UNIVERSITÄT PASSAU
Fakultät für Mathematik und Informatik

Compositional Minimization of Finite State Systems Using Interface Specifications

Susanne Graf
VERIMAG
Rue Lavoisier
F–38330 Monbonnot
France

Bernhard Steffen, Gerald Lüttgen
Fakultät für Mathematik und Informatik
Universität Passau
Innstraße 33
D–94032 Passau
Germany

MIP-9505
January 1995

Abstract

In this paper we present a method for the *compositional construction* of the *minimal transition system* that represents the semantics of a given distributed system. Our aim is to control the *state explosion* caused by the interleavings of actions of communicating parallel components by *reduction steps* that exploit *global* communication constraints given in terms of *interface specifications*. The *effect* of the method, which is developed for *bisimulation semantics* here, depends on the structure of the distributed system under consideration, and the *accuracy* of the interface specifications. However, its *correctness* does *not*: every "successful" construction is guaranteed to yield the desired minimal transition system, independent of the correctness of the interface specifications provided by the program designer.

Keywords

bisimulation, distributed system, interface specification, minimization, reduction operator, specification preorder, state explosion problem, transition system, verification.

Contents

1 Introduction

Many tools for the *automatic analysis or verification of finite state distributed systems* are based on the construction of the *global state graph* of the system under consideration (cf. [CES83, CPS89a, CPS89b, FSS83, Ste94]). Thus they often fail because of the *state explosion problem*: the state space of a distributed 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, CR94, CS90b, DGG93, Fer88, GL93, GP93, Jos87, KM89, Kru89, LSW94, LT88, LX90, Pnu90, SG89, SG90, Val93, Wal88, Win90, WL89]). In this paper we present a *method for the compositional minimization of finite state distributed 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 us 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 constructing the minimal system representation taking context information into account. 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 us to "cut off" these unreachable parts. As in [CS90b, Kru89, LT88, SG90, Wal88] 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, called $\mathcal{RM}$-*Method*,[1] is tailored for establishing P $\models$ *Spec*, i.e. whether P satisfies the specification or property *Spec*, when P is a system in standard concurrent form, i.e. P $= (p_1 \|_{I_1} \ldots \|_{I_{n-1}} p_n)\langle L \rangle$, which is annotated by interface specifications, and *Spec* is preserved by the semantic equivalence under consideration, i.e. $P \models \mathit{Spec} \Longleftrightarrow Q \models \mathit{Spec}$ if P and Q are semantically equivalent. To simplify the development of our theory, we assume that the processes p_i are already given as transition systems and that $\|$ represents the parallel composition operator, $\langle L \rangle$ is a window or hiding operator that abstracts from the activities considered as internal by transforming them into the unobservable action τ, and I_i are *interface specifications* between $R_i =_{df} (p_1 \| \ldots \| p_i)$ and $Q_i =_{df} (p_{i+1} \| \ldots \| p_n)$, namely supersets of the set of sequences that can be observed at the associated interfaces which are also represented by transition systems.

[1] $\mathcal{RM}$-Method means Reduction-Minimization-Method.

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 R_i, i.e. P_i is smaller than R_i with respect to the *specification preorder* $\preceq$. This is the key for proving the *correctness* of the $\mathcal{RM}$-Method.
2. P_n is semantically equivalent to the full system P, whenever the interface specifications are *correct.*[2] This guarantees the *completeness* of the $\mathcal{RM}$-Method.
3. P_i has the least number of states and transitions in its semantic equivalence class.

Subsequently, the validation of $\mathcal{P}_n \models$ *Spec* completes the proof as it implies P $\models$ *Spec*. In this paper, we are dealing with a refinement $\approx^d$ of *observational equivalence* [Mil80, Mil89]. However, the method also adapts to other equivalences.

An important factor in this approach are the interface specifications, which should be provided by the program designer. However, the *correctness* of the $\mathcal{RM}$-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, i.e. if $\mathcal{P}_n \models$ *Spec* is valid, then P satisfies *Spec*, too. Otherwise, if $\mathcal{P}_n \models$ *Spec* is *not* provable, then P may satisfy *Spec* or not. Thus, wrong interface specifications may only prevent a successful verification of a valid statement. The $\mathcal{RM}$-Method is *complete* in the following sense: If all the considered interface specifications are correct and if *Spec* is a $\approx^d$-consistent property, then $\mathcal{P}_n \approx^d$ P, $\mathcal{P}_n$ has the least number of states and transitions in its semantic equivalence class, and $\mathcal{P}_n \models$ *Spec* $\Longleftrightarrow$ P $\models$ *Spec*. Hence, if $\mathcal{P}_n \models$ *Spec* is *not* valid, then P does not satisfy *Spec*. It should be noted that the *total definedness* of $\mathcal{P}_n$ already implies the semantical equivalence of $\mathcal{P}_n$ and P. This criterion is sufficient for most practical applications.

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 P $\models \Phi$ depending on the syntax of the program P and the formula Φ. Unfortunately, the decomposition rules for processes involving the parallel operator are very restricted. Larsen and Xinxin [LX90] follow a similar line, however, their decomposition rules are based on an operational semantics of

[2]This does *not* mean that, in general, P_i and R_i are semantically equivalent for $1 \leq i \leq n-1$.

contexts rather than the syntax. In order to deal with the problems that arise from parallel compositions Pnueli [Pnu90] proposed a "conditional" inference system, where assertions of the form $\phi \mathrm{P} \psi$ can be derived, meaning that the program P satisfies the property ψ under the condition that its environment satisfies ϕ. This inference system has been used by Shurek and Grumberg in [SG90], 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 [Kru89]. 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. Other methods try to avoid the state explosion problem using preorders for verification [GP93, GW91, Val93] where unnecessary interleavings of actions are suppressed. In [LSW94] a constraint-oriented state-based proof methodology for concurrent software systems is presented which exploits compositionality and abstraction for the reduction of the (possibly infinite) verification problem under consideration. There, Modal Transition Systems are used for fine-granular, loose state-based specifications of constraints.

Halbwachs et al. [BFH90] proposed a method of the second category. It constructs directly a transition system minimized with respect to bisimulations by successive refinement of a single state. In this method 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 $\langle L \rangle$ operators together with an elementary rule for distributing them over the parallel operator (see Proposition 2.6) our method covers this approach. Larsen and Thomsen [LT88], and Walker [Wal88] 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 [BCG86, KM89, SG89, WL89] 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.

In practice, Binary Decision Diagrams are used to code state graphs for an interesting class of systems [Bry86]. These codings do not explode directly, but they may explode during verification. All mentioned techniques can be accompanied by *abstraction*. Parallel systems may be dramatically reduced by suppressing constraints that are irrelevant for the verification of the particular property under consideration [CC77, CGL92, CR94, DGG93, LGS$^+$92].

Structure of the Paper

The remainder of the paper is structured as follows. Section 2 presents the basic notions, and Section 3 the reduction operators our method, the $\mathcal{RM}$-method, is based upon. Subsequently, Section 4 develops the $\mathcal{RM}$-Method for the compositional minimization of finite state distributed systems, proves its correctness and completeness, and illustrates its power by means of an example, where the apparent exponential complexity is reduced to a linear algorithmic complexity. Finally, Section 5 draws our conclusions.

A completely self-contained development is given in [Lüt94].

2 General Notions

Our framework is based on processes (systems) as labelled transition systems extended by an undefinedness predicate on states. Processes can be structured by means of parallel composition and hiding thus allowing a hierarchical treatment. The introduction of undefinedness predicates naturally leads to a specification-implementation preorder between processes, which induces a slightly finer semantics on processes than observational equivalence [Mil80, Mil89]. This equivalence is captured by our technique, which is based on the language based notion of *interface specification* introduced subsequently.

2.1 Representation of Processes

We model distributed systems by *extended transition systems*, i.e. a transition system which is extended by an undefinedness predicate that plays an important role in the correctness proof of our $\mathcal{RM}$-Method.

Definition 2.1 (Extended Transition Systems)
An extended (finite state) transition system *is a quadruple* $(S, \mathcal{A} \cup \{\tau\}, \longrightarrow, \uparrow)$ *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. *$\longrightarrow \subseteq (S \times \mathcal{A} \cup \{\tau\} \times S)$ is a* transition relation*, and*
4. *$\uparrow \subseteq S \times 2^{\mathcal{A}\cup\{\tau\}}$ is a predicate expressing* guarded undefinedness.[3]

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. We write $p \xrightarrow{a}$ for $\exists q \in S.\, p \xrightarrow{a} q$. Finally, $p \uparrow a$

[3] 2^M denotes the power set of the set M.

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 preorder which provides the framework for proving our method correct.

Processes are rooted extended transition systems, i.e. pairs consisting of an extended transition system and a designated start state.

Definition 2.2 (Processes)
Let $T = (S, \mathcal{A} \cup \{\tau\}, \longrightarrow, \uparrow)$ be an extended transition system. A process *is a pair $((S_p, \mathcal{A}_p \cup \{\tau\}, \longrightarrow_p, \uparrow_p), p)$ for a state $p \in S$ where*

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

p is called start state *of the process. The set of all processes is denoted by $\mathcal{P}$.*

In future, obvious indices will be dropped, and we will shortly write p for $((S_p, \mathcal{A}_p \cup \{\tau\}, \longrightarrow_p, \uparrow_p), p)$. The following property characterizes the subset of "standard" transition systems: A process is *totally defined* if its undefinedness predicate $\uparrow$ is empty. Otherwise it is called *partial*. Moreover, if $p, q \in \mathcal{P}$ are identical up to renamings of states, we call p and q *isomorphic*, in signs $p \cong q$. If no confusion arises with syntactic equality, we will simply write $p = q$. A process p is called *deterministic* if $\forall q \in S_p, a \in \mathcal{A}_p.\ |\{q' | q \xrightarrow{a} q'\}| \leq 1$. Otherwise p is called *nondeterministic*.

As usually, processes can be characterized by their language. We will need this characterization when dealing with interface specifications. The following definition of the language of processes uses the weak transition relation $\Longrightarrow$ and undefinedness predicate $\Uparrow$ presented in Definition 2.8, which reflect the relation $\longrightarrow$ and the predicate $\uparrow$ of the view of an observer who cannot see the internal action τ.

Definition 2.3 (Language of Processes)
The language, *$\mathcal{L}(p)$, of a partially defined process p is defined as the least fixed point 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

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

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 $\mathcal{L}_a$.

The well-definedness of the above definition follows from elementary fixed point theory. Note that this definition is standard for totally defined processes. The language of an a-undefined process includes any sequence of actions starting with a and the language of an ϵ-undefined process is $\mathcal{A}^*$ which reflects our intuition that the language of an undefined state is unconstraint. Therefore, we have to make the worst case assumption that the language of an undefined process could be *all* possible sequences of actions.

2.2 Parallel Composition and Hiding

We now introduce a binary parallel operator $\|$ and unary hiding or window operators $\langle L\rangle$ on processes, where L is the set of actions remaining visible. Intuitively, $p\|q$ is the parallel composition of the processes p and q with synchronization of the actions common to both of their alphabets and interleaving of the others (like in CSP [Hoa85]), and $p\langle L\rangle$ is the process in which only the actions in L are observable.

Definition 2.4 (Operational Semantics)
Let $p = ((S_p, \mathcal{A}_p \cup \{\tau\}, \longrightarrow_p, \uparrow_p), p)$, $q = ((S_q, \mathcal{A}_q \cup \{\tau\}, \longrightarrow_q, \uparrow_q), q)$ be in $\mathcal{P}$, let $p', p'' \in S_p$, $q', q'' \in S_q$, and let L be a set of visible actions. We define the alphabets of the processes $p\langle L\rangle$ and $p\|q$ by $\mathcal{A}_{p\langle L\rangle} =_{df} \mathcal{A}_p \cap L$ and $\mathcal{A}_{p\|q} =_{df} \mathcal{A}_p \cup \mathcal{A}_q$, respectively, 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 $p\|q$, respectively, according to the following transition relations defined in Plotkin style notation*:*

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

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

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

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

6. $\dfrac{p' \uparrow_p a}{p'\langle L\rangle \uparrow_{p\langle L\rangle} a} \quad a \in L$ 7. $\dfrac{p' \uparrow_p a}{p'\langle L\rangle \uparrow_{p\langle L\rangle} \tau} \quad a \notin L$

8. $\dfrac{p' \uparrow_p a}{(p'\|q') \uparrow_{p\|q} a} \quad a \notin \mathcal{A}_q$ 9. $\dfrac{p' \uparrow_p a}{(p'\|q') \uparrow_{p\|q} a} \quad q' \xrightarrow{a}_q q''$

10. $\dfrac{q' \uparrow_q a}{(p'\|q') \uparrow_{p\|q} a} \quad a \notin \mathcal{A}_p$ 11. $\dfrac{q' \uparrow_q a}{(p'\|q') \uparrow_{p\|q} a} \quad p' \xrightarrow{a}_p p''$

12. $\dfrac{p' \uparrow_p a \quad q' \uparrow_q a}{(p'\|q') \uparrow_{p\|q} a}$.

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. The exact meaning of this definition will become clear in Section 3, where we introduce *reduction operators*. We may immediately conclude from Definition 2.4:

Proposition 2.5 (Associativity & Commutativity)
The parallel operator $\|$ *is* associative *and* commutative *in the following sense:*

1. $\forall p, q, r \in \mathcal{P}.\ (p\|q)\|r \cong p\|(q\|r)$
2. $\forall p, q \in \mathcal{P}.\ p\|q \cong q\|p$

Thus processes of the form $(p_1\| \dots \|p_n)\langle L \rangle$ are well-defined. Our method will concentrate on this form which is called *standard concurrent form* in CCS [Mil80, Mil89].

Usually, the following correspondence between the parallel operator and the window operators is central.

Proposition 2.6 (Window Operator Law)
Let $p, q \in \mathcal{P}$ *and let* L, L' *be sets of visible actions satisfying* $L' \supseteq L \cup (\mathcal{A}_p \cap \mathcal{A}_q)$. *Then*

$$(p\|q)\langle L \rangle \cong (p\langle L' \rangle \| q)\langle L \rangle$$

This proposition allows us to localize global hiding informations. In fact, this localization is the essence of the construction of the 'interface processes' in [CLM89]. The proof of the proposition is done by induction similar to the proof of Theorem 3.8 including a case analysis according to Definition 2.4 in the induction step. It needs a tedious case analysis occupying more than ten pages [Lüt94].

We finish this section by presenting a simple example, which will accompany the development of our method.

Example 2.7 *Our example system System* $=_{df} (P_1\|B\|P_2)\langle\{tk1, tk2\}\rangle$, *presented in Figure 1, consists of three processes* P_1, B, *and* P_2 *with alphabets* $\mathcal{A}_{P_1} = \{tk1, tk2, rb1, sb1\}$, $\mathcal{A}_B = \{rb1, sb1, rb2, sb2\}$, *and* $\mathcal{A}_{P_2} = \{tk1, tk2, rb2, sb2\}$, *respectively.* I_1 *and* I_2 *indicate interface specifications which are presented and explained in Section 2.4. Process* B *models a buffer which is used by the processes* P_1 *and* P_2 *to exchange data, i.e.* P_1 *reads data from and sends data to* P_2 *via* B *and vice versa. To guarantee mutual exclusion of the "shared" buffer a token is passed through the channels* tk_1 *and* tk_2 *between* P_1 *and* P_2. *If* P_i *possesses the token, it may read some data from* B *via* rbi *and write some data to* B *via* sbi. *The exact definition of* P_1, B, *and* P_2 *is given in Figure 2 at the top left corner, at the bottom, and at the top right corner, respectively. The "shaded" states represent the start states of the processes.*

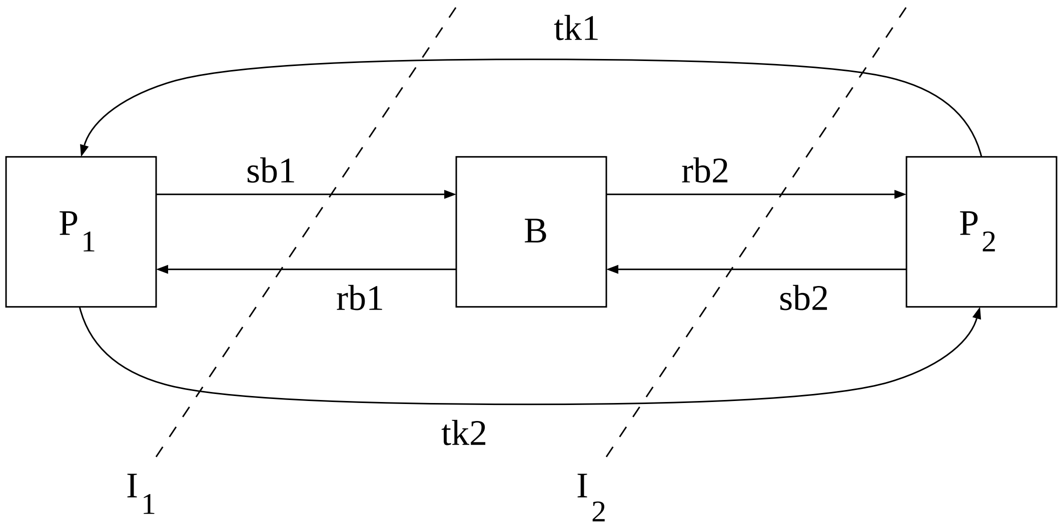


Figure 1: Communication Diagram of the Example System

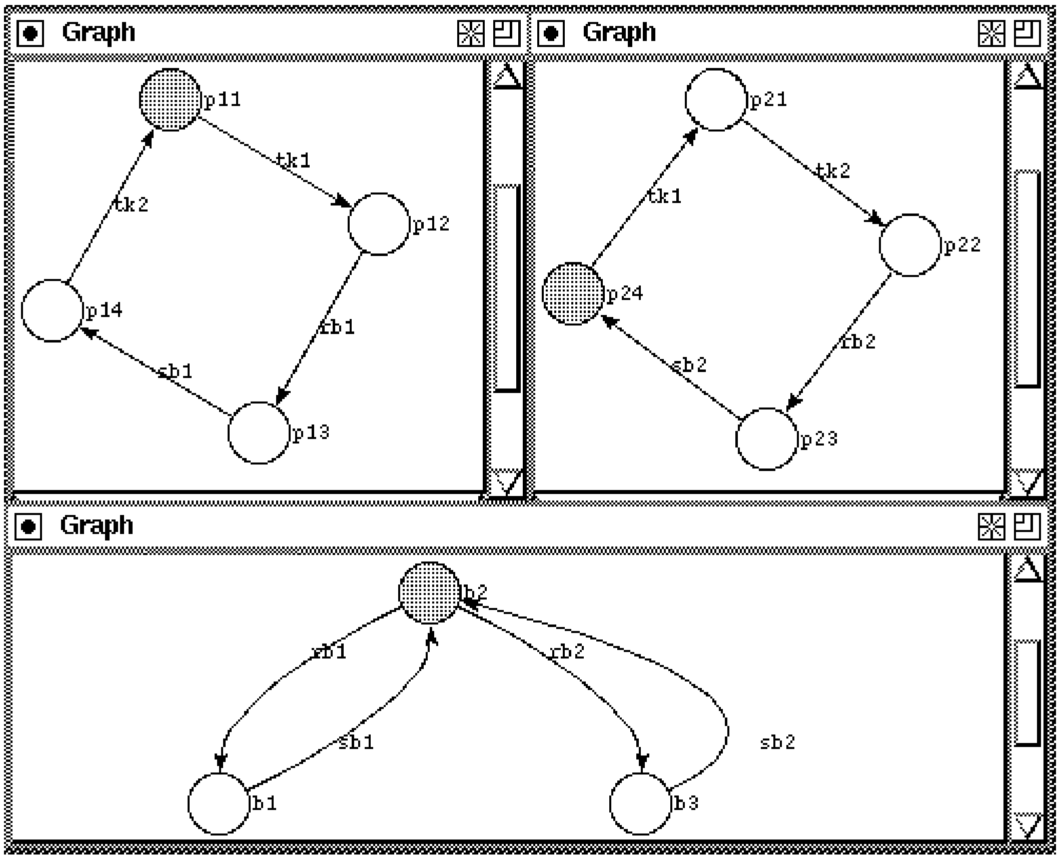


Figure 2: Definition of P_1, B, and P_2

2.3 Semantic Equivalence and Preorder

In this section, we define a semantics of extended labelled transition systems in terms of observational equivalence (cf. [Mil80]) 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 $\mathcal{RM}$-Method.

The $\longrightarrow$ 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 $\Longrightarrow$ and the *weak* undefinedness predicate $\Uparrow$ as usual.

Definition 2.8 (Weak Transition Relation and Undefinedness)
Let $\longrightarrow \subseteq S \times \mathcal{A} \cup \{\tau\} \times S$ be the transition relation and $\uparrow \subseteq S \times 2^{\mathcal{A} \cup \{\tau\}}$ be the undefinedness predicate of an extended transition system $(S, \mathcal{A}, \longrightarrow, \uparrow)$. The weak transition relation *$\Longrightarrow \subseteq S \times \mathcal{A} \times S$ and the* weak undefinedness predicate *$\Uparrow \subseteq S \times 2^{\mathcal{A}}$ are defined as the least relation satisfying for all $p, q \in S$ and all $a \in \mathcal{A}$:*

1. *$p \xrightarrow{\tau}{}^{*} \xrightarrow{a} \xrightarrow{\tau}{}^{*} q$ implies $p \overset{a}{\Longrightarrow} q$,*
2. *$p \xrightarrow{\tau}{}^{*} q$ implies $p \overset{\epsilon}{\Longrightarrow} q$,*
3. *$q \uparrow a$ and $p \overset{\epsilon}{\Longrightarrow} q$ implies $p \Uparrow a$,*
4. *$q \uparrow \tau$ and $p \overset{\epsilon}{\Longrightarrow} q$ implies $p \Uparrow \epsilon$,*
5. *$q \Uparrow \epsilon$ and $p \overset{a}{\Longrightarrow} q$ implies $p \Uparrow a$, and*
6. *$p \Uparrow \epsilon$ implies $p \Uparrow a$*

As already mentioned in Section 1, the minimization of transition systems is based on the fact that many computations are irrelevant from the observer's point of view. Our notion of semantics, which is defined by means of the following equivalence relation, reflects this intuition.[4]

Definition 2.9 (Semantic Equivalence)
Let $\longrightarrow \subseteq S \times \mathcal{A} \cup \{\tau\} \times S$ be the transition relation and $\uparrow \subseteq S \times 2^{\mathcal{A} \cup \{\tau\}}$ be the undefinedness predicate of an extended transition system $(S, \mathcal{A}, \longrightarrow, \uparrow)$. Then $\approx^d$ is the union of all relations $R \subseteq S \times S$ satisfying that $(p, q) \in R$ implies for all $a \in \mathcal{A}$:

1. *$p \Uparrow a$ if and only if $q \Uparrow a$,*
2. *$p \overset{a}{\Longrightarrow} p'$ implies $\exists q'.\, q \overset{a}{\Longrightarrow} q' \wedge (p', q') \in R$, and*
3. *$q \overset{a}{\Longrightarrow} q'$ implies $\exists p'.\, p \overset{a}{\Longrightarrow} p' \wedge (p', q') \in R$.*

This definition implies that only those processes could be equivalent which have the same alphabet, i.e. the same sychronisation potential. Note that $\approx^d$ coincides with the well-known observational equivalence $\approx$ (cf. [Mil80, Mil89]) if the first of the three defining requirements is dropped. Especially, isomorphic or syntactically identical processes are $\approx^d$-equivalent.

[4]A similar definition has been given in [CS90b].

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 $\mathcal{RM}$-Method.

Definition 2.10 (Specification Preorder)
Let $\longrightarrow \subseteq S \times \mathcal{A} \cup \{\tau\} \times S$ be the transition relation and $\uparrow \subseteq S \times 2^{\mathcal{A} \cup \{\tau\}}$ be the undefinedness predicate of an extended transition system $(S, \mathcal{A}, \longrightarrow, \uparrow)$. The specification preorder $\preceq$ is the union of all relations $R \subseteq S \times S$ satisfying $(p,q) \in R$ implies for all $a \in \mathcal{A}$ with $\neg(p \Uparrow a)$:

1. $\neg(q \Uparrow a)$,
2. $p \overset{a}{\Longrightarrow} p'$ *implies* $\exists q'. \, q \overset{a}{\Longrightarrow} q' \wedge (p', q') \in R$, *and*
3. $q \overset{a}{\Longrightarrow} q'$ *implies* $\exists p'. \, p \overset{a}{\Longrightarrow} p' \wedge (p', q') \in R$.

$\preceq$ is a variant of the divergence preorder (cf. [Wal88]) 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 [CS90b, Wal88] we do not require an implementation of an a-undefined process to possess any a-transition. This modification enhances the practicality of the preorder as specification-implementation relation. A more detailed discussion can be found in [CS90a].

Observational equivalence $\approx$ and our specification-preorder $\preceq$ induce slightly different semantics on processes. However, by definitions of $\approx^d$, $\approx$ and $\preceq$ we have that $\approx^d$ is a refinement of all of them.

Proposition 2.11 *For all processes $p, q \in \mathcal{P}$ we have $p \approx^d q$ implies $p \approx q$ and $p \preceq q$, and for totally defined processes $\approx^d$, $\preceq$ and $\approx$ coincide.*

Moreover, it can be proven in the usual way that both $\|$ and $\langle L \rangle$ preserve $\preceq$ and $\approx^d$ which is of particular importance for our minimization method.

Proposition 2.12 (Compositionality)
For all processes $p, q, r \in \mathcal{P}$ and all sets L of visible actions we have:

1. $p \preceq q$ *implies* $p \| r \preceq q \| r$,
2. $p \approx^d q$ *implies* $p \| r \approx^d q \| r$,
3. $p \preceq q$ *implies* $p\langle L \rangle \preceq q\langle L \rangle$, *and*
4. $p \approx^d q$ *implies* $p\langle L \rangle \approx^d q\langle L \rangle$.

The relationship between the notions *preorder*, *semantic equivalence* and *languages* is characterized by the following lemma.

Lemma 2.13 *For all processes $p, q \in \mathcal{P}$ we have:*

1. $p \preceq q$ *implies* $\mathcal{L}(p) \supseteq \mathcal{L}(q)$, *and*
2. $p \approx^d q$ *implies* $\mathcal{L}(p) = \mathcal{L}(q)$.

The proof of Part (1) is a consequence of the Definitions 2.10 and 2.3, whereas Part (2) is an immediate consequence of Part (1) and Proposition 2.11.

The $\mathcal{RM}$-Method presented in Section 4 works for every equivalence relation and every preorder satisfying Propositions 2.11, 2.12 and Lemma 2.13.

2.4 Interface Specifications

In this section we introduce our notion of *interface specification* together with a notion of *correctness*, which guarantees the success of the $\mathcal{RM}$-Method. These notions concentrate on the set of observable sequences that may pass the interface. Thus the *exact* specification of the interface between processes 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.

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 2.14 (Interface Specifications)
Given two processes $p, q \in \mathcal{P}$ we define:

1. *A totally defined process I is an* interface specification for p *iff $\mathcal{A}_I \subseteq \mathcal{A}_p$ and $\tau \notin \mathcal{A}_I$. It is an* interface specification for p and q *iff $\mathcal{A}_I = \mathcal{A}_p \cap \mathcal{A}_q$ and $\tau \notin \mathcal{A}_I$.*
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 is denoted by $\mathcal{I}$, the set of all interface specifications for p *by $\mathcal{I}(p)$, and the set of all* correct *interface specifications for p and q by $\mathcal{I}(p, q)$.*

Corollary 3.20 will show that these language-based definitions are adequate for our purpose. The following example illustrates the intuition-guided way of deriving interface specifications.

Example 2.15 (Interface Specifications for the Example System)
An interface specification I_1 for the system of Example 2.7 can be constructed according to the following intuition: process P_1 waits for the token passed via $tk1$ before it reads data from and writes data to B via $rb1$ and $sb1$, respectively. Subsequently, P_1 passes the token to P_2 via $tk2$ and waits until it receives the token again.

This intuition would already result in an exact interface specification for P_1 and $B||P_2$, which is identical to the process P_1 itself. Since a correct interface specification may describe a superset of the exact interface language (cf. Definition 2.14), the definition of I_1 given on the left in Figure 3 is correct.

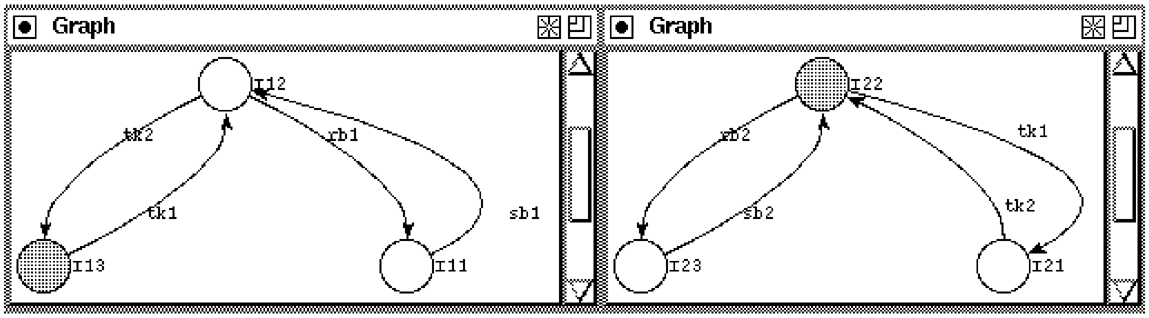


Figure 3: Interface Specifications I_1 and I_2

The same argument shows the correctness of the definition of I_2 given on the right in Figure 3.

The languages of I_1 and I_2 result from the following equation systems (cf. Definition 2.3).

$$\begin{array}{lcl}
\mathcal{L}(I_1) & = & \mathcal{L}(I_{13}) = tk1 \cdot \mathcal{L}(I_{12}) \cup \{\epsilon\} \\
\mathcal{L}(I_{12}) & = & rb1 \cdot \mathcal{L}(I_{11}) \cup tk2 \cdot \mathcal{L}(I_{13}) \cup \{\epsilon\} \\
\mathcal{L}(I_{11}) & = & sb1 \cdot \mathcal{L}(I_{12}) \cup \{\epsilon\} \\
\\
\mathcal{L}(I_2) & = & \mathcal{L}(I_{22}) = tk1 \cdot tk2 \cdot \mathcal{L}(I_{22}) \cup rb2 \cdot sb2 \cdot \mathcal{L}(I_{22}) \cup \{tk1 \cdot \epsilon\} \cup \{rb2 \cdot \epsilon\} \cup \{\epsilon\}
\end{array}$$

Note, however, that our method does not require to compute $\mathcal{L}(I_1)$ and $\mathcal{L}(I_2)$.

The following properties of interface specifications are important.

Lemma 2.16 (Properties of Interface Specifications)
For all processes $p, p', q \in \mathcal{P}$ we have:

1. $p \preceq p'$ *implies* $\mathcal{I}(p,q) \subseteq \mathcal{I}(p',q)$
2. $p \approx^d p'$ *implies* $\mathcal{I}(p,q) = \mathcal{I}(p',q)$

Proof:

Let $p, p', q \in \mathcal{P}$ satisfying $p \preceq p'$. As $\mathcal{A}_p = \mathcal{A}_{p'}$, we have by Proposition 2.12:

$$(p\|q)\langle \mathcal{A}_p \cap \mathcal{A}_q \rangle \preceq (p'\|q)\langle \mathcal{A}'_p \cap \mathcal{A}_q \rangle$$

and by Lemma 2.13(1):

$$(*) \qquad \mathcal{L}((p\|q)\langle \mathcal{A}_p \cap \mathcal{A}_q \rangle) \supseteq \mathcal{L}((p'\|q)\langle \mathcal{A}'_p \cap \mathcal{A}_q \rangle).$$

Now, let $I \in \mathcal{I}(p,q)$ be arbitrary. Then we conclude the first part as follows:

$$\begin{array}{lcl} & & I \in \mathcal{I}(p,q) \\ (\text{Definition 2.14}) & \Rightarrow & \mathcal{L}(I) \supseteq \mathcal{L}((p\|q)\langle \mathcal{A}_p \cap \mathcal{A}_q \rangle) \\ (*) & \Rightarrow & \mathcal{L}(I) \supseteq \mathcal{L}((p'\|q)\langle \mathcal{A}_{p'} \cap \mathcal{A}_q \rangle) \\ (\text{Definition 2.14}) & \Rightarrow & I \in \mathcal{I}(p',q). \end{array}$$

The second part is a consequence of Proposition 2.11 and part one.

□

The following proposition is particularly important for the completeness proof of the $\mathcal{RM}$-Method.

Proposition 2.17 *For all processes $p, q \in \mathcal{P}$ and all sets L of visible actions we have: $\mathcal{I}(p,q) = \mathcal{I}(p\langle(\mathcal{A}_p \cap \mathcal{A}_q) \cup L\rangle, q)$.*

Proof: Let $p, q \in \mathcal{P}$ and L be a set of visible actions. Then we have:

$$\begin{array}{lcl} & & I \in \mathcal{I}(p,q) \\ (\text{Def. 2.14}) & \Longleftrightarrow & \mathcal{L}(I) \supseteq \mathcal{L}((p\|q)\langle \mathcal{A}_p \cap \mathcal{A}_q \rangle) \\ (\text{Prop. 2.6 \& Lemma 2.13(2)}) & \Longleftrightarrow & \mathcal{L}(I) \supseteq \mathcal{L}((p\langle(\mathcal{A}_p \cap \mathcal{A}_q) \cup L\rangle\|q)\langle \mathcal{A}_p \cap \mathcal{A}_q \rangle) \\ (\text{Def. 2.14}) & \Longleftrightarrow & I \in \mathcal{I}(p\langle(\mathcal{A}_p \cap \mathcal{A}_q) \cup L\rangle, q). \end{array}$$

□

3 Reduction Operators

In this section we propose a general notion of *reduction operators*, and a special instance of it ($\overline{\Pi}$), which is suitable for our purposes (cf. Section 3.1). $\overline{\Pi}$ will then be analysed from two different views, the theoretical view (cf. Section 3.2) and the algorithmic view (cf. Section 3.3).

3.1 General Definitions and Properties

Reduction operators are characterized by three properties:

Definition 3.1 (Reduction Operators)
A partial mapping $\Pi : \mathcal{I} \times \mathcal{P} \longrightarrow \mathcal{P}$ *is called* reduction operator *if*

(i) $\forall p \in \mathcal{P}, I \in \mathcal{I}(p).\ \Pi(I,p) \preceq p$ (Correctness)

(ii) $\forall p,q \in \mathcal{P}, I \in \mathcal{I}(p,q).\ \Pi(I,p) \| q \approx^d p \| q$ (Context Preservation)

(iii) $\forall p \in \mathcal{P}, I \in \mathcal{I}(p).\ \mid S_{\Pi(I,p)} \mid \leq \mid S_p \mid$ *and* $|\longrightarrow_{\Pi(I,p)}| \leq |\longrightarrow_p|$ (Reduction)[5]

In the following we will often write $\Pi_I(p)$ *instead of* $\Pi(I,p)$*.*

The intuition behind this definition is the following: A reduction operator Π should eliminate those states and transitions of a process $p \in \mathcal{P}$ which are not reachable in each global context satisfying the interface specification $I \in \mathcal{I}(p)$. This 'algorithmic' intuition guarantees the first two conditions, which are essential for a sensible notion of reduction operator: the first condition is a correctness requirement. The reduction always yields a process which behaves as p on its defined part. The second condition guarantees that the reduction does not affect the behaviour of p in a context satisfying the interface specification. Finally, the third condition reflects the primary intuition of reduction: the number of states and transitions should be reduced. This is by no means guaranteed by a decrease in the preorder!

We obtain the following technical result.

Proposition 3.2
Let Π *be a reduction operator. Then we have for all* $p, p', q \in \mathcal{P}$ *and* $I \in \mathcal{I}(p,q)$*:*

$$p \approx^d p' \text{ implies } \Pi_I(p) \| q \approx^d \Pi_I(p') \| q$$

Proof: Let Π be a reduction operator, $p, p', q \in \mathcal{P}$ and $I \in \mathcal{I}(p,q)$. Then we may conclude:

$$\begin{array}{lcl} & & \Pi_I(p) \| q \\ \text{(Def. 3.1 (ii))} & \approx^d & p \| q \\ (p \approx^d p' \text{ \& Prop. 2.12}) & \approx^d & p' \| q \\ \text{(Lemma 2.13(2) \& Def. 3.1 (i))} & \approx^d & \Pi_I(p') \| q. \end{array}$$

□

[5] $|M|$ denotes the cardinality of the set M.

As we will see in Section 3.2, the following operator $\overline{\Pi}$ satisfies the conditions of Definition 3.1.

Definition 3.3 (The Reduction Operator $\overline{\Pi}$)
Let $p = ((S_p, \mathcal{A}_p \cup \{\tau\}, \longrightarrow_p, \uparrow_p), p) \in \mathcal{P}$ and $I \in \mathcal{I}(p)$. Then $\overline{\Pi} : \mathcal{I} \times \mathcal{P} \longrightarrow \mathcal{P}$ is defined by

$$(I, p) \longmapsto \overline{\Pi}(I, p) =_{df} \overline{\Pi}_I(p) =_{df} ((S, \mathcal{A} \cup \{\tau\}, \longrightarrow, \uparrow), p)$$

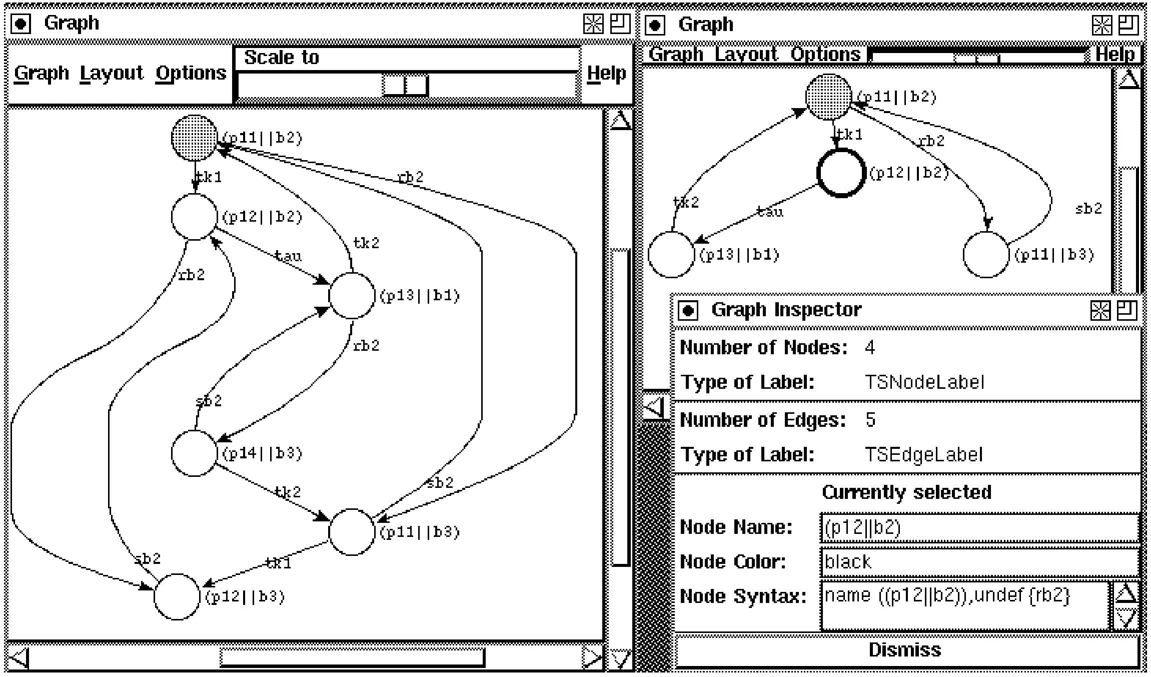
where

1. $S = \{q \in S_p | \exists i \in S_I.\, q \| i \in S_{p\|I}\}$,
2. $\mathcal{A} = \mathcal{A}_p$,
3. $\forall q, q' \in S, a \in \mathcal{A} \cup \{\tau\}.\, q \xrightarrow{a} q'$ *iff* $\exists i, i' \in S_I.\, q \| i \xrightarrow{a}_{p\|I} q' \| i'$,[6]
4. $\forall q \in S.\, q \uparrow \tau$ *iff* $q \uparrow_p \tau$, *and*
5. $\forall q \in S, a \in \mathcal{A}.\, q \uparrow a$ *iff*
   (a) $q \uparrow_p a$ *or*
   (b) $\exists q' \in S_p.\, q \xrightarrow{a}_p q'$ *and* $\not\exists q' \in S.\, q \xrightarrow{a} q'$.

The only difference between $\overline{\Pi}(I, p)$ and the projection of $p \| I$ onto p concerns the undefinedness predicates: $\Pi(I, p)$ inherits all undefinedness predicates from p, and new ones are introduced where transitions of p have been cut off by I. The point of the reduction operator is that for correct interface specifications this second kind of undefinedness disappears again in the full context $\overline{\Pi}(I, p) \| q$. This holds, because if an a-transition of p has been replaced by $\uparrow a$, this predicate disappears again in $\overline{\Pi}(I, p) \| q$ exactly if q, in its corresponding state, preempts the execution of an a-transition. Thus the presence of an $\uparrow a$ in $\overline{\Pi}(I, p) \| q$ indicates a fault in the interface specification, whenever p and q are totally defined processes. Note that it is possible that $\overline{\Pi}(I, p) \| q$ is totally defined, although I is *not* correct for p and q. This is the case if the incorrect parts of I need not be considered for the reduction.

Example 3.4 *Consider process p presented on the left in Figure 4 with alphabet $\mathcal{A}_p = \{tk1, tk2, rb2, sb2\}$ and the interface specification I_2 defined in Example 2.15. In order to determine $\overline{\Pi}_{I_2}(p)$ we first consider the projection of $p \| I_2$ onto p (Figure 4, right). Following Definition 3.3, $\overline{\Pi}_{I_2}(p)$ can now be derived by inserting some additional undefinednesses indicating a transition of p which is preempted by the interface. These are already computed by our system (implemented within the* META–Frame *environment [MCS95]) and can be revealed by the Graph Inspector: the field* node syntax *in Figure 4 shows that the highlighted state $(p12 \| b2)$ has an $rb2$-undefinedess. A further investigation would reveal the $tk1$-undefinedness of $(p11 \| b3)$ and the $rb2$-undefinedness of $(p13 \| b1)$.*

[6] This already implies that $q \| i$ is reachable in $p \| I$.



Number of Nodes: 4
Type of Label: TSNodeLabel
Number of Edges: 5
Type of Label: TSEdgeLabel
Node Name: (p12||b2)
Node Color: black
Node Syntax: name ((p12||b2)),undef {rb2}

Figure 4: An Example Reduction

3.2 Theoretical View

In this section we establish that $\overline{\Pi}$ is indeed a reduction operator. The correctness property (cf. Definition 3.1(i)) is straightforward:

Lemma 3.5 $\forall p \in \mathcal{P}, I \in \mathcal{I}(p).\ \overline{\Pi}_I(p) \preceq p.$

The remaining properties require two lemmata. The first lemma, which intuitively states that no "new" states and transitions are inserted, is a consequence of Definition 3.3 (iii) and Definition 2.4 (3) and (5).

Lemma 3.6 *Let $p \in \mathcal{P}$ and $I \in \mathcal{I}(p)$ be arbitrary. Then we have:*

$$\forall p', p'' \in S_{\overline{\Pi}(I,p)}, a \in \mathcal{A}_p \cup \{\tau\}.\ p' \xrightarrow{a}_{\overline{\Pi}(I,p)} p'' \text{ implies } p' \xrightarrow{a}_p p''$$

The second lemma guarantees that $\overline{\Pi}$ does not cut off too many states or transitions. It requires a more involved argument.

Lemma 3.7 *Let $p, q \in \mathcal{P}$, $I \in \mathcal{I}(p,q)$, $a \in \mathcal{A}_p \cup \{\tau, \epsilon\}$ and $p||q \longrightarrow^*_{p||q} p'||q' \xrightarrow{a}_{p||q} p''||q''$. Then we have:*

1. *$\exists I'' \in S_I.\ p''||I''$ is reachable in $p||I$.*

2. $p' \xrightarrow{a}_{\Pi(I,p)} p''$.

Proof: Because of Corollary 3.20 which is the central result of Section 3.4, we may assume that I is a deterministic interface specification.

We prove Lemma 3.7 by induction on n where n is the length of the path $p\|q \longrightarrow^n_{p\|q} p''\|q''$.

Base Case: $(n = 0)$

Here we have $a = \epsilon$ and $p\|q = p'\|q' = p''\|q''$, i.e. $p = p' = p''$ and $q = q' = q''$. Choose $I'' =_{df} I$ and hence that $p''\|I'' = p\|I$ is reachable in $p\|I$, such that (1) holds. Part (2) is trivial because $p' \xrightarrow{\epsilon}_{\Pi(I,p)} p''$.

Induction step: $(n \longrightarrow n{+}1)$

Here, we have $p\|q \longrightarrow^n_{p\|q} p'\|q' \xrightarrow{a}_{p\|q} p''\|q''$.

By induction hypothesis it exists $I' \in S_I$ satisfying:

$$(*) \qquad p'\|I' \text{ is reachable in } p\|I.$$

The application of $\langle \mathcal{A}_p \cap \mathcal{A}_q \rangle$ yields:

$$(p\|q)\langle \mathcal{A}_p \cap \mathcal{A}_q \rangle \longrightarrow^n_{p\|q} (p'\|q')\langle \mathcal{A}_p \cap \mathcal{A}_q \rangle \xrightarrow{b}_{p\|q} (p''\|q'')\langle \mathcal{A}_p \cap \mathcal{A}_q \rangle$$

where $b = \begin{cases} a & \text{if } a \in \mathcal{A}_q \\ \tau & \text{otherwise} \end{cases}$. Let $b' = \begin{cases} b & \text{if } b \neq \tau \\ \epsilon & \text{otherwise} \end{cases}$.

By the induction hypothesis, the premise $\mathcal{L}((p\|q)\langle \mathcal{A}_p \cap \mathcal{A}_q \rangle) \subseteq \mathcal{L}(I)$ (cf. Definition 2.14), the deterministic interface specification I, and Definition 2.3 we conclude the existence of some $I'' \in S_I$ satisfying $I' \stackrel{b'}{\Longrightarrow}_I I''$, i.e.

$$\exists i', i''.\, I' \xrightarrow{\tau}_I \ldots \xrightarrow{\tau}_I i' \xrightarrow{b'}_I i'' \xrightarrow{\tau}_I \ldots \xrightarrow{\tau}_I I''$$

Hence by Definition 2.4 Rule (3) (and (5) if $b' = a$):

$$p'\|I' \xrightarrow{\tau}_{p\|I} \ldots \xrightarrow{\tau}_{p\|I} p'\|i' \xrightarrow{a}_{p\|I} p''\|i'' \xrightarrow{\tau}_{p\|I} \ldots \xrightarrow{\tau}_{p\|I} p''\|I''$$

This shows together with $(*)$ that $p''\|I''$ is reachable in $p\|I$, i.e. (1) holds. Part (2) is a consequence of Definition 3.3 (3), because of the existence of i' und i'', the reachability of $p''\|i''$ in $p\|I$, and $p'\|i' \xrightarrow{a}_{p\|I} p''\|i''$. $\square$

Now we are able to prove the key property for the *completeness proof* of the $\mathcal{RM}$-Method, which implies *context preservation* in the sense of Definition 3.1(ii), as we require *isomorphie*, $=$, instead of *semantic equivalence*.

Proposition 3.8 (Context Preservation) $\forall p, q \in \mathcal{P}, I \in \mathcal{I}(p,q).\ p\|q = \overline{\Pi}_I(p)\|q$.

Proof: Let $p = ((S_p, \mathcal{A}_p \cup \{\tau\}, \longrightarrow_p, \uparrow_p), p), q = ((S_q, \mathcal{A}_q \cup \{\tau\}, \longrightarrow_q, \uparrow_q), q) \in \mathcal{P}$, $I \in \mathcal{I}(p,q)$ and $\mathcal{A} =_{df} \mathcal{A}_I = \mathcal{A}_p \cap \mathcal{A}_q$. For this proof we define the following processes:

$$\begin{array}{rcl} p\|q & = & ((S_1, \mathcal{A}_p \cup \mathcal{A}_q \cup \{\tau\}, \longrightarrow_1, \uparrow_1), p\|q) \\ \Pi_I(p) & = & ((S_{p_I}, \mathcal{A}_p \cup \{\tau\}, \longrightarrow_{p_I}, \uparrow_{p_I}), p) \\ \Pi_I(p)\|q & = & ((S_2, \mathcal{A}_p \cup \mathcal{A}_q \cup \{\tau\}, \longrightarrow_2, \uparrow_2), p\|q) \end{array}$$

Then both S_1 and S_2 are subsets of $\{(p'\|q') | p' \in S_p, q' \in S_q\}$. Thus, it remains to show that $S_1 = S_2$, $\longrightarrow_1 = \longrightarrow_2$, and $\uparrow_1 = \uparrow_2$ holds. For this purpose we define for $i = 1, 2$:

- S_i^n, the subset of states reachable in n steps from the initial state,
- $\longrightarrow_i^n$, 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[7] 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. $\longrightarrow_1^{n-1} = \longrightarrow_2^{n-1}$
2. $S_1^n = S_2^n$
3. $\uparrow_1^{n-1} = \uparrow_2^{n-1}$.

First we verify point (1) of the induction step according to the operational rules (cf. Definition 2.4), i.e. we show the equivalence "$p'\|q' \xrightarrow{a}_1 p''\|q'' \iff p'\|q' \xrightarrow{a}_2 p''\|q''$". This requires the investigation of Rules 3, 4 and 5 of Definition 2.4.

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

$$\begin{array}{lcl} & & p'\|q' \xrightarrow{a}_1 p''\|q'' \\ \text{(Rule 3)} & \iff & p' \xrightarrow{a}_p p'' \\ \text{(Lemma 3.7, 3.6, resp.}^8\text{)} & \iff & p' \xrightarrow{a}_{p_I} p'' \\ \text{(Rule 3)} & \iff & p'\|q' \xrightarrow{a}_2 p''\|q'' \end{array}$$

[7]Remember that we are dealing with finite state systems.
[8]More precisely: for "$\Rightarrow$" we apply Lemma 3.7 and for "$\Leftarrow$" Lemma 3.6.

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

$$\begin{array}{lcl} & & p' \| q' \xrightarrow{a}_1 p'' \| q'' \\ (\text{Rule 4}) & \Longleftrightarrow & q' \xrightarrow{a}_q q'' \\ (\text{Rule 4}) & \Longleftrightarrow & p' \| q' \xrightarrow{a}_2 p'' \| q'' \end{array}$$

Regel 5: Here we have $a \in \mathcal{A}$, $p' \xrightarrow{a}_p p''$, and $q' \xrightarrow{a}_q q''$:

$$\begin{array}{lcl} & & p' \| q' \xrightarrow{a}_1 p'' \| q'' \\ (\text{Rule 5}) & \Longleftrightarrow & p' \xrightarrow{a}_p p'' \wedge q' \xrightarrow{a}_q q'' \\ (\text{Lemma 3.7, 3.6, resp.}) & \Longleftrightarrow & p' \xrightarrow{a}_{p_I} p'' \wedge q' \xrightarrow{a}_q q'' \\ (\text{Rule 5}) & \Longleftrightarrow & p' \| q' \xrightarrow{a}_2 p'' \| q'' \end{array}$$

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 3.3), which leaves us with the case of an a-undefinedness. In order to prove the equivalence "$(p' \| q') \uparrow_1 a \Longleftrightarrow (p' \| q') \uparrow_2 a$" we must deal with the five applicable rules given in Definition 2.4.

Rule 8: Here we have $p' \uparrow_p a$ and $a \notin \mathcal{A}_q$, and therefore:

$$\begin{array}{lcl} & & (p' \| q') \uparrow_1 a \\ (\text{Rule 8}) & \Longleftrightarrow & p' \uparrow_p a \\ (\text{Definition 3.3 (5) (a)}^{9}) & \Longleftrightarrow & p' \uparrow_{p_I} a \\ (\text{Rule 8}) & \Longleftrightarrow & (p' \| q') \uparrow_2 a \end{array}$$

Rule 9: Here we have $p' \uparrow_p a$ and $q' \xrightarrow{a}_q$, hence $a \in \mathcal{A}_q$:

$$\begin{array}{lcl} & & (p' \| q') \uparrow_1 a \\ (\text{Rule 9}) & \Longleftrightarrow & p' \uparrow_p a \wedge q' \xrightarrow{a}_q \\ (\text{Definition 3.3 (5) (a)}) & \Longleftrightarrow & p' \uparrow_{p_I} a \wedge q' \xrightarrow{a}_q \\ (\text{Rule 9}) & \Longleftrightarrow & (p' \| q') \uparrow_2 a \end{array}$$

[9] Ad "$\Leftarrow$": The application of Definition 3.3 (5) (b) is not possible. Otherwise we would have by Rule (3) that $p' \| q' \xrightarrow{a}_1$, and therefore by Rule (3), $p' \| q' \xrightarrow{a}_2$ in contradiction to $(p' \| q') \uparrow_2 a$.

Rule 10: Here we have $q' \uparrow_q a$ and $a \notin \mathcal{A}_p$:

$$\begin{array}{rcl} & & (p' \| q') \uparrow_1 a \\ (\text{Rule } 10) & \Longleftrightarrow & q' \uparrow_q a \\ (\text{Rule } 10) & \Longleftrightarrow & (p' \| q') \uparrow_2 a \end{array}$$

Rule 11: Here we have $q' \uparrow_q a$ and $p' \xrightarrow{a}_p$, hence $a \in \mathcal{A}_p$:

$$\begin{array}{lcl} & & (p' \| q') \uparrow_1 a \\ (\text{Rule } 11) & \Longleftrightarrow & q' \uparrow_q a \wedge p' \xrightarrow{a}_p \\ (\text{Lemma 3.7, 3.6, resp.}) & \Longleftrightarrow & q' \uparrow_q a \wedge p' \xrightarrow{a}_{p_I} \\ (\text{Rule } 11) & \Longleftrightarrow & (p' \| q') \uparrow_2 a \end{array}$$

Rule 12: Here we have $p' \uparrow_p a$ and $q' \uparrow_q a$, and therefore:

<u>**"$\Rightarrow$":**</u>

$$\begin{array}{lcl} (p' \| q') \uparrow_1 a & \Rightarrow & p' \uparrow_p a \wedge q' \uparrow_q a \\ (\text{Definition 3.3 (5) (a)}) & \Rightarrow & p' \uparrow_{p_I} a \wedge q' \uparrow_q a \\ (\text{Rule } 12) & \Rightarrow & (p' \| q') \uparrow_2 a \end{array}$$

<u>**"$\Leftarrow$":**</u>

$$\begin{array}{lcl} (p' \| q') \uparrow_2 a & \Rightarrow & p' \uparrow_{p_I} a \wedge q' \uparrow_q a \\ (\text{Definition 3.3 (5)}) & \Rightarrow & (5a) \quad p' \uparrow_p a \wedge q' \uparrow_q a \Rightarrow \ (\text{Rule } 12) \ (p' \| q') \uparrow_1 a \\ & \text{or} & (5b) \quad p' \xrightarrow{a}_p \wedge q' \uparrow_q a \Rightarrow \ (\text{Rule } 11) \ (p' \| q') \uparrow_1 a \end{array}$$

□

Together with Lemma 3.5 and the obvious fact that $\overline{\Pi}$ is reducing, this proposition yields:

Proposition 3.9 *The mapping $\overline{\Pi} : \mathcal{I} \times \mathcal{P} \longrightarrow \mathcal{P}$ of Definition 3.3 defines a reduction operator.*

We conjecture that the reduction operator $\overline{\Pi}$ is optimal in the following sense.

Conjecture 3.10 (Optimality of $\overline{\Pi}$)
Let Π be an arbitrary reduction operator according to Definition 3.1, $p \in \mathcal{P}$ and $I \in \mathcal{I}(p)$. Then $\mathcal{M}(\overline{\Pi}_I(p))$ has not more states and transitions than $\mathcal{M}(\Pi_I(p))$, i.e. $\overline{\Pi}$ is maximal reducing.

3.3 Algorithmic View

In this section we present an algorithmic characterization of $\overline{\Pi}$ on the basis of a data flow analysis algorithm. As a byproduct we obtain the important *representation independency* of the interface specifications (cf. Corollary 3.20), which we have used already in the proof of Lemma 3.7.

In order to prepare an algorithmic characterization of the reduction operator $\overline{\Pi}_I(p)$ the following proposition introduces sets IL_q that contain exactly those actions which q can perform within the global context described by I. In particular, as every state can engage in ϵ, this implies that a state q is reachable in the process $\overline{\Pi}_I(p)$ if and only if $\epsilon \in IL_q$.

Proposition 3.11 (IL$_q$-Sets)
Let $p \in \mathcal{P}$, $I \in \mathcal{I}(p)$, $q \in S_p$, and $\mathcal{L}_I(q) =_{df} \{a \in \mathcal{A}_I \cup \{\epsilon\} | \exists q \| i \in S_{p\|I}.\, i \xrightarrow{a}_I\}$. Then $\overline{\Pi}_I(p)$ is determined by the sets $IL_q =_{df} \mathcal{L}_I(q) \cup (\mathcal{A}_p \setminus \mathcal{A}_I) \cup \{\tau\}$ and a finite sequence of reduction steps for all $q \in S_p$:

1. *If $\epsilon \notin IL_q$, then eliminate q and all transitions ending or starting at q.*
2. *If $q \xrightarrow{a}_p$ but $a \notin IL_q$, then eliminate all a-transitions starting at q and add $q \uparrow_p a$ to the divergence relation.*

Proof: It suffices to show that

1. $\forall q \in S_p.\, q \in S \iff \epsilon \in IL_q$ and
2. $\forall q \in S, a \in \mathcal{A} \cup \{\tau\}.\, q \xrightarrow{a} \iff q \xrightarrow{a}_p \wedge a \in IL_q$.

The proof of (1) is straightforward by using the definition of IL_q and Definition 3.3 (1). A case distinction is needed for (2): for the case $a \in (\mathcal{A} \setminus \mathcal{A}_I) \cup \{\tau\}$ Definitions 3.3 (3), 2.4 (3), and the definition of IL_q must be considered. The case $a \in \mathcal{A}_I \cup \{\epsilon\}$ requires to look at Definitions 3.3 (3), 2.4 (5), 2.14 and the definition of IL_q. □

Proposition 3.11 shows how to compute $\overline{\Pi}_I(p)$ on the basis of $\mathcal{L}_I(q)$, $q \in S_p$. The key to a complete algorithm for $\overline{\Pi}_I(p)$ is the following easy to prove characterization for the sets $\mathcal{L}_I(q)$, which is the basis for the algorithm developed in the next section.

Lemma 3.12 (Characterization of $\mathcal{L}_I(q)$)
Let $q \in S_p$ and $\mathcal{L}'_I(q) =_{df} \{a \in \mathcal{A}_I \cup \{\epsilon\} | \exists v \in \mathcal{A}_I^*.\, av \in \bigcup\{\mathcal{L}(i) | q\|i \in S_{p\|I}\}\}$ the set of prefixes of length at most one of $\bigcup\{\mathcal{L}(i) | q\|i \in S_{p\|I}\}$. Then $\mathcal{L}_I(q) = \mathcal{L}'_I(q)$ holds.

3.4 Determining $\mathcal{L}_I(q)$

In this section we show how to compute the sets $\mathcal{L}_I(q)$ on the basis of their alternative characterizations by means of a data flow analysis algorithm. This requires a brief review of the relevant data flow analysis scenario.

Given a complete partial order $\langle C; \sqsubseteq \rangle$, whose elements are intended to express the relevant data flow information, the *local abstract semantics* of a process $((S_p, \mathcal{A}_p \cup \{\tau\}, \longrightarrow_p, \uparrow_p), p)$ is defined by the semantic functional $[\![\cdot]\!] : (\longrightarrow_p) \to (C \to C)$ which maps each transition $t \in \longrightarrow_p$ to a transformation on C. $[\![\cdot]\!]$ extends to paths $pth = (t_1, \ldots, t_q)$, $q \geq 0$, in p in the usual way: $[\![pth]\!] =_{df} [\![t_q]\!] \circ \cdots \circ [\![t_1]\!]$.

Let us now fix an arbitrary process $p = ((S_p, \mathcal{A}_p \cup \{\tau\}, \longrightarrow_p, \uparrow_p), p) \in \mathcal{P}$ and an interface specification $I = ((S_I, \mathcal{A}_I, \longrightarrow_I, \uparrow_I), I) \in \mathcal{I}$. For technical reasons we assume w.l.o.g. that p possesses no transition to its start state.

For our application we need the following local abstract semantics over the complete partial order $\langle 2^{\mathcal{L}(I)}; \subseteq \rangle$.

Definition 3.13 (Local Abstract Semantics)
For $a \in \mathcal{A}$ consider the functions $\mathcal{E}^I_a : 2^{\mathcal{L}(I)} \to 2^{\mathcal{L}(I)}$ with

$$\mathcal{E}^I_a(\mathcal{L}) =_{df} \begin{cases} \mathcal{L}_a & \textit{if } a \in \mathcal{A}_I \\ \mathcal{L} & \textit{otherwise} \end{cases}$$

Then the local abstract semantic function for $(q, a, q') \in \longrightarrow_p$ with respect to I is defined by $[\![(q, a, q')]\!] =_{df} \mathcal{E}^I_a$.

The following property of these local semantic functions is important:

Lemma 3.14 (Additivity)
The functions $\mathcal{E}^I_a$ are additive, i.e. for all $\{\mathcal{L}_k | k \geq 0\} \subseteq 2^{\mathcal{L}(I)}$ we have

$$\mathcal{E}^I_a(\bigcup\{\mathcal{L}_k | k \geq 0\}) = \bigcup\{\mathcal{E}^I_a(\mathcal{L}_k) | k \geq 0\}$$

As a consequence, the local abstract semantic functions $[\![t]\!]$ are additive for all $t \in \longrightarrow_p$.

The local semantics can be globalized according to two strategies: the "operational" *join over all paths (JOP) strategy*, which (usually) directly reflects the intuition behind the analysis problem and the "denotational" *minimal fixed point (MFP) strategy*, which is algorithmic (cf. [Kil73, KU77]).[10] In the following $P[q, q']$ denotes the set of all finite paths from q to q'.

[10] Originally, a dual setup was proposed, considering *meet over all paths* and *maximal fixed point* strategies.

Definition 3.15 (*JOP*-Solution)
$\forall q \in S_p, c_0 \in C.\ JOP_{c_0}(q) =_{df} \bigsqcup\{[\![pth]\!](c_0) | pth \in P[p,q]\}$

For our application we have the following characterization of the *JOP*-Solution.

Proposition 3.16 (Characterization of the *JOP*-Solution)
$\forall q \in S_p.\ JOP_{\mathcal{L}(I)}(q) = \bigcup\{\mathcal{L}(i) | q \| i \in S_{p\|I}\}$

Proof: For "$\supseteq$" one easily establishes

$$\forall q\|i \in S_{p\|I} \exists pth \in P[p,q].\ \mathcal{L}(i) \subseteq [\![pth]\!](\mathcal{L}(I))$$

by induction on the length of a path from $p\|I$ to $q\|i$, and for "$\subseteq$" it is sufficient to prove for all paths $pth \in P[p,q]$ that $w \in [\![pth]\!](\mathcal{L}(I))$ implies $\exists q\|i \in S_{p\|I}.w \in \mathcal{L}(i)$ by induction on the length of pth. □

The *MFP*-solution iteratively approximates the smallest solution of a set of simultaneous equations that express consistency between data flow informations.

Definition 3.17 (*MFP*-Solution)
The least solution l_{c_0} of the equation system consisting of the equation

$$l(q) = \begin{cases} c_0 & \textit{if } q = p \\ l(q) \sqcup \bigsqcup\{[\![(q', a, q)]\!](l(q')) | (q', a, q) \in \longrightarrow_p\} & \textit{otherwise} \end{cases}$$

for each $q \in S_p$ includes the MFP-Solution *with repect to the initial information $c_0 \in C$.*

$$\forall q \in S_p, c_0 \in C.\ MFP_{c_0}(q) =_{df} l_{c_0}(q)$$

As in our application, this often leads to an algorithmic description. The well-known coincidence theorem of Kam and Ullman [KU77] bridges the gap between the *JOP*-Solution and the *MFP*-Solution.

Theorem 3.18 (Coincidence Theorem)
If all local abstract semantic functions $[\![t]\!]$ for $t \in \longrightarrow_p$ are additive, then the MFP-Solution is correct *and* complete *with respect to the JOP-Solution, i.e.*

$$\forall q \in S_p, c_0 \in C.\ JOP_{c_0}(q) = MFP_{c_0}(q)$$

Thus according to Lemma 3.14, we can compute the desired *JOP*-solution (cf. Proposition 3.16) by iteratively approximating the smallest solution of the equation system defined in Definition 3.17 in the following fashion:

Procedure 3.19 (Language Labelling)
Given $p \in \mathcal{P}$ and $I \in \mathcal{I}(p)$, the iterative Language Labelling Procedure *works by successively enlarging approximative labellings according to the following two steps:*

1. *It initially labels p with $\mathcal{L}(I)$ and all the other states with the empty language.*
2. *If a state q of p is currently labelled by $\mathcal{L}$ and q' is one of its a-successors then $\mathcal{E}^I_a(\mathcal{L})$ is joined to the current language labelling q', until a fixed point is reached.*

In terms of DFA, the Language Labelling Procedure computes the minimal fixed point solution with respect to the start information $\mathcal{L}(I)$.

The Language Labelling Procedure 3.19 has been implemented by means of a *workset algorithm*, as a part of the META–Frame environment [MCS95]. Its time and state complexity can be estimated by the product of the number of transitions of p and the number of states of I.

As the algorithm does not exploit the structure of the representation of interface language, we obtain as a simple but important consequence:

Corollary 3.20 (Representation Independency)
For all $p \in \mathcal{P}$ and for all $I, I' \in \mathcal{I}(p)$ we have:

$$\mathcal{L}(I) = \mathcal{L}(I') \text{ implies } \overline{\Pi}_I(p) = \overline{\Pi}_{I'}(p)$$

4 Minimization Method

In this section, we develop the $\mathcal{RM}$-method, which compositionally minimizes finite state distributed systems, on top of a reduction operator Π. This method is *correct* in that it always produces processes that are smaller in the specification-implementation preorder (cf. Theorem 4.1), guaranteeing that even faulty interface specifications will never allow us to establish wrong properties. Moreover, it is complete, in that it only produces semantically equivalent processes as long as the interface specifications are correct (cf. Theorem 4.2), guaranteeing that the reduction preserves all the considered properties in this case. The $\mathcal{RM}$-method is illustrated by means of an example, where the apparent complexity is exponential in the number of components, whereas the algorithmic and the real complexity are linear. The effects of the $\mathcal{RM}$-Method are shown via screen dumps that have been obtained from our implementation within the META–Frame environment.

The $\mathcal{RM}$-Method is taylored to deal with processes of the form $\mathrm{P} = (p_1 \| \dots \| p_n)\langle L \rangle$. This form, called standard concurrent form in CCS, is of particular interest, as it is responsible for the state explosion problem and therefore characterizes the processes that are critical during analysis and verification. Our method expects the finite state system 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:

$$\mathrm{P} = (p_1 \|_{I_1} p_2 \|_{I_2} \dots \|_{I_{n-1}} p_n)\langle L\rangle$$

It proceeds by successively constructing transition systems $\mathcal{P}_i$ as follows:

$$\underbrace{\underbrace{\underbrace{(p_1 \|_{I_1}}_{\mathcal{P}_1} p_2}_{\mathcal{P}_2} \|_{I_2} \dots \|_{I_{n-1}} p_n)\langle L\rangle}_{\vdots \atop \mathcal{P}_n}$$

where $\mathcal{P}_i$ is defined by:

- $\mathcal{P}_1 =_{df} \mathcal{M}(\Pi_{I_1}(\mathcal{M}(p_1\langle \mathcal{A}_{I_1} \cup L\rangle)))$,
- $\mathcal{P}_i =_{df} \mathcal{M}(\Pi_{I_i}(\mathcal{M}((\mathcal{P}_{i-1}\|p_i)\langle \mathcal{A}_{I_i} \cup L\rangle)))$ for $2 \leq i \leq n-1$, and
- $\mathcal{P}_n =_{df} \mathcal{M}((\mathcal{P}_{n-1}\|p_n)\langle L\rangle)$.

In order to avoid unnecessarily large intermediate transition systems during the construction of the minimal transition system, it is important to minimize all the intermediate constructions as it is done above. Note that our method covers the naive method (only using $\mathcal{M}$) and methods which only consider the correspondence of the parallel and the window operator (see Proposition 2.6). The new additional power of the $\mathcal{RM}$-Method is due to the reduction operator Π which minimizes all intermediate transition systems $\mathcal{P}_i$ according to global constraints specified in terms of the interface specifications I_i (for $1 \leq i \leq n-1$).

In the remainder of this section let P and $\mathcal{P}_i$ be as defined above and $Q_i =_{df} (p_{i+1}\| \dots p_n)$ for $1 \leq i \leq n$.[11] Then we obtain the following correctness result, which is independent of the correctness of the interface specifications.

Theorem 4.1 (Correctness of the $\mathcal{RM}$-Method) $\forall 1 \leq i \leq n.\, (\mathcal{P}_i\|Q_i)\langle L\rangle \preceq P$.

Proof: The proof is done by induction on i.

Base case ($i = 1$):

$$\begin{array}{lcl} & & (\mathcal{P}_1\|Q_1)\langle L\rangle \\ (\text{def. } \mathcal{P}_1) & = & (\mathcal{M}(\Pi_{I_1}(\mathcal{M}(p_1\langle \mathcal{A}_{I_1} \cup L\rangle)))\|Q_1)\langle L\rangle \\ (\text{Prop. 2.11, 2.12, Def. 3.1(i)}) & \preceq & (p_1\langle \mathcal{A}_{I_1} \cup L\rangle\|Q_1)\langle L\rangle \\ (\text{Prop. 2.6}) & = & (p_1\|Q_1)\langle L\rangle \\ (\text{def. P}) & = & \mathrm{P} \end{array}$$

[11] Q_n denotes the empty process consisting of a single state, an empty alphabet and no transition.

Induction step ($i-1 \longrightarrow i$):

The case $2 \leq i \leq n-1$ proceeds as follows:

$$\begin{array}{lcl}
 & & (\mathcal{P}_i \| Q_i)\langle L\rangle \\
\text{(def. } \mathcal{P}_i\text{)} & = & (\mathcal{M}(\Pi_{I_i}(\mathcal{M}((\mathcal{P}_{i-1}\|p_i)\langle \mathcal{A}_{I_i} \cup L\rangle)))\|Q_i)\langle L\rangle \\
\text{(Prop. 2.11, 2.12, Def. 3.1(i))} & \preceq & ((\mathcal{P}_{i-1}\|p_i)\langle \mathcal{A}_{I_i} \cup L\rangle \| Q_i)\langle L\rangle \\
\text{(Prop. 2.6)} & = & ((\mathcal{P}_{i-1}\|p_i)\|Q_i)\langle L\rangle \\
\text{(Prop. 2.5)} & = & (\mathcal{P}_{i-1}\|(p_i\|Q_i))\langle L\rangle \\
\text{(def. } Q_{i-1}\text{)} & = & (\mathcal{P}_{i-1}\|Q_{i-1})\langle L\rangle \\
\text{(ind. hyp.)} & \preceq & \mathrm{P}
\end{array}$$

For $i = n$ we conclude:

$$\begin{array}{lcl}
 & & (\mathcal{P}_n \| Q_n)\langle L\rangle \\
\text{(def. } Q_n\text{)} & = & \mathcal{P}_n\langle L\rangle \\
\text{(def. } \mathcal{P}_n\text{)} & = & (\mathcal{M}((\mathcal{P}_{n-1}\|p_n)\langle L\rangle))\langle L\rangle \\
\text{(Prop. 2.11, 2.12, Def. 3.1(i), def. of } \langle\cdot\rangle\text{)} & \preceq & (\mathcal{P}_{n-1}\|p_n)\langle L\rangle \\
\text{(def. } Q_{n-1}\text{)} & = & (\mathcal{P}_{n-1}\|Q_{n-1})\langle L\rangle \\
\text{(ind. hyp.)} & \preceq & \mathrm{P}
\end{array}$$

□

For $i = n$ Theorem 4.1 states that $\mathcal{P}_n \preceq \mathrm{P}$. This is already enough to guarantee the correctness of the method, i.e. that a proof of a $\approx^d$-consistent property for $\mathcal{P}_n$ is valid for P. Thus wrong interface specifications never lead to wrong proofs. They may only prevent a successful verification of a valid statement. In order to guarantee the success of the method, the correctness of the interface specifications is sufficient.

Theorem 4.2 (Completeness of the $\mathcal{RM}$-Method)
$\forall 1 \leq i \leq n. (\forall j \leq i. I_j \in \mathcal{I}(p_1\|\ldots\|p_j, Q_j))$ *implies* $(\mathcal{P}_i\|Q_i)\langle L\rangle \approx^d P$.

Proof: The proof, which is done by induction on i again, requires special attention: an instance of the induction hypothesis is necessary to establish an auxiliary statement concerning the correctness of the given interface specifications in the special proof context.

For $i = 1$ we have:

$$\begin{array}{lcl}
 & & (\mathcal{P}_1 \| Q_1)\langle L\rangle \\
(\text{def. } \mathcal{P}_1) & = & (\mathcal{M}(\Pi_{I_1}(\mathcal{M}(p_1\langle \mathcal{A}_{I_1} \cup L\rangle)))\|Q_1)\langle L\rangle \\
(\text{def. } \mathcal{M}, \text{Prop. } 2.12) & \approx^d & (\Pi_{I_1}(\mathcal{M}(p_1\langle \mathcal{A}_{I_1} \cup L\rangle))\|Q_1)\langle L\rangle \\
(\text{Prop. } 3.2, 2.12) & \approx^d & (\Pi_{I_1}(p_1\langle \mathcal{A}_{I_1} \cup L\rangle)\|Q_1)\langle L\rangle \\
(\text{Def. } 3.1\text{(ii)}, \text{Prop. } 2.17, 2.12) & \approx^d & (p_1\langle \mathcal{A}_{I_1} \cup L\rangle\|Q_1)\langle L\rangle \\
(\text{Theorem } 2.6) & = & (p_1\|Q_1)\langle L\rangle \\
(\text{def. P}) & = & \mathrm{P}
\end{array}$$

The induction step, $i-1 \longrightarrow i$, needs the following auxiliary statement:

$$(*) \qquad I_i \in \mathcal{I}(p_1\|\ldots\|p_i, Q_i) \text{ implies } I_i \in \mathcal{I}(\mathcal{M}((\mathcal{P}_{i-1}\|p_i)\langle \mathcal{A}_{I_i} \cup L\rangle), Q_i)$$

The statement follows by Definition 2.14 considering

$$\begin{array}{lcl}
 & & \mathcal{L}(I_i) \\
(\text{Def. } 2.14) & \supseteq & \mathcal{L}(((p_1\|\ldots\|p_i)\|Q_i)\langle \mathcal{A}_{I_i}\rangle) \\
(\text{def. } Q_i, \text{Prop. } 2.5, 2.12, \text{La. } 2.13(2)) & = & \mathcal{L}((p_1\|\ldots\|p_n)\langle \mathcal{A}_{I_i}\rangle) \\
(\text{ind. hyp. for } L = \mathcal{A}_{I_i}, \text{La. } 2.13(2)) & = & \mathcal{L}((\mathcal{P}_{i-1}\|Q_{i-1})\langle \mathcal{A}_{I_i}\rangle) \\
(\text{def. } Q_{i-1}, \text{Prop. } 2.5, 2.12, \text{La. } 2) & = & \mathcal{L}(((\mathcal{P}_{i-1}\|p_i)\|Q_i)\langle \mathcal{A}_{I_i}\rangle) \\
(\text{Prop. } 2.6, \text{La. } 2.13(2)) & = & \mathcal{L}(((\mathcal{P}_{i-1}\|p_i)\langle \mathcal{A}_{I_i} \cup L\rangle\|Q_i)\langle \mathcal{A}_{I_i}\rangle) \\
(\text{def. } \mathcal{M}, \text{Prop. } 2.12, \text{La. } 2.13(2)) & = & \mathcal{L}((\mathcal{M}((\mathcal{P}_{i-1}\|p_i)\langle \mathcal{A}_{I_i} \cup L\rangle)\|Q_i)\langle \mathcal{A}_{I_i}\rangle)
\end{array}$$

Now, the case $2 \leq i \leq n-1$ proceeds as follows:

$$\begin{array}{lcl}
 & & (\mathcal{P}_i \| Q_i)\langle L\rangle \\
(\text{def. } \mathcal{P}_i) & = & (\mathcal{M}(\Pi_{I_i}(\mathcal{M}((\mathcal{P}_{i-1}\|p_i)\langle \mathcal{A}_{I_i} \cup L\rangle)))\|Q_i)\langle L\rangle \\
(\text{def. } \mathcal{M}, \text{Prop. } 2.12) & \approx^d & (\Pi_{I_i}(\mathcal{M}((\mathcal{P}_{i-1}\|p_i)\langle \mathcal{A}_{I_i} \cup L\rangle))\|Q_i)\langle L\rangle \\
((*), \text{Def. } 3.1(\text{ii}), \text{Prop. } 2.12) & \approx^d & (\mathcal{M}((\mathcal{P}_{i-1}\|p_i)\langle \mathcal{A}_{I_i} \cup L\rangle)\|Q_i)\langle L\rangle \\
(\text{def. } \mathcal{M}, \text{Prop. } 2.12) & \approx^d & ((\mathcal{P}_{i-1}\|p_i)\langle \mathcal{A}_{I_i} \cup L\rangle\|Q_i)\langle L\rangle \\
(\text{Prop. } 2.6) & = & ((\mathcal{P}_{i-1}\|p_i)\|Q_i)\langle L\rangle \\
(\text{Prop. } 2.5, 2.12, \text{def. } Q_{i-1}) & \approx^d & (\mathcal{P}_{i-1}\|Q_{i-1})\langle L\rangle \\
(\text{ind. hyp.}) & \approx^d & \mathrm{P}
\end{array}$$

For $i = n$ we conclude:

$$\begin{array}{lcl}
 & & (\mathcal{P}_n \| Q_n)\langle L\rangle \\
(\text{def. } Q_n) & = & \mathcal{P}_n\langle L\rangle \\
(\text{def. } \mathcal{P}_n) & = & (\mathcal{M}((\mathcal{P}_{n-1}\|p_n)\langle L\rangle))\langle L\rangle \\
(\text{def. } \mathcal{M}, \text{def. of } \langle\cdot\rangle, \text{Prop. } 2.12) & \approx^d & (\mathcal{P}_{n-1}\|p_n)\langle L\rangle \\
(\text{def. } Q_{n-1}) & = & (\mathcal{P}_{n-1}\|Q_{n-1})\langle L\rangle \\
(\text{ind. hyp.}) & \approx^d & \mathrm{P}
\end{array}$$

□

In practice, P is usually totally defined. Applying Theorem 4.1 it is easy to see that the proof of $\mathcal{P}_n \approx^d \mathrm{P}$ reduces to the verification of the total definedness of $\mathcal{P}_n$ in this case:

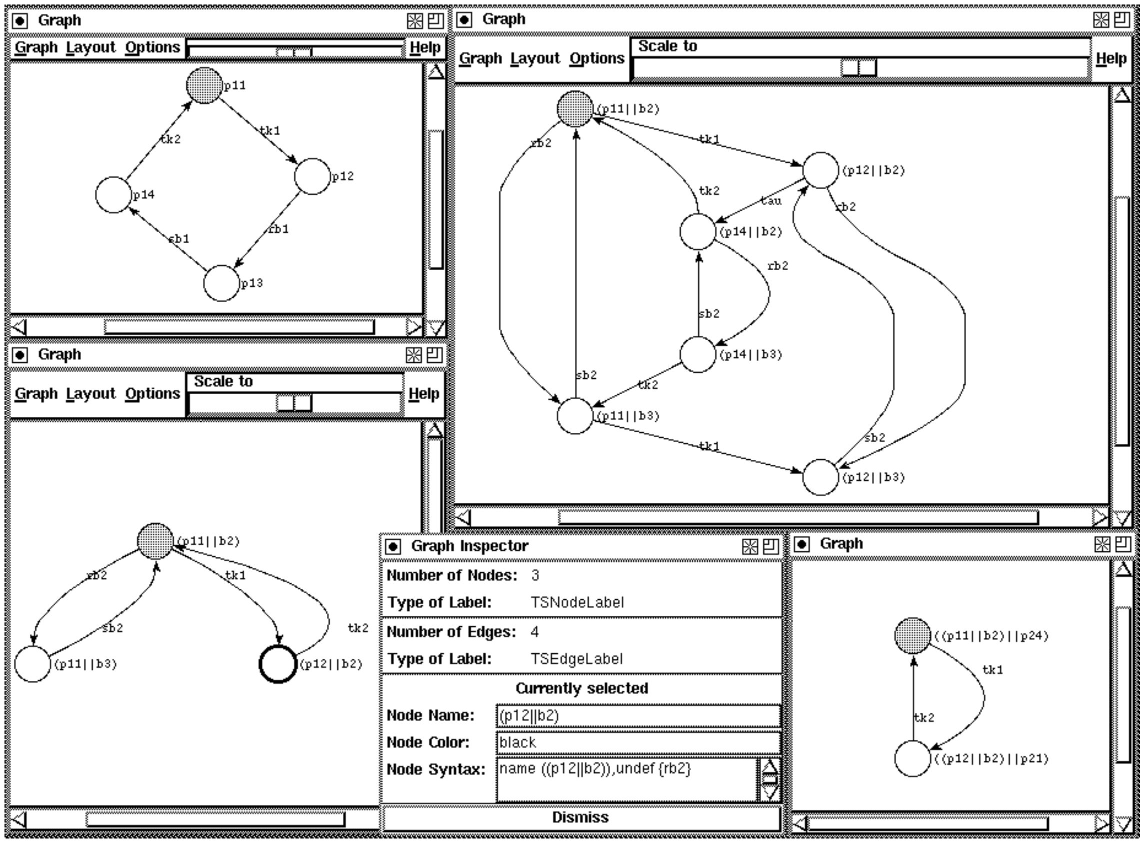
Corollary 4.3 (Total Definedness)
Whenever P is totally defined, we have: $\mathcal{P}_n \approx^d P$ iff $\mathcal{P}_n$ is totally defined.

Up to now, we considered the reduction operator as a parameter. The following applications use the reduction operator $\overline{\Pi}$.

4.1 An Application of the $\mathcal{RM}$-Method

The application of our method to the system of Example 2.7 and the interface specifications defined in Example 2.15 leads to a successive computation of the following three processes:

$$\begin{aligned}\mathcal{P}_1 &= \mathcal{M}(\overline{\Pi}_{I_1}(\mathcal{M}(P_1\langle\{tk1, tk2, rb1, sb1\}\rangle)))\\ \mathcal{P}_2 &= \mathcal{M}(\overline{\Pi}_{I_2}(\mathcal{M}((\mathcal{P}_1\|B)\langle\{tk1, tk2, rb2, sb2\}\rangle)))\\ \mathcal{P}_3 &= \mathcal{M}((\mathcal{P}_2\|P_2)\langle\{tk1, tk2\}\rangle)\end{aligned}$$



Number of Nodes: 3
Type of Label: TSNodeLabel
Number of Edges: 4
Type of Label: TSEdgeLabel
Currently selected
Node Name: (p12||b2)
Node Color: black
Node Syntax: name ((p12||b2)),undef {rb2}
Dismiss

Figure 5: Application of the Method to our Example

As already suggested in Example 2.15, $\overline{\Pi}_{I_1}$ has no effect on P_1 due to the IL_q-sets $IL_{p_{11}} = \{tk1, \epsilon\}$, $IL_{p_{12}} = \{rb1, tk2, \epsilon\}$, $IL_{p_{13}} = \{sb1, \epsilon\}$, and $IL_{p_{14}} = \{rb1, tk2, \epsilon\}$ (cf. Figure 5, top left corner). Let us now look at the stepwise computation of $\mathcal{P}_2$. Computing $\mathcal{M}((\mathcal{P}_1\|B)\langle\{tk1, tk2, rb2, sb2\}\rangle)$ leads to the process presented at the right top of Figure 5, and the application of the algorithm of Section 3.3 provides the following IL_q-sets: $IL_{p11\|b2} = \{tk1, rb2, \epsilon\}$, $IL_{p12\|b2} = \{tk2, \epsilon\} = IL_{p14\|b2}$, $IL_{p11\|b3} = \{sb2, \epsilon\}$, and $IL_{p12\|b3} = \emptyset = IL_{p14\|b3}$. $\mathcal{P}_2$ is presented at the bottom left corner of Figure 5 with the undefinedness for $rb2$ at the state $(p12\|b2)$ and for $tk1$ at the state $(p11\|b3)$.

The result of the reduction algorithm is $\mathcal{P}_3$ (cf. Figure 5, bottom right corner), and as $\mathcal{P}_3$ is totally defined, Corollary 4.3 yields that $\mathcal{P}_3 \approx^d \mathit{System}$ holds. This reflects our intuition that an observer may only see the cyclical passing of the token on the channels $tk1$ and $tk2$.

4.2 The Power of the $\mathcal{RM}$-Method

Let us consider a system guaranteeing the mutually exclusive access of n processes P_i to a common resource R as illustrated in Figure 6 for $n = 4$.[12] The idea behind the system 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 sends its request via ps_i to the resource R, which responds by transmitting the requested object. The corresponding transmission line is modelled by a buffer B_i. This choice is motivated by thinking of large objects whose transmission cannot be modelled by an atomic "handshake" communication.

In order to prove that the access is modelled as intended, we can hide everything except for the actions corresponding to the transmission of the token, and prove that the resulting process is equivalent to the process *Spec*(n) that just repeatedly executes the sequence $tk_1, \ldots, tk_n$. I.e. it is enough to show

$$\mathit{System}(n) =_{\mathit{df}} (\,R\,\|P_1\|B_1\|\ldots\|P_n\|B_n)\langle\{tk_1,\ldots,tk_n\}\rangle \approx^d \mathit{Spec}(n)$$

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 presented in Figure 7:

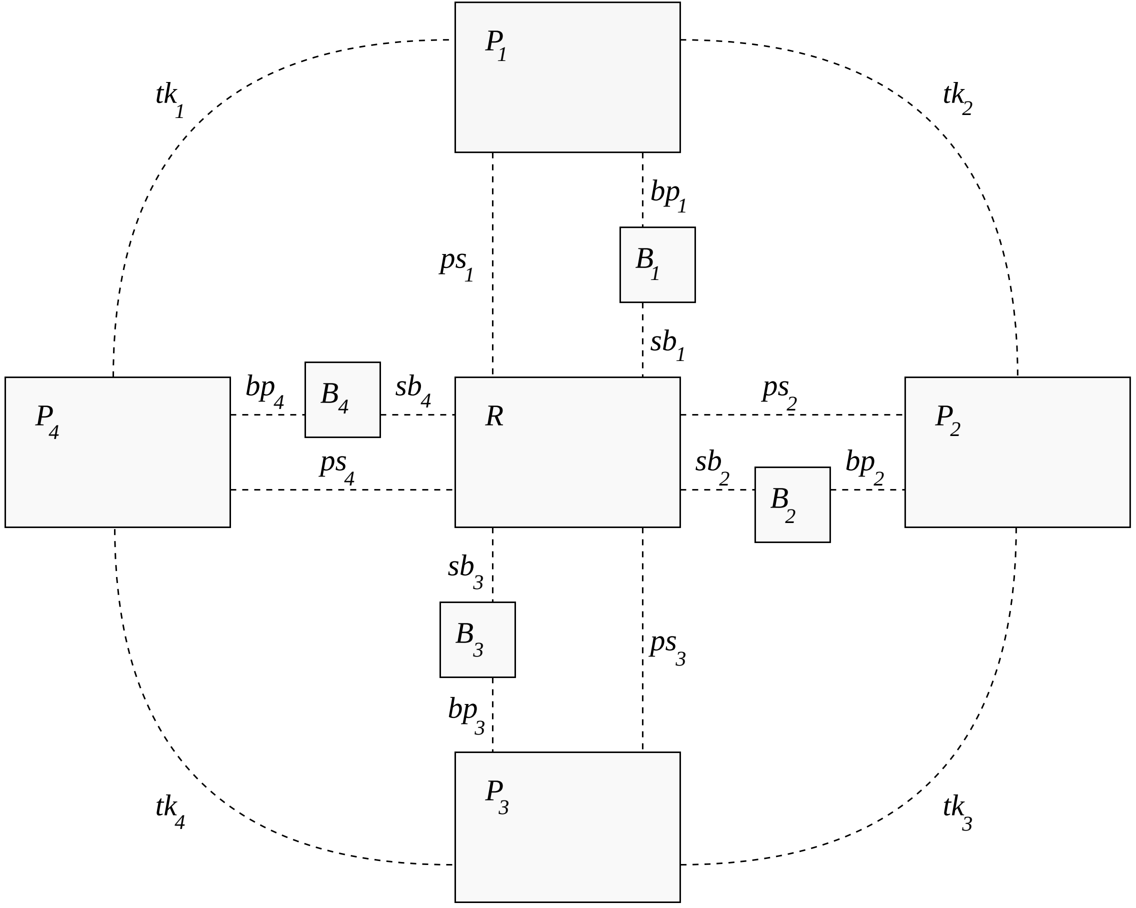
$$(\overbrace{R\|P_1\|B_1}\,\|_{I_1}\,\overbrace{P_2\|B_2}\,\|_{I_2}\ldots\|_{I_{n-1}}\,\overbrace{P_n\|B_n})\langle\{tk_1,\ldots,tk_n\}\rangle$$

The table below summarizes a quantitative evaluation of the effect 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 exact interface specifications (the algorithmic complexity), and 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:

[12]It should be noted that in contrast to all the other P_i, which are displayed correctly, P_n is assumed to be initially in the bottom right state.



where

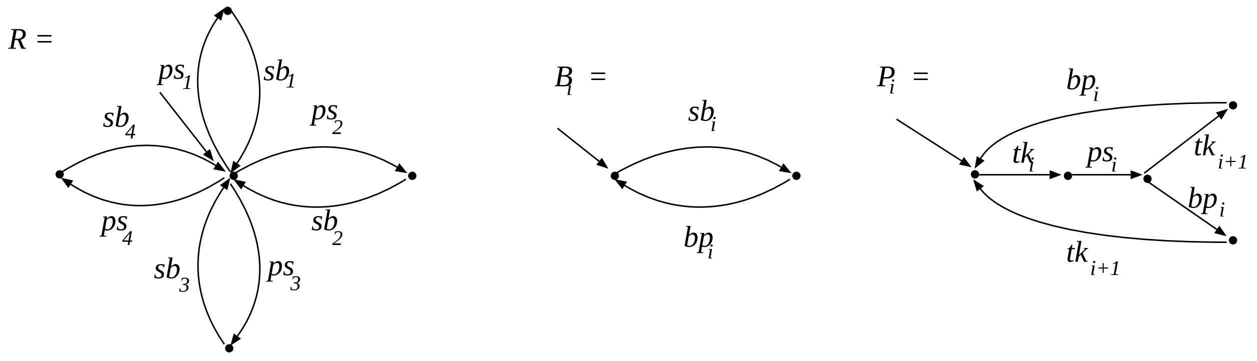


Figure 6: Round Robin Access

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

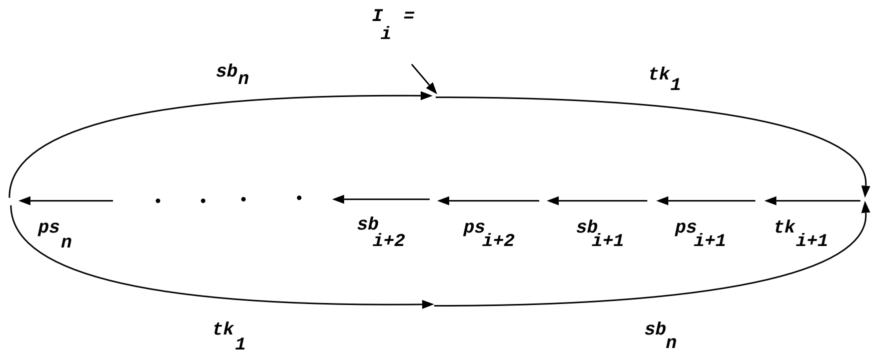


Figure 7: Exact Interface Specifications

This stresses the importance of interface specifications for automatic proof techniques. Software designers 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. Note the similarity to the situation for *while*-programs, where automatic verification depends on *loop invariants* that also need to be provided by the programmer.

5 Conclusions

We have presented a method, called $\mathcal{RM}$-Method, for the compositional minimization of finite state distributed 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 $\mathcal{RM}$-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. The $\mathcal{RM}$-Method is implemented as part of the META–Frame tool [MCS95] for the reduction operator $\overline{\Pi}$ and the semantic equivalence $\approx^d$.

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 $\mathcal{RM}$-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.

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.

Acknowledgements

We would like to thank Rance Cleaveland and Ernst–Rüdiger Olderog for helpful discussions and Dirk Koschützki for his support in implementing our method within the META–Frame environment.

References

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

[BFH90] A. Bouajjani, J.-C. Fernandez, and N. Halbwachs. Minimal model generation. In *Workshop on Automatic Verification '90*, volume 531. LNCS, 1990.

[Bry86] R. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computation*, 35(8), 1986.

[CC77] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction and approximation of fixpoints. In *Symp. Principles of Programming Languages '77*, 1977.

[CES83] E.M. Clarke, E.A. Emerson, and E. Sistla. Automatic verification of finite state concurrent systems using temporal logic specification: A practical approach. In *Symp. Principles of Programming Languages '83*, 1983.

[CGL92] E. Clarke, O. Grumberg, and D. Long. Model checking and abstraction. In *Symp. Principles of Programming Languages '92*, 1992.

[CLM89] E. Clarke, D. Long, and K. McMillan. Compositional model checking. *Proc. IEEE Symp. Logic in Computer Science*, pages 353–362, 1989.

[CPS89a] R. Cleaveland, J. G. Parrow, and B. Steffen. The concurrency workbench. In *Workshop on Automatic Verification Methods for Finite State Systems, Grenoble (France)*, volume 407. LNCS, 1989.

[CPS89b] R. Cleaveland, J. G. Parrow, and B. Steffen. A semantics based verification tool for finite state system. In *Proceedings of the Ninth International Symposium on Protocol Specification, Testing, and Verification.* North Holland, 1989.

[CR94] R. Cleaveland and J. Riely. Testing-based abstractions for value passing systems. In *Proceedings of CONCUR'94, Stockholm (Sweden)*, volume 836. LNCS, 1994.

[CS90a] R. Cleaveland and B. Steffen. A preorder for partial process specifications. In *Proceedings of CONCUR '90, Amsterdam (Netherlands)*, volume 458. LNCS, 1990.

[CS90b] R. Cleaveland and B. Steffen. When is “partial” adequate? A logic-based proof technique using partial specifications. *Proc. IEEE Symp. Logic in Computer Science*, 1990.

[DGG93] D. Dams, O. Grumberg, and R. Gerth. Generation of reduced models for checking fragments of CTL. In *Proceedings of the International Workshop on Computer-Aided Verification (CAV'93)*, volume 697, pages 479–490. LNCS, 1993.

[Fer88] J.-C. Fernandez. *Aldébaran: Un Système de Vérification par Réduction de Processus Communicants*. PhD 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’. *LNCS 207*, 1983.

[GL93] S. Graf and C. Loiseaux. Program verification using compositional abstraction. In *Proceedings FASE/TAPSOFT'93*, 1993.

[GP93] P. Godefroid and D. Pirottin. Dependencies improves partial-order verification methods. In *Proceedings of the International Workshop on Computer-Aided Verification (CAV'93)*, volume 697, pages 438–449. LNCS, 1993.

[GW91] P. Godefroid and P. Wolper. Using partial orders for the efficient verification of deadlock freedom and safety properties. In *Proceedings of the International Workshop on Computer-Aided Verification (CAV'91)*, volume 575, pages 332–342. LNCS, 1991.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall International, 1985.

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

[Kil73] G. A. Kildall. A unified approach to global program optimization. In *Symp. Principles of Programming Languages '73*, pages 194 – 206, 1973.

[KM89] R.P. Kurshan and K. McMillan. A structural induction theorem for processes. In *ACM Symposium on Principles of Distributed Computing*, 1989.

[Kru89] H. Krumm. Projections of the reachability graph and environment models, two approaches to facilitate the functional analysis of systems of cooperating finite state machines. In *Workshop on Automatic Verification of Finite State Systems, Grenoble (France)*, volume 407. LNCS, 1989.

[KU77] J. B. Kam and J. D. Ullman. Monotone data flow analysis frameworks. *Acta Informatica*, 7:309–317, 1977.

[LGS$^+$92] C. Loiseaux, S. Graf, J. Sifakis, A. Bouajjani, and S. Bensalem. Property preserving abstractions for the verification of concurrent systems. *Formal Methods in System Design, Vol 6, Iss 1, January 1995*, 1992. first published in CAV'92, LNCS 663.

[LSW94] K. G. Larsen, B. Steffen, and C. Weise. A constraint oriented proof methodology based on modal transition systems. In BRICS Notes 94-6, December 1994.

[LT88] K. G. Larsen and B. Thomsen. Compositional proofs by partial specification of processes. *Proc. IEEE Symp. Logic in Computer Science*, 1988.

[Lüt94] G. Lüttgen. Kompositionelle Minimierung endlicher verteilter Systeme, März 1994. Diplomarbeit an der RWTH Aachen.

[LX90] K.G. Larsen and L. Xinxin. Compositionality through an operational semantics of contexts. In *ICALP '90*, volume 443. LNCS, 1990.

[MCS95] T. Margaria, A. Claßen, and B. Steffen. Computer aided tool synthesis in META–Frame. GI/ITG Workshop on "Anwendung formaler Methoden beim Entwurf von Hardwaresystemen", Passau (Germany), March 1995.

[Mil80] R. Milner. A calculus for communicating systems. *LNCS 92*, 1980.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice Hall, 1989.

[Pnu90] A. Pnueli. In transition from global to modular temporal reasoning about programs. In *Logics and Models for Concurrent Systems*, volume 13. Springer Verlag, 1990.

[SG89] Z. Stadler and O. Grumberg. Network grammars, communication behaviours and automatic verification. In *Workshop on Automatic Verification Methods for Finite State Systems, Grenoble (France)*, volume 407. LNCS, 1989.

[SG90] G. Shurek and O. Grumberg. The modular framework of computer-aided verification. In *Workshop on Automatic Verification '90*, volume 531, pages 214–223. LNCS, 1990.

[Ste94] B. Steffen. Finite model checking and beyond. In BRICS Notes 94-6, December 1994.

[Val93] A. Valmari. On-the-fly verification with stubborn sets. In *Proceedings of the 5th International Conference on Computer Aided Verification (CAV'93)*, volume 697, pages 397–408. LNCS, 1993.

[Wal88] D.J. Walker. Bisimulation and divergence in CCS. *Proc. IEEE Symp. Logic in Computer Science*, 1988.

[Win90] G. Winskel. Compositional checking of validity on finite state processes. In *Workshop on Theories of Communication, CONCUR*, 1990.

[WL89] P. Wolper and V. Lovinfosse. Verifying properties of large sets of processes with network invariants. In *Workshop on Automatic Verification Methods for Finite State Systems, Grenoble (France)*, volume 407. LNCS, 1989.