted for case **b2** if $A_1 = \{\overrightarrow{ba}\}$. Summing this up yields $4 \cdot 2^{|Act|}$ as an upper bound for W.

Critical cases for a better approximation are **b0** and **b1** from which we can get a smaller upper bound W. If no process contains a visible self-loop the set A_0 in case **b0** is empty. Under this condition case **b1** may contribute $2 \cdot c$ different relations where c is the number of sets of visible actions which occur in transitions between any two states of any component, see Corollary 6.2. Case **b0** contributes two relations. The other cases are as above however they are not always counted in case **b1**. In conclusion we get $2 \cdot |Act| + 4 + 2 \cdot c$ as an upper bound for W under the condition of Corollary 6.2.

References

1. Bryant RE: Graph-based algorithms for boolean function manipulation. IEEE Trans Comput C-35 8:677–691 (1986)

2. Burch JR, Clarke EM, McMillan KL, Dill DL, Hwang LJ: Symbolic model checking: 10^{20} states and beyond. In: Proceedings of the 5th IEEE Symp. on Logic in Computer Science, Philadelphia 1990. Computer Society Press, 1990, pp 428–439
3. Cleaveland R, Parrow J, Steffen B: The concurrency workbench. In: Sifakis J (ed) Automatic verification methods for finite state systems. Proceedings, Grenoble 1989. Lect Notes Comput Sci, vol 407. Springer, Berlin Heidelberg New York 1990, pp 24–37
4. de Simone R, Vergamini D: Abord auto. Rapports Techniques 111, INRIA, Sophia Antipolis 1989
5. Emerson EA, Lei C-L: Efficient model checking in fragments of the propositional mu-calculus. In: Proc. of the First Annual Symp. on Logic in Computer Science. Computer Society Press, 1986, pp 267–278
6. Estenfeld K, Schneider H-A, Taubner D, Tidén E: Computer aided verification of parallel processes. In: Pfitzmann A, Raubold E (eds) VIS '91 Verläßliche Informationssysteme. Proceedings, Darmstadt 1991. Informatik Fachberichte, vol 271. Springer, Berlin Heidelberg New York 1991, pp 208–226
7. Fernandez J-C: An implementation of an efficient algorithm for bisimulation equivalence. Sci of Comput Program 13:219–236 (1989/90)
8. Filkorn T: Unifikation in endlichen Algebren und ihre Integration in Prolog. Master's Thesis, Techn. Universität München 1988
9. Fischer S, Scholz A, Taubner D: Verification in process algebra of the distributed control of track vehicles – A case study. In: Proceedings of CAV'92, Workshop on Computer-Aided Verification, Montreal 1992
10. Groote JF, Vaandrager F: An efficient algorithm for branching bisimulation and stuttering equivalence. In: Automata, languages and programming, ICALP '90. Lect Notes Comput Sci, vol 443. Springer, Berlin Heidelberg New York 1990
11. Milner R: Communication and concurrency. Prentice Hall, New York 1989
12. Stirling C, Walker D: Local model checking in the modal mu-calculus. In: Díaz J, Orejas F (eds) TAPSOFT '89. vol 1. Proceedings, Barcelona 1989. Lect Notes Comput Sci, vol 351. Springer, Berlin Heidelberg New York 1989, pp 369–383
13. Taubner D: Finite representations of CCS and TCSP programs by automata and petri nets. Lect Notes Comput Sci 369:61–94 (1989)