Compositional Minimization of Finite State Systems*

Susanne Graf †   Bernhard Steffen ‡

Aachener Informatikberichte
Nr. 91 – 23

Abstract

In this paper we develop a compositional method for the construction of the minimal transition system that represents the semantics of a given reactive system. The point of this method is that it exploits structural properties of the reactive system in order to avoid the consideration of large intermediate representations. Central is the use of *interface specifications* here, which express constraints on the components' communication behaviour, and therefore to control the state explosion caused by the interleavings of actions of communicating parallel components.

The *effect* of the method, which is developed for bisimulation semantics here, depends on the structure of the reactive system under consideration, in particular on the accuracy of the interface specifications. However, its *correctness* does not: every "successful" construction is guaranteed to yield the desired minimal transition system, independently of the correctness of the interface specifications provided by the designer.

*This paper is an extended version of . – The work has been partially supported by ESPRIT Basic Research Action 'SPEC'

†IMAG-LGI, BP 53X, F-38041 Grenoble
‡Lehrstuhl für Informatik II, RWTH Aachen

1 Motivation

Many tools for the automatic analysis or verification of finite state concurrent systems are based on the construction of the global state graph of the system under consideration (cf. [CES83,FSS83,CPS89,CPSb89]). Thus they often fail because of the *state explosion problem*: the state space of a system potentially increases exponentially in the number of its parallel components. To overcome this problem techniques have been developed in order to avoid the construction of the complete state graph (cf. [BFH90,CLM89,ClSte90,Fer88,Jos87,KuMcM89,Kr89,LaTh88,LaXi90,Pnu85,ShGr89, Wa88,Win90,WoLo89"). In this paper we present a method for the compositional minimization of finite state concurrent systems, which is practically motivated by the following observation:

> For the verification of a system it is usually sufficient to consider an abstraction of its global state graph, because numerous computations are irrelevant from the observer's point of view. Such abstractions often allow to reduce the state graph drastically by collapsing semantically equivalent states to a single state without affecting the observable behaviour. For example, the so obtained "minimization" of a complex communication protocol may be a simple buffer.

Let us refer to the size of the original state space of a system S as its *apparent* complexity, and to the size of the minimized state space as its *real* complexity. The intention of our method is to avoid the apparent complexity by compositionally constructing the minimal system representation. Unfortunately, the straightforward idea to just successively combine and minimize the components of the system is not satisfactory, because "local" minimization does not take context constraints into account and therefore may even lead to subsystems with a higher real complexity than the apparent complexity of the overall system. This is mainly due to the fact that parts need to be considered that can never be reached in the global context. *Partial* or *loose* specifications allow to "cut off" these unreachable parts (see section 4.3). As in [ClSte90,Kr89,LaTh88,ShGr90,Wa88]) we will exploit this feature to take advantage of context information. Furthermore, we will refer to the size of the maximal transition system that is encountered by our method as the *algorithmic* complexity.

Our method is tailored for establishing $\mathcal{P} \models \phi$, i.e., whether $\mathcal{P}$ satisfies the property ϕ, when $\mathcal{P}$ is a system of the form $(\, p_1 \|_{\mathtt{I}_1} \cdots \|_{\mathtt{I}_{n-1}} p_n \,)\,\langle L \rangle$. To simplify the development of our theory, we assume that the processes p_i are already given as transition systems and that $\mathtt{I}_i$ are *interface specifications*, namely supersets of the set of sequences that can be observed at the associated interfaces which are also represented by transition systems; $\|$ represents the parallel composition operator and $\langle L \rangle$ a hiding operator that abstracts from the activities considered as internal by transforming them into the unobservable action τ (see Section 3.2).

The point of our method is the successive construction of *partially defined* transition systems P_i with the following properties:

1. P_i is less specified than $p_1 \| \cdots \| p_i$, i.e., is smaller in the sense of $\preceq$ (see Definition 4.2).

2. P_n is semantically equivalent to the full system $(p_1 \| \cdots \| p_n)\langle L \rangle$, whenever the interface specifications are *correct* (see Definition 5.3).

3. P_i has the least number of states and transitions in its semantic equivalence class.

Subsequently, the validation of $P_n \models \phi$ completes the proof. Of course, this requires that ϕ is preserved by the semantic equivalence under consideration. In this paper, we are dealing with a refinement of observational equivalence (see Definition 4.1). However, the method also adapts to other equivalences (see Section 7).

Important factor in this approach are the interface specifications, which should be provided by the program designer. However, the correctness of the method does not depend on the correctness of these specifications. They are only used to "guide" the proof. Thus wrong interface specifications will never lead to wrong proofs. They may only prevent a successful verification of a valid statement. It should be noted that the total definedness (see Definition 3.1) of P_n already implies the observational equivalence of P_n and $\mathcal{P}$, and therefore guarantees the successful verification of all $\approx^d$- consistent properties (see Definition 4.1) satisfied by $\mathcal{P}$.

The power of our method is demonstrated by means of a setup that handles the mutually exclusive access of processes to a common resource. In fact, in this example, the apparent complexity is exponential in the number of components, whereas the algorithmic and the real complexity are linear (see Section 6). This is illustrated by means of numerical results that have been obtained using the Aldebaran verification tool [Fer88]. The method can easily and efficiently be implemented in systems like the Edinburgh Concurrency Workbench [CPS89] or the Aldebaran verification tool [Fer88].

2 Related work

A great effort has already been made in order to avoid the construction of the complete state graph, and therefore to avoid the state explosion problem. Roughly, the proposed methods can be split into two categories, the *compositional verification* and the *compositional minimization.* Characteristic for the former category is that the global system need not be considered at all during the verification process, and for the latter that a minimal semantically equivalent representation of the global system is constructed. This minimal representation can subsequently be used for all kinds of verification.

A pure approach to compositional verification has been proposed by Winskel in [Win90], where rules are given to decompose assertions of the form $\mathcal{P} \models \phi$ depending on the syntax of the program $\mathcal{P}$ and the formula ϕ. Unfortunately, the decomposition rules for processes involving the parallel operator are very restricted. Larsen and Xinxin [LaXi90] follow a similar line, however, their decomposition rules are based on an operational semantics of contexts rather than the syntax.

In order to deal with the problems that arise from parallel compositions Pnueli [Pnu85] proposed a "conditional" inference system, where assertions of the form $\phi \mathcal{P} \psi$ can be derived, meaning that the program $\mathcal{P}$ satisfies the property ψ under the condition that its environment satisfies ϕ. This inference system has been used by Shurek and Grumberg in [ShGr90], where a semi-automatic modular verification method is presented, which, like ours, is based on "guesses" for context specifications. However, in contrast to our method it requires a separate proof of the correctness of these guesses. Another method based on interface specifications which must be proved correct separately is given in [Kr89].

Josko [Jos87] also presented a method, where the assumptions on the environment of a component are expressed by a formula, which must be proved in a separate step. The main disadvantage of his method is that the algorithm is exponential in the size of the assumptions about the environment.

A method of the second category was proposed by Halbwachs et al. in [BFH90]. It constructs directly a transition system minimized with respect to bisimulations by successive refinement of a single state. In this method, which has been tailored for Lustre [CHPP87], symbolic computation is needed in order to keep the expressions small which in general may grow exponentially.

Another approach of this category was presented by Clarke et al. [CLM89]. They exploit the knowledge about the alphabet of interest in order to abstract and minimize the system's components. Using $\langle L \rangle$ operations together with an elementary rule for distributing them over the parallel operator (see Proposition 3.4) our method covers this approach.

Larsen and Thomsen [LaTh88], and Walker [Wa88] use partial specifications in order to take context constraints into account. Our method is an elaboration of theirs. It uses a more appropriate preorder and defines a concrete strategy for (semi-)automatic proofs where the required user support is kept to a minimum.

The methods proposed in [WoLo89,StGr89,KuMcM89,BCG86] are tailored to verify properties of classes of systems that are systematically built from large numbers of identical processes. These methods are somewhat orthogonal to ours. This suggests to consider a combination of both types of methods.

3 Representation of Processes

In this section, we establish our framework in which processes (systems) are labelled transition systems extended by an undefinedness predicate with parallel composition and hiding defined on them. The extension provides a notion of partial definedness, which naturally leads to a specification-implementation relation between processes (see Section 4.3).

3.1 Extended Transition Systems

An *extended finite state transition system* T is a quadruple $(S, \mathcal{A} \cup \{\tau\}, \rightarrow, \uparrow)$ where

1. S is a finite set of *processes* or *states*;
2. $\mathcal{A}$ is a finite *alphabet* of *observable actions*, and τ represents an internal or unobservable action not in $\mathcal{A}$;
3. $\rightarrow$ is a mapping associating with each $a \in \mathcal{A} \cup \{\tau\}$, a *transition relation* $\xrightarrow{a} \subseteq S \times S$;
4. $\uparrow \subseteq S \times 2^{\mathcal{A} \cup \{\tau\}}$ is a predicate expressing *guarded undefinedness*. Given $(p, L) \in \uparrow$, we write $p \uparrow a$ for $a \in L$.

Typically, S is a set of program states, and the relationship $p \xrightarrow{a} q$ indicates that p can evolve to q under the observation of a. Finally, $p \uparrow a$ expresses that an a-transition would allow p to enter an undefined state. We say that p is *a-undefined* in this case. Thus, transition systems involving the undefinedness predicate are only *partially defined* or *specified*. It is this notion of partial specification together with its induced partial order which provides the framework for proving our method correct.

Processes are rooted transition systems, i.e. pairs consisting of a transition system and a designated start state. Given a transition system $T = (S, \mathcal{A} \cup \{\tau\}, \rightarrow, \uparrow)$, we identify (as usual) a state $p \in S$ with the process $((S_p, \mathcal{A}_p \cup \{\tau\}, \rightarrow_p, \uparrow_p), p)$, where

- S_p is the set of states that are reachable from p in T,
- $\mathcal{A}_p = \mathcal{A}$ and
- $\rightarrow_p$ and $\uparrow_p$ are $\rightarrow$ and $\uparrow$ restricted to S_p, respectively.

In future, obvious indices will be dropped. The following property characterizes the subset of "standard" transition systems.

Definition 3.1 *A process is* totally defined *if its undefinedness predicate $\uparrow$ is empty.*

3.2 Critical Processes

We now introduce a binary parallel operator $\|$ and unary hiding operators $\langle L \rangle$, where L is a set of observable actions. Intuitively, $p \| q$ is the parallel composition of p and q with synchronization of the actions common to both of their alphabets and interleaving of the others, and $p\langle L \rangle$ is the process in which only the actions in L are observable.

Definition 3.2 (Operational Semantics)
Let be $p = ((S_p, \mathcal{A}_p \cup \{\tau\}, \rightarrow_p, \uparrow_p), p)$, $q = ((S_q, \mathcal{A}_q \cup \{\tau\}, \rightarrow_q, \uparrow_q), q)$, $p' \in S_p$, $q' \in S_q$ and L a set of actions. Then the alphabets of $p \| q$ and $p\langle L \rangle$ are defined by $\mathcal{A}_{p\langle L \rangle} = \mathcal{A}_p \cap L$ and $\mathcal{A}_{p\|q} = \mathcal{A}_p \cup \mathcal{A}_q$, and their state sets as the subsets of states of $\{p'\langle L \rangle \mid p' \in S_p\}$ and $\{ p' \| q' \mid p' \in S_p, q' \in S_q \}$ which are reachable from the initial states $p\langle L \rangle$ and $p \| q$ according to the following transition relations:

1. $$\frac{p \xrightarrow{a}_p p'}{p\langle L\rangle \xrightarrow{a} p'\langle L\rangle} \quad a \in L$$

2. $$\frac{p \xrightarrow{a}_p p'}{p\langle L\rangle \xrightarrow{\tau} p'\langle L\rangle} \quad a \notin L$$

3. $$\frac{p \xrightarrow{a}_p p'}{p\|q \xrightarrow{a} p'\|q} \quad (\, a \notin \mathcal{A}_q \,)$$

4. $$\frac{q \xrightarrow{a}_q q'}{p\|q \xrightarrow{a} p\|q'} \quad (\, a \notin \mathcal{A}_p \,)$$

5. $$\frac{p \xrightarrow{a}_p p' \quad q \xrightarrow{a}_q q'}{p\|q \xrightarrow{a} p'\|q'} \quad a \neq \tau$$

The undefinedness predicates of $p\|q$ and $p\langle L\rangle$ are defined by:

6. $$\frac{p{\uparrow_p}a}{p\langle L\rangle{\uparrow}\, a} \quad a \in L$$

7. $$\frac{p{\uparrow_p}a}{p\langle L\rangle{\uparrow}\, \tau} \quad a \notin L$$

8. $$\frac{p{\uparrow_p}a}{(p\|q){\uparrow}\, a} \quad a \notin \mathcal{A}_q$$

9. $$\frac{p{\uparrow_p}a}{(p\|q){\uparrow}\, a} \quad q \xrightarrow{a} q'$$

10. $$\frac{q{\uparrow_p}a}{(p\|q){\uparrow}\, a} \quad a \notin \mathcal{A}_p$$

11. $$\frac{q{\uparrow_p}a}{(p\|q){\uparrow}\, a} \quad p \xrightarrow{a} p'$$

12. $$\frac{p{\uparrow_p}a \quad q{\uparrow_q}a}{(p\|q){\uparrow}\, a}$$

Thus $p\!\uparrow a$ $(q\!\uparrow a)$ implies $(p\|q)\!\uparrow a$, whenever q (p) does not preempt the execution of a, i.e., whenever $a \notin \mathcal{A}_q$ or $q \xrightarrow{a} q'$ $(a \notin \mathcal{A}_p$ or $p \xrightarrow{a} p')$. Remember that $\tau \notin \mathcal{A}_p$ for any p, thus $\xrightarrow{\tau}$ is defined by the clauses 2, 3 and 4, and $\uparrow\tau$ by clauses 7, 8, 9, 10 and 11. The exact meaning of this definition will become clear in Section 5.2, where we introduce the reduction operator. We have:

Proposition 3.3 $\|$ *is associative and commutative.*

Thus processes of the form $(p_1\|\cdots\|p_n)\langle L\rangle$ are well-defined. Our method will concentrate on this form[1]. Also in our setting, we have the usual correspondence between the parallel and the hiding operator:

Proposition 3.4 $\forall p, q\, \forall L.\ (p\|q)\langle L\rangle \;=\; (p\langle L\cup \mathcal{A}_q\rangle \,\|\, q\langle L\cup \mathcal{A}_p\rangle)\langle L\cap (\mathcal{A}_p\cup \mathcal{A}_q)\rangle$

This proposition is particularly important, because it allows to localize global hiding informations. In fact, this localization is the essence of the construction of the 'interface processes' in [CLM89].

4 Equivalence and Partial Order

In this section, we define a semantics of extended labelled transition systems in terms of observational equivalence (cf. [Mi80]) and establish a specification-implementation relation in terms of a preorder, which is compatible with this semantics. This preorder plays a key role in the correctness proof of our method.

4.1 Guarded Undefinedness

The $\rightarrow$ relation does not distinguish between observable and unobservable actions. In order to reflect that τ is internal, and hence not visible, we define the *weak* transition relation $\Rightarrow$ and the *weak* undefinedness predicate $\Uparrow$ for arbitrary $p, q \in S$ and $a \in \mathcal{A}$ as the least relation defined by:

1. $p \xrightarrow{\tau}{}^{*} \xrightarrow{a} \xrightarrow{\tau}{}^{*} q$ implies $p \overset{a}{\Longrightarrow} q$
2. $p \xrightarrow{\tau}{}^{*} q$ implies $p \overset{\varepsilon}{\Longrightarrow} q$
3. $q\!\uparrow a \;\wedge\; p \overset{\varepsilon}{\Longrightarrow} q$ implies $p\!\Uparrow a$.
4. $q\!\uparrow \tau \;\wedge\; p \overset{\varepsilon}{\Longrightarrow} q$ implies $p\!\Uparrow \varepsilon$.
5. $q\!\Uparrow \varepsilon \;\wedge\; p \overset{a}{\Longrightarrow} q$ implies $p\!\Uparrow a$.
6. $p\!\Uparrow \varepsilon$ implies $p\!\Uparrow a$.

As usual, the effect of weakening is to swallow the invisible τ-actions.

[1] This form is called *standard concurrent form* in CCS ([Mi80,Mi89]).

4.2 Semantic Equivalence

Our notion of semantics is defined by means of the following equivalence relation[2]:

Definition 4.1 *$\approx^d$ is the union of all relations $R \subseteq S \times S$ satisfying that pRq implies for all $a \in \mathcal{A}$:*

1. *$p{\Uparrow}a$ if and only if $q{\Uparrow}a$*
2. *$p \stackrel{a}{\Rightarrow} p'$ implies $\exists q'.\ q \stackrel{a}{\Rightarrow} q' \wedge p'Rq'$*
3. *$q \stackrel{a}{\Rightarrow} q'$ implies $\exists p'.\ p \stackrel{a}{\Rightarrow} p' \wedge p'Rq'$*

Note that $\approx^d$ coincides with the well-known observational equivalence $\approx$ (cf. [Mi80,Mi89]) if the first of the three defining requirements is dropped.

4.3 The Specification - Implementation Relation

The following preorder which intuitively defines a "less defined than" relation between processes is the basis of the framework in which we establish the correctness of our method:

Definition 4.2 *$\preceq$ is the union of all relations R satisfying that pRq implies for all $a \in \mathcal{A}$:*

1. *$p \stackrel{a}{\Rightarrow} p'$ implies $\exists q'.\ q \stackrel{a}{\Rightarrow} q' \ \wedge\ p'Rq'$*
2. *$\neg(p{\Uparrow}a)$ implies $(\neg(q{\Uparrow}a) \ \wedge\ q \stackrel{a}{\Rightarrow} q'$ implies $\exists p'.\ p \stackrel{a}{\Rightarrow} p' \ \wedge\ p'Rq')$*

$\preceq$ is a variant of the divergence preorder $\lessapprox$ (cf. [Wa88]) in which a-divergence does not require the potential of an a-move. Our modification serves for a different intend. We do not want to cover divergence, i.e. the potential of an infinite internal computation, but (guarded) undefinedness. This establishes $\preceq$ as a specification-implementation relation: a partial specification p is met by an implementation q if and only if $p \preceq q$; in contrast to [ClSte90,Wa88] we do not require an implementation of an a-undefined process to possess any a-transition. This modification enhances the practicality of preorders as specification-implementation relations. In fact, similar definitions of preorders already appeared in [Ste89,SI,Sti87], but have not been investigated as specification-implementation relations.

Observational equivalence $\approx$, divergence preorder $\lessapprox$, and our preorder $\preceq$ induce slightly different semantics on processes. However, it turns out that $\approx^d$ is a refinement of all of them:

Proposition 4.3 *If $p \approx^d q$ then $p \approx q$ and $p \lessapprox q$ and $p \preceq q$, and for totally defined processes $\approx^d$ and $\preceq$ and $\approx$ all coincide.*

Moreover, it can be proved in the usual way that both $\|$ and $\langle L \rangle$ preserve $\preceq$ and $\approx^d$.

[2] A similar definition has been given in [ClSte90].

Proposition 4.4 *For all processes* p, p' *and* q*, and all sets of actions* L*, we have:*

1. $p \preceq p'$ implies $p\|q \preceq p'\|q$
2. $p \approx^d p'$ implies $p\|q \approx^d p'\|q$
3. $p \preceq p'$ implies $p\langle L\rangle \preceq p'\langle L\rangle$
4. $p \approx^d p'$ implies $p\langle L\rangle \approx^d p'\langle L\rangle$

5 The Reduction Method

5.1 Interface Specifications

In this section we introduce our notion of *interface specification* together with a notion of *correctness*, which guarantees the success of our method. These notions concentrate on the set of observable sequences that may pass the interface. Thus the *exact* specification of the interface between p and q is the *language* of $(p\|q)\langle \mathcal{A}_p \cap \mathcal{A}_q\rangle$, i.e. its set of observable sequences.

The well-definedness of the following definition follows from elementary fixpoint theory:

Definition 5.1 *The* language*,* $\mathcal{L}(p)$*, of a partially defined process* p *is defined as the least fixpoint of the following equation system:*

$$\mathcal{L}(p) = \begin{cases} \mathcal{A}_p^* & \text{if } p \Uparrow \varepsilon \\ \bigcup \{\, a \cdot \mathcal{L}_a(p) \mid \mathcal{L}_a(p) \neq \emptyset \,\} \cup \{\varepsilon\} & \text{otherwise} \end{cases}$$

and

$$\mathcal{L}_a(p) = \begin{cases} \mathcal{A}_p^* & \text{if } p \Uparrow a \\ \bigcup \{\, \mathcal{L}(p') \mid p \stackrel{a}{\Rightarrow} p' \,\} & \text{otherwise} \end{cases}$$

for any action a*. Furthermore, given a language* $\mathcal{L}$*, we denote the language of its* a*-suffixes,* $\{w \mid a \cdot w \in \mathcal{L}\,\}$*, by* $\mathcal{L}_a$*.*

Note that this definition is standard for totally defined processes. The language of an a-undefined process is considered to be any sequence of actions starting with a and the language of a ϵ-undefined process is $\mathcal{A}^*$. For this definition of the language of a process p the following property is straightforward.

Proposition 5.2 $\forall p, q \;\; \mathcal{L}((p\|q)\langle \mathcal{A}_p \cap \mathcal{A}_q\rangle) = \mathcal{L}(p\langle \mathcal{A}_q\rangle) \cap \mathcal{L}(q\langle \mathcal{A}_p\rangle)$

We are going to use interface specifications in order to express context constraints. Thus interface specifications are correct or safe if the corresponding exact interface specification is more constraint. This motivates the following definition:

Definition 5.3 *Given two processes* p *and* q*, we define:*

1. *A totally defined process* $\mathtt{I}$ *is an* interface specification *for* p *iff* $\mathcal{A}_{\mathtt{I}} \subseteq \mathcal{A}_p$*, and it is an interface specification for* p *and* q *if it is an interface specification for both* p *and* q.
2. *An interface specification* $\mathtt{I}$ *for* p *and* q *is called* correct *for* p *and* q *iff* $\mathcal{L}((p\|q)\langle \mathcal{A}_p \cap \mathcal{A}_q\rangle) \subseteq \mathcal{L}(\mathtt{I})$.

The set of all interface specifications for p *is denoted by* $\mathcal{I}(p)$*, and the set of all correct interface specifications for* p *and* q *is denoted by* $\mathcal{I}(p,q)$.

Proposition 5.8 will show that these language-based definitions are adequate for our purpose.

5.2 The Reduction Operator

For a process p and an interface specification $\mathtt{I} \in \mathcal{I}(p)$ the reduction $\Pi_{\mathtt{I}}(p)$ of p with respect to $\mathtt{I}$ is essentially the projection of $p\|\mathtt{I}$ onto its first component.

Definition 5.4 (The Reduction Operator)
Let be p *a process and* $\mathtt{I} \in \mathcal{I}(p)$*. Then we define* $\Pi_{\mathtt{I}}(p) = ((S, \rightarrow, \mathcal{A} \cup \{\tau\}, \uparrow), p)$ *by*

1. $S = \{q \in S_p \mid \exists i \in S_{\mathtt{I}}.\ q\|i \in S_{p\|\mathtt{I}}\}$
2. $\mathcal{A} = \mathcal{A}_p$
3. $\forall q, q' \in S\ \forall a \in \mathcal{A} \cup \{\tau\}.\ \ q \xrightarrow{a} q'$ iff
   $\exists i, i' \in S_{\mathtt{I}}.\ q\|i \xrightarrow{a}_{p\|\mathtt{I}} q'\|i'$ *and* $q\|i$ *reachable in* $p\|\mathtt{I}$
4. $\forall q \in S\ q\uparrow\tau$ *iff* $q\uparrow\tau$ *in the transition system of* p
5. $\forall q \in S\ \forall a \in \mathcal{A}.\ q\uparrow a$ *iff one of the following conditions holds:*
   (a) $q\uparrow a$ *in the transition system of* p
   (b) $\exists q' \in S_p.\ q \xrightarrow{a}_p q'$ and $\nexists q' \in S.\ q \xrightarrow{a} q'$

The only difference between $\Pi_{\mathtt{I}}(p)$ and the projection of $p\|\mathtt{I}$ onto p concerns the undefinedness predicates: $\Pi_{\mathtt{I}}(p)$ inherits all undefinedness predicates from p, and new ones are introduced where transitions of p have been "cut" away by $\mathtt{I}$. The point of the reduction operator is that for correct interface specifications this second kind of undefinedness disappears again in the full context $\Pi_{\mathtt{I}}(p)\|q$. This holds, because if an a-transition of p has been replaced by $\uparrow a$, this predicate disappears again in $\Pi_{\mathtt{I}}(p)\|q$ exactly if q, in its corresponding state, preempts the execution of an a-transition. Thus the presence of an $\uparrow a$ in $\Pi_{\mathtt{I}}(p)\|q$ indicates the faultiness of the interface specification, whenever p and q are totally defined processes.

5.3 Representation Independency

First, we observe that $\Pi_{\mathtt{I}}(p)$ is determined by p and the set of all pairs

$$(\, q,\ \{i \in S_{\mathtt{I}} \mid q\|i \in S_{p\|\mathtt{I}}\}\,)$$

The set of states of $\Pi_{\mathtt{I}}(p)$ is just given by the first components of these pairs with non-empty second component and, in fact, the impact of $\mathtt{I}$ on its transition and divergence relation only depends on the following potential initial action sets for states q in the reduced process:

$$IL_q =_{df} \{a \mid \exists q\|i \in S_{p\|\mathtt{I}}.\ i \xrightarrow{a}\}$$

This shows that the set of pairs (q, IL_q) is already sufficient to construct $\Pi_{\mathtt{I}}(p)$ from p by means of a finite sequence of reductions steps of the following kind:

> If q has an a-transition in p and $a \notin IL_q$ then eliminate all the a-transitions and add $q{\uparrow}\, a$ to the divergence relation.

Obviously, these transformation steps successively lead to processes that are smaller in the preorder $\preceq$. Thus we have:

Proposition 5.5 $\forall p\, \forall \mathtt{I} \in \mathcal{I}(p).\ \Pi_{\mathtt{I}}(p) \preceq p$

In order to prove representation independency, it is enough to show that we can compute the pairs (q, IL_q) starting from p and $\mathcal{L}(\mathtt{I})$, i.e., without referring to the structure of $\mathtt{I}$. This can be done by means of a simple iterative procedure, which labels the states q of p with

$$\bigcup \{\, \mathcal{L}(i) \mid q\|i \in S_{p\|\mathtt{I}}\, \}$$

by successive collection. Heart of this procedure is the following functional

$$\mathcal{E}^p_{\mathtt{I}} :\ \text{Languages} \times \text{Actions} \ \rightarrow\ \text{Languages}$$

with

$$\mathcal{E}^p_{\mathtt{I}}(\mathcal{L}, a) \ = \ \begin{cases} \mathcal{L} & \text{if } a \in \mathcal{A}_p \backslash \mathcal{A}_{\mathtt{I}} \\ \mathcal{L}_a & \text{otherwise, i.e., if } \ a \in \mathcal{A}_{\mathtt{I}}\ (\subseteq \mathcal{A}_p) \end{cases}$$

Remember, $\mathcal{L}_a$ is the set of $\mathcal{L}$'s a-suffixes (see Definition 5.1).

Note that $\mathcal{E}^p_{\mathtt{I}}$ does not depend on the specific representation of the interface $\mathtt{I}$. In fact, it only depends on the *alphabets* of $\mathtt{I}$ and p. Moreover, we obviously have:

Lemma 5.6 *$\mathcal{E}^p_{\mathtt{I}}$ is* additive *in its first component, i.e.,*

$$\mathcal{E}^p_{\mathtt{I}}(\bigcup \{\, \mathcal{L}_k \mid k \geq 0\}, a\,) \ = \ \bigcup \{\, \mathcal{E}^p_{\mathtt{I}}(\mathcal{L}_k,\ a) \mid k \geq 0\,\}$$

In particular, $\mathcal{E}^p_{\mathtt{I}}$ is monotonic in its first component.

The iterative procedure works by successively enlarging approximative labellings according to the following two steps:

1. it initially labels p with $\mathcal{L}(\mathtt{I})$ and all the other states with the empty language
2. if a state q is currently labelled by $\mathcal{L}$ and q' is one of its a-successors then it adds $\mathcal{E}^p_{\mathtt{I}}(\mathcal{L}, a)$ to the current language labelling q', until a fixpoint is reached.

According to the additivity of $\mathcal{E}^p_{\mathtt{I}}$ in its first component, this procedure[3] determines the desired action sets IL_q.

Proposition 5.7 *Starting with $\mathcal{L} = \mathcal{L}(\mathtt{I})$ the reduction procedure computes the following state labelling for p[4]:*

$$\forall q \in S_p. \quad q \leftarrow \bigcup \{ \mathcal{L}(i) \mid q \| i \in S_{p\|\mathtt{I}} \}$$

Remembering that the whole reduction process is determined by the IL_q, we obtain:

Proposition 5.8 $\forall p\, \forall \mathtt{I}, \mathtt{I}' \in \mathcal{I}(p).\ \ \mathcal{L}(\mathtt{I}) = \mathcal{L}(\mathtt{I}')$ implies $\Pi_{\mathtt{I}}(p) = \Pi_{\mathtt{I}'}(p)$

In order to complete the picture about the relationship between interfaces and languages, we state the following proposition, which can be obtained in the same spirit by investigating Definition 5.4:

Proposition 5.9 *For any process p and any interface specifications $\mathtt{I}$ and $\mathtt{I}'$ for p,*

1. $\mathcal{L}(\mathtt{I}) \subseteq \mathcal{L}(\mathtt{I}')$ implies $\Pi_{\mathtt{I}}(p) \preceq \Pi_{\mathtt{I}'}(p)$

2. $\mathcal{L}(p\langle \mathcal{A}_{\mathtt{I}} \rangle) \subseteq \mathcal{L}(\mathtt{I})$ implies $\Pi_{\mathtt{I}}(p) = p$

The first part is a consequence of the monotonicity of $\mathcal{E}^p_{\mathtt{I}}$ in its first component (Lemma 5.6) and the fact, that smaller sets IL_q obviously give rise to a process which is smaller in the preorder. The second part results from the fact that a reduction with respect to an interface language $\mathcal{L}(\mathtt{I})$ that contains $\mathcal{L}(p\langle \mathcal{A}_{\mathtt{I}} \rangle)$ obviously has no effect on p.

5.4 Correctness of the Reduction Method

Theorem 5.10 (Correctness of the Reduction)
Let p and q be processes and $\mathtt{I} \in \mathcal{I}(p, q)$. Then $p\|q \;=\; \Pi_{\mathtt{I}}(p)\|q$.

Proof: Let $p = ((S_p, \rightarrow_p, \mathcal{A}_p \cup \{\tau\}, \uparrow_p), p)$ and $q = ((S_q, \rightarrow_q, \mathcal{A}_q \cup \{\tau\}, \uparrow_q), q)$ be processes, $\mathtt{I}$ be a correct interface specification for p and q, $\mathcal{A} =_{df} \mathcal{A}_{\mathtt{I}} \;=\; \mathcal{A}_p \cap \mathcal{A}_q$, and

$$\begin{array}{rcl} p\|q & =_{df} & ((S_1, \rightarrow_1, \mathcal{A}_p \cup \mathcal{A}_q \cup \{\tau\}, \uparrow_1), p\|q) \\ \Pi_{\mathtt{I}}(p) & =_{df} & ((S_{p_{\mathtt{I}}}, \rightarrow_{p_{\mathtt{I}}}, \mathcal{A}_p \cup \{\tau\}, \uparrow_{p_{\mathtt{I}}}), p) \\ \Pi_{\mathtt{I}}(p)\|q & =_{df} & ((S_2, \rightarrow_2, \mathcal{A}_p \cup \mathcal{A}_q \cup \{\tau\}, \uparrow_2), p\|q) \end{array}$$

Then both S_1 and S_2 are subsets of $\{ p'\|q' \mid p' \in S_p \wedge q' \in S_q \}$. Thus remains to show that $S_1 = S_2$ and $\rightarrow_1 = \rightarrow_2$ and $\uparrow_1 = \uparrow_2$. For this purpose we define for $i = 1, 2$:

[3]The procedure computes the maximal fixped point solution wrt $\mathcal{E}^p_{\mathtt{I}}$, which coincides with the desired meet over all paths solutions for additive functionals (cf. [KU]).

[4]The time complexity of this computation can be estimated by product of the number of transitions of p and $\mathtt{I}$.

- S_i^n, the subset of states reachable in n steps from the initial state,
- $\rightarrow_i^n$, the set of transitions leaving the states of S_i^n, and
- $\uparrow_i^n$, the undefinedness predicate of states of S_i^n.

Because of $S_1^0 = S_2^0 = \{p \| q\}$, it is enough[5] to verify the following simultaneous induction step for $n \geq 1$ under the induction hypothesis "$S_1^{n-1} = S_2^{n-1}$", in order to complete the proof:

1. $\rightarrow_1^{n-1} = \rightarrow_2^{n-1}$
2. $S_1^n = S_2^n$
3. $\uparrow_1^{n-1} = \uparrow_2^{n-1}$

This needs three auxiliary results:

$$(*) \qquad \forall p' \| q' \in S_1 \; \exists w \in \mathcal{L}(p\langle\mathcal{A}\rangle) \cap \mathcal{L}(q\langle\mathcal{A}\rangle) \subseteq \mathcal{L}(\mathtt{I}). \;\; p\langle\mathcal{A}\rangle \stackrel{w}{\Longrightarrow}_{p\langle\mathcal{A}\rangle} p'\langle\mathcal{A}\rangle$$

$$(**) \qquad p\langle\mathcal{A}\rangle \stackrel{w}{\Longrightarrow}_{p\langle\mathcal{A}\rangle} p'\langle\mathcal{A}\rangle \stackrel{a}{\longrightarrow}_{p\langle\mathcal{A}\rangle} p''\langle\mathcal{A}\rangle \;\wedge\; wa \in \mathcal{L}(\mathtt{I}) \;\; \textit{implies} \;\; p' \stackrel{a}{\longrightarrow}_{p_\mathtt{I}} p''$$

$$(***) \quad p\langle\mathcal{A}\rangle \stackrel{w}{\Longrightarrow}_{p\langle\mathcal{A}\rangle} p'\langle\mathcal{A}\rangle \uparrow_p a \;\wedge\; wa \in \mathcal{L}(\mathtt{I}) \;\; \textit{implies} \;\; p' \uparrow_{p_\mathtt{I}} a$$

Whereas the correctness of the first statement follows immediately from the reachability of in $p' \| q'$ in $p \| q$ and the correctness of the interface specification $\mathtt{I}$ with respect to p and q (see Definition 5.3 and Proposition 5.2), the correctness of $(**)$ and $(***)$ can be derived directly from Definitions 3.2 and 5.4 and the representation independency of the interface specifications.

Now we are going to verify point (1) of the induction step according to the operational rule (Definition 3.2) that justifies $p' \| q' \stackrel{a}{\longrightarrow}_1 p'' \| q''$. There are three cases:

Rule 3: Here we have $a \in \{\mathcal{A}_p \cup \{\tau\}\} \setminus \mathcal{A}_q$ and $q' = q''$, which allows to complete the argument as follows:

$$\begin{array}{lcl} p' \| q' \stackrel{a}{\longrightarrow}_1 p'' \| q'' & \Longleftrightarrow & p' \stackrel{a}{\longrightarrow}_p p'' \\ \text{(Def. 5.2 (2))} & \Longleftrightarrow & p' \stackrel{a}{\longrightarrow}_{p_\mathtt{I}} p'' \\ \text{(Rule 3)} & \Longleftrightarrow & p' \| q' \stackrel{a}{\longrightarrow}_2 p'' \| q'' \end{array}$$

Rule 4: Here we have $a \in \{\mathcal{A}_q \cup \{\tau\}\} \setminus \mathcal{A}_p$ and $p' = p''$, and therefore:

[5] Remember that we are dealing with finite state systems!

$$p'\|q' \xrightarrow{a}_1 p''\|q'' \quad \Longleftrightarrow \quad q' \xrightarrow{a}_q q''$$

$$\text{(Rule 4)} \quad \Longleftrightarrow \quad p'\|q' \xrightarrow{a}_2 p''\|q''$$

Rule 5: Here we have $a \in \mathcal{A}$, $p' \xrightarrow{a}_p p''$ and $q' \xrightarrow{a}_q q''$, and therefore because of $(*)$, which can be applied according to the induction hypothesis:

$$p\langle\mathcal{A}\rangle \stackrel{w}{\Longrightarrow}_{p\langle\mathcal{A}\rangle} p'\langle\mathcal{A}\rangle \xrightarrow{a}_{p\langle\mathcal{A}\rangle} p''\langle\mathcal{A}\rangle \;\wedge\; w \in \mathcal{L}(p\langle\mathcal{A}\rangle) \cap \mathcal{L}(q\langle\mathcal{A}\rangle) \subseteq \mathcal{L}(\mathtt{I})$$

Now the correctness of the interface specification $\mathtt{I}$ with respect to p and q (see Definition 5.3) yields $wa \in \mathcal{L}(\mathtt{I})$ and therefore together with $(**)$ also $p' \xrightarrow{a}_{p_{\mathtt{I}}} p''$. Thus we can apply Rule 5 in order to obtain $p'\|q' \xrightarrow{a}_2 p''\|q''$ as desired. The remaining converse implication "$p'\|q' \xrightarrow{a}_2 p''\|q''$ *implies* $p'\|q' \xrightarrow{a}_1 p''\|q''$" is analogous.

Point (2) of the induction step is an immediate consequence of point (1). Thus it remains to verify point (3).

The τ-undefinedness of a process is not affected by the reduction operator (see clause (4) of Definition 5.2), which leaves us with the case of a-undefinedness. In order to prove the first implication "$(p'\|q')\uparrow_1 a$ *implies* $(p'\|q')\uparrow_2 a$" we must deal with the five possible justifications of $(p'\|q')\uparrow_1 a$ according to the rules given in Definition 3.2:

Rule 8: Here we have $p'\uparrow_p a$ and $a \notin \mathcal{A}_q$ and therefore $p'\uparrow_{p_{\mathtt{I}}} a$, which by application of Rule 8 yields as desired: $(p'\|q')\uparrow_2 a$.

Rule 9: Here we have $p'\uparrow_p a$ and $q' \xrightarrow{a}_q$, and therefore $a \in \mathcal{A}_q$. On the other hand, the induction hypothesis allows to apply $(*)$. Thus we obtain $p\langle\mathcal{A}\rangle \stackrel{w}{\Longrightarrow}_{p\langle\mathcal{A}\rangle} p'\langle\mathcal{A}\rangle\uparrow_p a$ and $wa \in \mathcal{L}(p\langle\mathcal{A}\rangle) \cap \mathcal{L}(q\langle\mathcal{A}\rangle) \subseteq \mathcal{L}(\mathtt{I})$, and therefore together with $(***)$ also $p'\uparrow_{p_{\mathtt{I}}} a$, which by application of Rule 9 yields $(p'\|q')\uparrow_2 a$ as desired.

Rule 10 & 11: Here a slightly simplified version of the proofs above applies.

Rule 12: Here we have $p'\uparrow_p a$ and $q'\uparrow_q a$, which, in a similar fashion as in the case of Rule 9 yields $p'\uparrow_{p_{\mathtt{I}}} a$. Thus we can apply Rule 12 in order to obtain $(p'\|q')\uparrow_2 a$, which completes the final case.

The remaining converse implication "$(p'\|q')\uparrow_2 a$ *implies* $(p'\|q')\uparrow_1 a$" is analogous. $\square$

5.5 The Method

In this section, we show how the reduction operator can be used for the compositional minimization of processes of the form $\mathcal{P} = (p_1\| \cdots \|p_n)\langle L\rangle$. This form is of particular interest, because it is responsible for the state explosion problem and therefore characterizes the processes that are critical during analysis and verification[6]. Our method works by successive construction of minimal extended transition systems for components of $\mathcal{P}$. The point of this construction is that it exploits interface specifications expressing context constraints for the component under investigation. Thus, the *effect* of the method depends on the information, which has been be provided by the designer of the system.

Our method expects $\mathcal{P}$ as to be annotated with interface specifications that describe the interface between the right hand process and the left hand process of the parallel operator they are attached to:

$$\mathcal{P} \;=\; (p_1 \|_{\mathtt{I}_1} p_2 \|_{\mathtt{I}_2} \cdots \|_{\mathtt{I}_{n-1}} p_n)\langle L\rangle$$

where $\mathtt{I}_i$ is an interface specification for $p_1\| \cdots \|p_i$ and $p_{i+1}\| \cdots \|p_n$. We proceed by successive construction of reductions P_i for the prefix processes of $\mathcal{P}$:

$$\underbrace{(\underbrace{\underbrace{p_1 \|_{\mathtt{I}_1}}_{P_1} p_2 \|_{\mathtt{I}_2}}_{P_2} \cdots \|_{\mathtt{I}_{n-1}} p_n)\langle L\rangle}_{P_n}$$

where the P_i are defined as follows:

- $P_1 \;=\; \mathcal{M}(\Pi_{\mathtt{I}_1}(\mathcal{M}(p_1\langle \mathcal{A}_{\mathtt{I}_1} \cup L\rangle)))$
- $P_i \;=\; \mathcal{M}(\Pi_{\mathtt{I}_i}(\mathcal{M}((P_{i-1}\|p_i)\langle \mathcal{A}_{\mathtt{I}_i} \cup L\rangle)))$ for $2 \le i \le n-1$
- $P_n \;=\; \mathcal{M}((P_{n-1}\|p_n)\langle L\rangle)$

and $\mathcal{M}$ is a function that minimizes extended transition systems up to $\approx^d$.

The goal of this method is to avoid unnecessarily large intermediate transition systems during the construction of the minimal transition system representing the semantic equivalence class of $\mathcal{P}$. Thus, it is important to minimize all the intermediate constructions as it is done above.

In the remainder of this section let $\mathcal{P}$ and P_i be as defined above and Q_i denote the parallel composition of all processes with index greater than i. Then we obtain independently of the correctness of the interface specification just by means of Propositions 4.4 and 5.5:

Proposition 5.11 $\forall\, 1 \le i \le n.\ (P_i\|Q_i)\langle L\rangle \preceq \mathcal{P}$

[6]Of course, the method can be applied in a structured way to each of the p_i. Thus, it is possible to analyze complex structures by successive construction of increasingly large transition systems.

This is already enough to guarantee the correctness of the method, i.e., that the success of a subsequent validation of a $\approx^d$-consistent property for P_n proves this property for $\mathcal{P}$. The correctness of the interface specifications comes into play in order to guarantee the success of the method. The following theorem is an immediate consequence of Theorem 5.10:

Theorem 5.12

$$\forall\, 1 \leq i \leq n.\ (\forall j \leq i.\ \mathtt{I}_j \in \mathcal{I}(p_1 \| \cdots \| p_j,\, Q_j)) \ \text{ implies } \ (P_i \| Q_i)\langle L \rangle \approx^d \mathcal{P}$$

In practice, $\mathcal{P}$ is usually totally defined. In this case, the proof of $P_n \approx^d \mathcal{P}$ reduces to the verification of the total definedness of P_n:

Corollary 5.13 *Whenever $\mathcal{P}$ is is totally defined, we have:*

$P_n \approx^d \mathcal{P}$ iff P_n is totally defined.

6 An Example

Finally, we demonstrate our method by means of an example. It describes a system that is intended to ensure round robin access of n processes P_i to a common resource R. Figure 1 illustrates such a system for $n = 4$. The idea is to pass a "token" via the communication channels tk_i and to allow access to R only for the process that currently possesses the token. This process then sends its request via ps_i to the resource R, which responds by transmitting the object requested. The corresponding transmission line is modelled by a buffer B_i. This is motivated by thinking of large objects whose transmission cannot be modelled just by an atomic "handshake" communication.

Let us now assume that we want to prove that the access is modelled as intended. For this purpose we can hide everything but the actions corresponding to the transmission of the token, and subsequently prove that the resulting process is equivalent to the process $Spec(n)$ that just repetedly executes the sequence $tk_1, \cdots, tk_n$, i.e., it is enough to show for

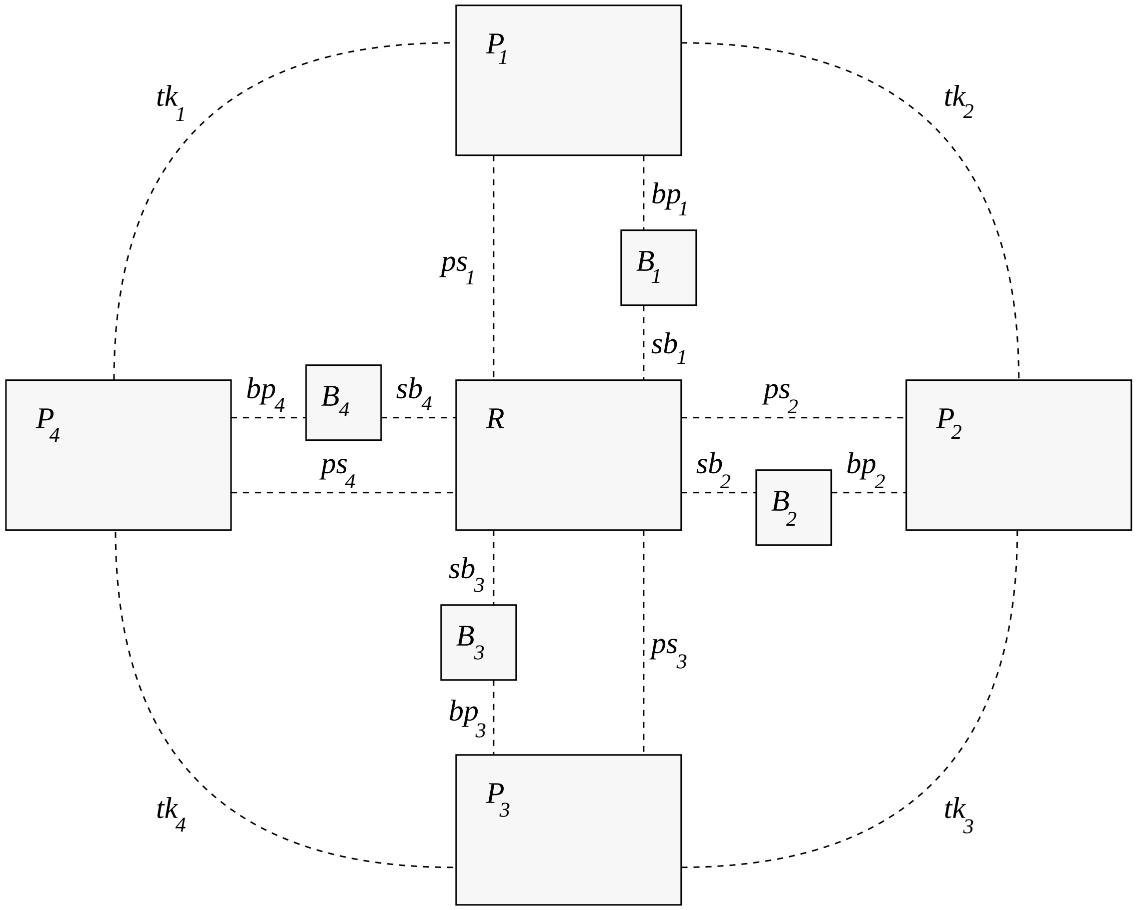
$$System(n) \ =_{df} \ (\, R \,\|P_1\|B_1\| \cdots \|P_n\|B_n\,)\langle\{tk_1, .., tk_n\}\rangle$$

that $System(n) \approx^d Spec(n)$.

It is easy to see that the apparent complexity of $System(n)$ is exponential in n, whereas its real complexity is linear. In fact, it is also possible to obtain an algorithmic complexity that is linear in n. This can be achieved by processing the system according to the structure indicated below, where the $\mathtt{I}_i$ denote exact interface specifications:

$$(\overbrace{R \,\|\, P_1\|B_1} \|_{\mathtt{I}_1} \overbrace{P_2\|B_2} \|_{\mathtt{I}_2} \cdots \|_{\mathtt{I}_{n-1}} \overbrace{P_n\|B_n}\,)\langle\{tk_1, .., tk_n\}\rangle$$

The table below summarizes a numerical investigation of the efficiency of our method by means of the Aldebaran verification tool [Fer88]. It displays the size of the global state graph (its apparent complexity), the size of the maximal transition system constructed



where

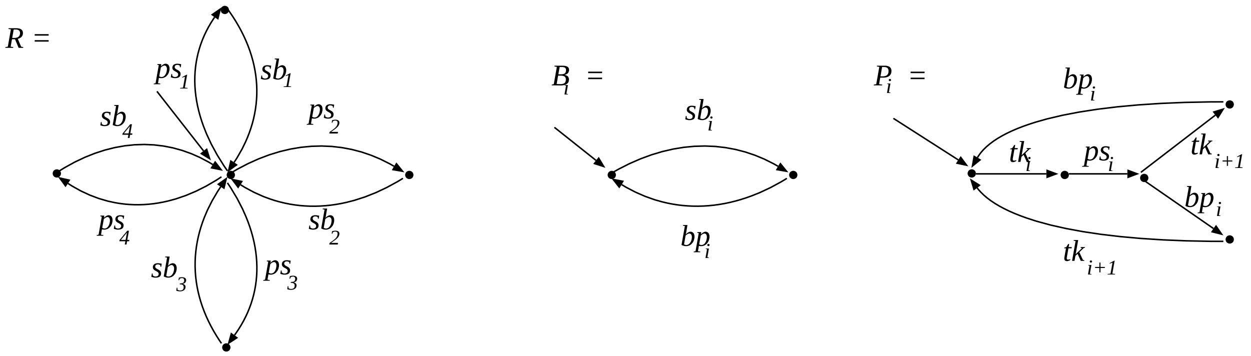


Figure 1: Round Robin Access

during stepwise minimization when exploiting exact interface specifications (the algorithmic complexity), and finally the size of the minimized global state graph (its real complexity).

n	apparent complexity		algorithmic complexity		real complexity	
	states	trans.	states	trans.	states	trans.
4	144	368	20	29	4	4
5	361	1101	24	35	5	5
6	865	3073	28	41	6	6
7	2017	8177	32	47	7	7

It is worth mentioning that the method which works just by stepwise composition and minimization of components encounters transition systems that are even larger than the global state graph:

n	states	trans.
4	96	243
5	324	927
6	972	3024
7	2916	9801

This stresses the importance of interface specifications for automatic proof techniques. It is our opinion that a software designer should always provide these specifications as part of the implementation. We believe that besides enabling automatic verification, this requirement also leads to a transparent and well structured programming[7].

7 Conclusions and Future Work

We have presented a method for the compositional minimization of finite state systems, which is intended to avoid the state explosion problem. This method can be used to support the verification of any property that is consistent with $\approx^d$. However, the method is not tailored to this particular semantic equivalence. Other equivalences can be dealt with by adapting the preorder definition and the minimization function accordingly (cf. Theorem 5.10).

The *effect* of our method, which is intended to get the algorithmic complexity as close as possible to the real complexity, depends on interface specifications, which we assume as to be given by the program designer. However, the *correctness* of the method does not depend on the correctness of these interface specifications. Wrong interface specifications never lead to wrong proofs. They may only prevent a successful verification of a valid property. This is very important, because it allows the designer to simply "guess" interface specifications, while maintaining the reliability of a successful verification.

[7]This reminds for the situation for *while*-programs, where automatic verification depends on *loop invariants* that also need to be provided by the programmer.

Indeed, a way to obtain interface specifications is by using the property to be verified as interface specification. This is what Clarke et al. [CLM89] had in mind. However, their approach only exploits the alphabet of the property under consideration. A refined treatment of property constraints using our notion of interface specification is under investigation.

Finally, it should be mentioned that our method can easily be implemented. In fact, an implementation in the Edinburgh Concurrency Workbench [CPS89] and in the Aldebaran verification tool [Fer88] is planned.

Acknowledgements

We would like to thank Rance Cleaveland and Ernst–Rüdiger Olderog for helpful discussions.

References

[BKP84] H. Barringer, R. Kuiper and A. Pnueli. *Now you may compose Temporal Logic Specifications*, Symposium on Theory of Computing, 1984

[BFH90] A. Bouajjani, J.-C. Fernandez and N. Halbwachs. *Minimal Model Generation*, Workshop on Automatic Verification (CAV), 1990, LNCS 531

[BCG86] M.C. Browne, E.M. Clarke and O. Grumberg. *Reasoning about networks with many identical finite state processes*, in ACM Symposium on Principle of Distributed Computing, 1986

[CHPP87] P. Caspi, N. Halbwachs, N. Pilaud and J. Plaice. *LUSTRE, a declarative language for programming synchronous systems*, 14th POPL, Munich, 1987

[CES83] E.M. Clarke, E.A. Emerson and E. Sistla. *Automatic Verification of Finite State Concurrent Systems using Temporal Logic Specification: A Practical Approach*, POPL 1983

[CLM89] E.M. Clarke, D.E. Long and K.L. McMillan. *Compositional Model Checker*, LICS, 1989

[CPS89] R. Cleaveland, J. G. Parrow and B. Steffen. *The Concurrency Workbench*, Workshop on Automatic Verification Methods for Finite State Systems, Grenoble, France, 1989, LNCS 407

[CPSb89] R. Cleaveland, J. G. Parrow and B. Steffen. *A Semantics based Verification Tool for Finite State Systems*, in the proceedings of the Ninth International Symposium on Protocol Specification, Testing, and Verification; North Holland, 1989

[ClSte90] R. Cleaveland and B. Steffen. *When is "Partial" Complete? A Logic-Based Proof Technique using Partial Specifications*, LICS, 1990

[Fer88] J.-C. Fernandez. *Aldébaran: Un Système de Vérification par Réduction de Processus Communicants*, Ph.D. Thesis, Université de Grenoble, 1988

[FSS83] J.-C. Fernandez, J.-Ph. Schwartz and J. Sifakis. *An Example of Specification and Verification in Cesar*, 'The Analysis of Concurrent Systems', 1983, LNCS 207

[GS90] S. Graf and B. Steffen. "Using Interface Specifications for Compositional Reduction." To appear in *Proceedings of the Workshop on Computer-Aided Verification.*

[Jos87] B. Josko. *MCTL - An extension of CTL for modular verification of concurrent systems*, Workshop on Temporal Logic in Specification 1987, LNCS 398

[KuMcM89] R.P. Kurshan and K. McMillan. *A Structural Induction Theorem for Processes*, in ACM Symposium on Principles of Distributed Computing, 1989

[Kr89] H. Krumm. *Projections of the Reachability Graph and Environment Models, two approaches to facilitate the functional analysis of Systems of cooperating finite state machines*, Workshop on Automatic Verification of Finite State Systems, Grenoble 89, LNCS 407.

[KU] J. B. Kam and J. D. Ullman. *"Monotone Data Flow Analysis Frameworks"*. Acta Informatica 7, 309 - 317, 1977

[LaMi87] K. G. Larsen and R. Milner. *Verifying a Protocol Using Relativized Bisimulation*, ICALP 87, LNCS 267, Springer-Verlag, Berlin.

[LaTh88] K. G. Larsen and B. Thomsen. *Compositional Proofs by Partial Specification of Processes*, LICS, 1988

[LaXi90] K.G. Larsen and L. Xinxin. *Compositionality through an Operational Semantics of Contexts*, ICALP, LNCS 443, 1990

[Mi80] R. Milner. *A Calculus for Communicating Systems*, LNCS 92, 1980

[Mi89] R. Milner. *Communication and Concurrency*, Prentice Hall, 1989

[Old90] E.-R. Olderog. *Nets, Terms and Formulas: Three Views of Concurrent Processes*, Habilitationsschrift, Universität Kiel, to appear in Tracts in Theoretical Computer Science, Cambridge University Press

[Pnu85] A. Pnueli. *In Transition from Global to Modular Temporal Reasoning about Programs*, in Logics and Models for Concurrent Systems, NATO ASI Series F, Vol. 13, Springer Verlag

[ShGr90] G. Shurek and O. Grumberg. *The Modular Framework of Computer-aided Verification: Motivation, Solutions And Evaluation Criteria*, Workshop on Automatic Verification (CAV), 1990, LNCS 531

[StGr89] Z. Stadler and O. Grumberg. *Network Grammars, Communication Behaviours and Automatic Verification*, Workshop on Automatic Verification Methods for Finite State Systems, Grenoble, France, 1989, LNCS 407

[Ste89] B. Steffen. *Characteristic Formulae*, ICALP 1989, LNCS 372

[SI] B. Steffen and A. Ingólfsdóttir. "Characteristic Formulae for CCS with Divergence." Accepted for *Theoretical Computer Science.*

[Sti87] C. Stirling. *Modal Logics for Communicating Systems*, TCS 49, pp. 311-347, 1987

[StiWa89] C. Stirling and D. J. Walker. *Local Model Checking in the Modal Mu-Calculus*, CAAP 1989, LNCS 351

[Wa88] D.J. Walker. *Bisimulation and Divergence in CCS*, LICS 1988

[Win90] G. Winskel. *Compositional Checking of Validity on Finite State Processes*, Workshop on Theories of Communication, CONCUR, 1990

[WoLo89] P. Wolper and V. Lovinfosse. *Verifying Properties of Large sets of Processes with Network Invariants*, Workshop on Automatic Verification Methods for Finite State Systems, Grenoble, France, 1989, LNCS 407