Compositional Minimization of Finite State Systems[*]

Susanne Graf [♮] Bernhard Steffen [‡]

Abstract

In this paper we describe a method for the obtention of the minimal transition system, representing a communicating system given by a set of parallel processes, avoiding the complexity of the non minimal transition system. We consider minimization with respect to observational equivalence, but the method may be adapted to any other equivalence.

An interesting method to achieve this goal is to proceed by stepwise composition and minimization of the components of the system. However, if no precautions are taken, the intermediate state graphs generated by this method may contain a lot of transitions which are impossible in the whole context. We give here a variant of this method which allows to avoid these impossible transitions by taking into account at each composition step a guess of the interface behaviour of the context. This "interface specification" must be provided by the user. The method is based on a *reduction operator* for the composition of a subsystem with its interface specification, which is similar to the parallel operator but introduces *undefinedness* predicates whereever the interface "cuts off" a transition. The parallel operator is defined in a way that these undefinedness predicates disappear again in the full context if and only if the corresponding transition is in fact impossible in the whole system.

The *efficiency* of the method depends in fact on the accuracy with which the designer is able to approximate the possible sequences of the context, but its *correctness* does not. The proof of the correctness of the method is based on a preorder relation similar to the one defined by Walker.

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 geometrically 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,ShGr90, StGr89,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 observers point of view. Such abstractions often allow to reduce the state graph drastically by collapsing semantically equivalent states to a single state, which does not affect the observable behaviour. For example, the so obtained "minimization" of a complex communication protocol may be a simple buffer.

[*]This work has been partially supported by ESPRIT Basic Research Action 'Spec'

[♮]IMAG-LGI, BP 53X, F-38041 Grenoble

[‡]Department of Computer Science, University of Aarhus, DK-8000 Aarhus C

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 compositionally construct the minimal system representation and therefore to avoid the apparent complexity. 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 $(p_1 \|_{I_1} \cdots \|_{I_{n-1}} p_n)\langle L \rangle$. Here we assume the processes p_i already as to be given as transition systems, I_i to be *interface specifications*, namely supersets of the set of sequences that can be observed at the associated interfaces, $\|$ to represent parallel composition 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.
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 easily adapts to other equivalences as well (see Section 5.3).

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 $P_n \approx^d \mathcal{P}$, which is enough to guarantee successful verification of all ($\approx^d$- consistent) properties 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, 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 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.

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 bisimulation 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 systems components. Using $\langle L \rangle$ operations together with an elementary rule for distributing them over the parallel operator (see Proposition 3.3) our method covers this approach.

Krumm [Kr89] considers minimization on the fly and reduction with respect to an interface specification that must be provided by the system designer. Whereas the former is limited, because the complete state graph need to be traversed, the latter requires a separate proof of the correctness of the interface specification. This contrasts with our method, where such a proof is unnecessary.

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 allowing to define a strategy for (semi-)automatic proofs where the required user support is kept to a minimum.

The methods proposed in [WoLo89,StGr89,KuMcM89] are tailored to verify properties of classes of systems that are systematically built from large numbers of similar 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. The transition relations for the resulting processes are defined by:

1. $\dfrac{p \xrightarrow{a} p'}{p\langle L \rangle \xrightarrow{a} p'\langle L \rangle}\ a \in L$ 2. $\dfrac{p \xrightarrow{a} p'}{p\langle L \rangle \xrightarrow{\tau} p'\langle L \rangle}\ a \notin L$

3. $\dfrac{p \xrightarrow{a} p'}{p \| q \xrightarrow{a} p' \| q}\ (a \notin \mathcal{A}_q)$ 4. $\dfrac{q \xrightarrow{a} q'}{p \| q \xrightarrow{a} p \| q'}\ (a \notin \mathcal{A}_p)$

5. $\dfrac{p \xrightarrow{a} p' \quad q \xrightarrow{a} q'}{p \| q \xrightarrow{a} p' \| q'}\ a \neq \tau$

their undefinedness predicates by:

6. $\dfrac{p\!\uparrow a}{p\langle L \rangle\!\uparrow a}\ a \in L$ 7. $\dfrac{p\!\uparrow a}{p\langle L \rangle\!\uparrow \tau}\ a \notin L$

8. $\dfrac{p\!\uparrow a}{(p \| q)\!\uparrow a}\ (a \notin \mathcal{A}_q \text{ or } q \xrightarrow{a} q')$ 9. $\dfrac{q\!\uparrow a}{(p \| q)\!\uparrow a}\ (a \notin \mathcal{A}_p \text{ or } p \xrightarrow{a} p')$

10. $\dfrac{p\!\uparrow a \quad q\!\uparrow a}{(p \| q)\!\uparrow a}$

and their alphabets by $\mathcal{A}_{p\langle L\rangle} = \mathcal{A}_p \setminus L$ and $\mathcal{A}_{p\|q} = \mathcal{A}_p \cup \mathcal{A}_q$. Finally $S_{p\langle L\rangle} = \{p'\langle L\rangle | p' \in S_p\}$ and $S_{p\|q}$ is the set of pairs $p'\|q'$ that are reachable in $p\|q$.

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 and 9. The exact meaning of this definition will become clear in Section 5.2, where we introduce the reduction operator. We have:

Proposition 3.2 $\|$ *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].

Proposition 3.3 $\forall p, q\, \forall L.\ (p\|q)\langle L\rangle \;=\; (p\langle L\cup\mathcal{A}_q\rangle \,\|\, q\langle L\cup\mathcal{A}_p\rangle)\langle L\cap\mathcal{A}_p\cap\mathcal{A}_q\rangle$

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.

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. $p \overset{a}{\Rightarrow} p'$ *implies* $\exists q'.\ q \overset{a}{\Rightarrow} q' \wedge p'Rq'$
3. $q \overset{a}{\Rightarrow} q'$ *implies* $\exists p'.\ p \overset{a}{\Rightarrow} p' \wedge p'Rq'$

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.

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

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

4.3 The Specification - Implementation Relation

The following preorder 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. *$p \overset{a}{\Rightarrow} p'$ implies $\exists q'.\, q \overset{a}{\Rightarrow} q' \wedge p'Rq'$*
2. *$\neg p{\Uparrow}a$ implies $(\neg q{\Uparrow}a$ and $q \overset{a}{\Rightarrow} q'$ implies $\exists p'.\, p \overset{a}{\Rightarrow} p' \wedge p'Rq')$*

$\preceq$ 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 iff $p \preceq q$, i.e. in contrast to [ClSte90,Wa88] we do not require an implementation to be able to pass these guards. This modification enhences the practicality of preorders as specification-implementation relations. In fact, similar definitions of preorders already appeared in [Ste89,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 $p \preceq q$.*

Furthermore, on totally defined processes $\approx^d$ and $\preceq$ and $\approx$ all coincide. Finally we have the following monotonicity properties:

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 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. Denoting the language of a process p by $\mathcal{L}(p)$ we have:

Proposition 5.1 $\forall p, q.\ \mathcal{L}((p\|q)\langle \mathcal{A}_p \cap \mathcal{A}_q\rangle) = \mathcal{L}(p\langle \mathcal{A}_q\rangle) \cap \mathcal{L}(q\langle \mathcal{A}_p\rangle)$

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 5.2 *Given two processes p and q we define:*

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

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 5.4 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 wrt $\mathtt{I}$ is essentially the projection of $p\|\mathtt{I}$ onto its first component; we define $\Pi_{\mathtt{I}}(p) = ((S, \rightarrow, \mathcal{A} \cup \{\tau\}, \uparrow), p)$, where:

- $S = \{q \in S_p \mid \exists i' \in S_{\mathtt{I}}.\ q\|i' \in S_{p\|\mathtt{I}}\}$
- $\mathcal{A}_p = \mathcal{A}$
- $\forall q, q' \in S\ \forall a \in \mathcal{A} \cup \{\tau\}.\ q \xrightarrow{a} q'$ iff $\exists i, i' \in S_{\mathtt{I}}.\ q\|i \xrightarrow{a}_{p\|\mathtt{I}} q'\|i'$
- $\forall q \in S\ q\uparrow\tau$ iff $q\uparrow\tau$ in the transition system of p
- $\forall q \in S\ \forall a \in \mathcal{A}.\ q\uparrow a$ iff one of the following conditions holds:
  - $q\uparrow a$ in the transition system of p
  - $\exists q' \in S_p.\ q \xrightarrow{a}_p q'$ and $\neg\exists q' \in S.\ q \xrightarrow{a} q'$

The only difference between $\Pi_{\mathtt{I}}(p)$ and the projection of $p\|\mathtt{I}$ onto p are 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. Remember, 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 the corresponding state preempts the execution of an a-transition. This means that $\uparrow a$ are used as 'error' predicates, indicating in the full context where an interface specification is badly defined.

It is possible to show that $\Pi_{\mathtt{I}}(p)$ can be constructed in time proportional to the product of the number of transitions of p and $\mathtt{I}$. We have:

Proposition 5.3 $\forall p\, \forall \mathtt{I} \in \mathcal{I}(p).\ \Pi_{\mathtt{I}}(p) \preceq p$

Now we establish the promised independency of the reduction operator of the specific representation of the language specifying the interface.

Proposition 5.4

1. $\forall p\, \forall \mathtt{I}, \mathtt{I}' \in \mathcal{I}(p).\ \mathcal{L}(\mathtt{I}) = \mathcal{L}(\mathtt{I}')$ implies $\Pi_{\mathtt{I}}(p) = \Pi_{\mathtt{I}'}(p)$
2. $\forall p\, \forall \mathtt{I}, \mathtt{I}' \in \mathcal{I}(p).\ \mathcal{L}(\mathtt{I}) \subseteq \mathcal{L}(\mathtt{I}')$ implies $\Pi_{\mathtt{I}}(p) \preceq \Pi_{\mathtt{I}'}(p)$
3. $\forall p\, \forall \mathtt{I} \in \mathcal{I}(p).\ \mathcal{L}(p) \subseteq \mathcal{L}(\mathtt{I})$ implies $\Pi_{\mathtt{I}}(p) = p$

The correctness of our method is based on the following theorem:

Theorem 5.5 *Let p and q be processes and $\mathtt{I}$ an interface specification for p and q. Then $\mathtt{I} \in \mathcal{I}(p,q)$ implies that $p\|q$ is isomorphic to $\Pi_{\mathtt{I}}(p)\|q$, i.e. there exists a bijection $\iota : S_{p\|q} \rightarrow S_{\Pi_{\mathtt{I}}(p)\|q}$, such that:*

1. $\forall p', p'' \in S_{p\|q}\ \forall a \in \mathcal{A}_{p\|q} \cup \{\tau\}.\ p' \xrightarrow{a}_{p\|q} p''$ iff $\iota(p') \xrightarrow{a}_{\Pi_{\mathtt{I}}(p)\|q} \iota(p'')$
2. $\forall p' \in S_{p\|q}\ \forall a \in \mathcal{A}_{p\|q} \cup \{\tau\}.\ p'\uparrow a$ iff $\iota(p')\uparrow_{\Pi_{\mathtt{I}}(p)\|q}(a)$

This result illustrates the generality of the reduction operator. It can be used for the verification of all properties, which are preserved by isomorphism. In particular, we have the following corollary concerning the semantic equivalence we are focusing on here:

Corollary 5.6 $\forall q, p\, \forall \mathtt{I}.\ \mathtt{I} \in \mathcal{I}(p,q)$ implies $p\|q \approx^d \Pi_{\mathtt{I}}(p)\|q$

5.3 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[3]. 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. However, the *correctness* of the method does not depend on the correctness of these interface specifications. They are only used to "guide" the proof; and more precise interface specifications may allow more reductions. Thus, wrong interface specifications will never lead to wrong proofs. They may only prevent a successful verification of a valid statement.

The 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:

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

where 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 $\mathcal{P}$:

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

where the P_i are defined as follows:

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

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. Our method does not depend on a particular semantic equivalence. Other equivalences can be dealt with just by changing the minimization function accordingly (cf. Theorem 5.5). Of course, in order to prove the correctness of the method, this also requires to adapt the preorder definition.

Independently of the correctness of the interface specification, we obtain just by means of Propositions 4.4 and 5.3:

Proposition 5.7 *Let $1 \leq i \leq n$ and Q denote the parallel composition of all processes with index greater than i. Then we have: $(P_i \| Q)\langle L \rangle \preceq \mathcal{P}$,*

This is already enough to guarantee the correctness of our 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 Corollary 5.6:

[3]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.

Theorem 5.8 *For $1 \le j \le n$ let Q_j denote the parallel composition of all processes with index greater than j. Then we have for $1 \le i \le n$:*

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

The situation for totally defined overall systems is particularly simple, as can be inferred from:

Corollary 5.9 *Whenever P_n is totally defined, we have: $P_n \approx^d \mathcal{P}$.*

6 An Application

Finally, we demonstrate our method by means of a setup that is intended to ensure round robin access of n processes P_i to a common resource R. The idea is to pass a "token" via the communication channels tk_i in round robin manner 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 iteratively executes the sequence $tk_1, \cdots, tk_n$, i.e. it is enough to show for

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

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 I_i denote the exact interface specifications:

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

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 during stepwise minimization when exploiting the 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 by stepwise composition and minimization of components encounters transition systems that are even larger than the global state graph: e.g. when using this method in order to construct the minimal state graph of the above system for $n = 7$, the largest intermediate state graph that must be constructed has 2916 states and 9801 transitions.

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[4].

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 $\approx^d$. Even better, the choice of $\approx^d$ only affects the minimization step. Thus our method can be adapted to other semantic equivalences simply by modifying the minimization operator $\mathcal{M}$.

In this paper, we proposed a left to right strategy for the minimization of a highly parallel process. Of course, the correctness of our method does not depend on this particular choice.

The *effect* of our method 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 will 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.

Another way to obtain interface specifications is by exploiting the property we are going to verify. 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

[BFH90] A. Bouajjani, J.-C. Fernandez, N. Halbwachs. *Minimal Model Generation*, this volume

[CES83] E.M. Clarke, E.A. Emerson, 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, K.L. McMillan. *Compositional Model Checker*, LICS, 1989

[CPS89] R. Cleaveland, J. G. Parrow and B. Steffen. *The Concurrency Workbench*, Proceeding of the 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*, in Proceedings of LICS'90, 1990

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

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

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

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

[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, K. McMillan. *A Structural Induction Theorem for Processes*, in ACM Symposium on Principles of Distributes 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*, Proceedings of the Workshop on Automatic Verification of Finite State Systems, Grenoble 89, LNCS 407.

[LaTh88] Larsen, K.G., and B. Thomsen. *Compositional Proofs by Partial Specification of Processes*, in Proceedings LICS'88, 1988

[LaXi90] K.G. Larsen, L. Xinxin. *Compositionality through an Operational Semantics of Contexts*, in Proceedings ICALP'90, LNCS, 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, O. Grumberg. *The Modular Framework of Computer-aided Verification: Motivation, Solutions And Evaluation Criteria*, this volume

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

[Ste89] B. Steffen. *Characteristic Formulae*, in Proceedings ICALP 1989

[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*, in Proceedings CAAP 1989

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

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

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