Compositional State Space Generation from Lotos Programs [⋆]

Jean-Pierre Krimm and Laurent Mounier

Verimag
Zirst-Miniparc
F-38330 Montbonnot Saint-Martin, France

Abstract. This paper describes a compositional approach to generate the labeled transition system representing the behavior of a Lotos program by repeatedly alternating composition and reduction operations on subsets of its processes. To restrict the size of the intermediate Ltss generated, we generalize to the Lotos parallel composition operator the results proposed in [GS90], which consist in representing the environment of a process by an *interface*, i.e., a set of "authorized" execution sequences. This generalization allows to handle both user-given interfaces and automatically computed ones. This compositional generation method has been implemented within the Cadp toolbox and experimented on several realistic case-studies.

Introduction

Formal verification is a part of software engineering that consists in evaluating a set of specifications on a formal description of a program. When this program is "finite state", which happens in particular when considering only essential features of parallel and reactive systems, one of the practical approaches is to generate a *model* of this program, for instance a Labeled Transition System (Lts), describing its exhaustive behavior. Then, verification can be performed on this finite model, using appropriate decision procedures. This approach, usually named *model-checking*, can be fully automated and therefore gave rise to numerous verification tools ([Arn89,CPS89,RS90,Hol91,FGK$^+$96], etc.).

In spite of its apparent simplicity, one of the major drawback of model-checking is that the size of the model generated may exponentially grow up when considering large programs, and thus rapidly exceed the machine capabilities. Several solutions have been investigated to overcome this *state explosion problem*, for instance avoiding either an explicit storage of the whole model ("on-the-fly" techniques), or even its exhaustive generation ("reduced model generation" techniques).

A particular instance of this later solution consists in performing the verification not on the Lts S obtained from the initial program description, but rather on its *quotient* S/R where R is an equivalence relation preserving

[⋆] An extended version can be found in [KM97].

the properties under verification. Then, the main difficulty remains to obtain this quotient without generating first the initial LTS.

However, if the program under consideration can be described by a *composition expression* between communicating LTSs, and provided that R is a congruence with respect to the operators of this expression, the quotient S/R can be easily generated following a so-called *compositional approach* [Val96]: it consists in (repeatedly) generating the LTS S' associated to a given sub-expression, and replacing this sub-expression in the initial one by the quotient S'/R. Unfortunately, this technique is not so appealing in practice. In particular, the LTS S' may often contain lots of unnecessary execution sequences, forbidden by the synchronizations expected by the rest of the composition expression (its *environment*). In the worst cases, the size of S' may even exceed the one of S, leading to a failure of this approach.

A solution to this problem has been proposed in [GS90,GLS96] and [CK93,CK95] for composition expressions based on the CSP [Hoa78] parallel operator. Intuitively, it consists in expressing the environment of a sub-expression by an *interface*, i.e., an LTS representing a set of "authorized" execution sequences that can be performed by this sub-expression. Thus, using a *projection* operator, only a "restricted" LTS S' is generated, in which useless execution sequences have been cut off according to its corresponding interface.

The main objective of this work is to evaluate this compositional generation method on realistic case-studies, in order to compare its efficiency with respect to some other existing solutions for the state explosion problem. To this purpose, we have generalized the results of [GS90] and [CK93] to the LOTOS language [ISO88], an international ISO standard for the description of communication protocols. In particular a new projection operator – named *semi-composition* – has been defined, able to deal either with user-given interfaces (as in [GS90]), or with automatically computed ones (as in [CK93]). Then, compositional generation have been integrated within the CADP toolbox [FGK+96] and experimented on some of the large case-studies already carried out with this toolbox.

This paper is organized as follows. In section 1, we recall some basic definitions concerning LTSs and behavioral equivalences. In section 2, we present the general framework we used for compositional generation of LOTOS programs, and section 3 and 4 show how to perform this generation using either automatically computed or user-given interfaces. Finally, section 5 briefly presents our implementation and gives the results obtained on two different case-studies.

1 Preliminary definitions

The behavior of a (sequential) process can be modelized by a labeled transition system, namely, a set of states (the possible values of its program counter and local variables), and a labeled transition relation between states (each transition describing the execution of a given instruction).

More formally, let $\mathcal{Q}$ be a set of *states*, $\mathcal{A}$ a set of *label* (or instruction names), τ a particular label representing a *hidden* or *unobservable* instruction ($\tau \notin \mathcal{A}$), and let $\mathcal{A}_\tau = \mathcal{A} \cup \{\tau\}$. For a set X, X^* will represent the set of *finite sequences* on X.

Definition 1. A Labeled Transition System (LTS, for short) is a quadruplet $S = (Q, A, T, q_0)$ where Q is a finite set of *states* ($Q \subseteq \mathcal{Q}$), A is a finite set of *actions* ($A \subseteq \mathcal{A}_\tau$), T is a *transition relation* ($T \subseteq Q \times A \times Q$) and q_0 is a distinguished element of Q called the *initial state.* □

For an LTS $S = (Q, A, T, q_0)$, and for a given state p in Q, we adopt the following notations:

– The predicate $(p, a, q) \in T$ is noted $p \xrightarrow{a}_T q$ (or even $p \xrightarrow{a} q$).
  This notation is extended to label sequences: let $\lambda \subseteq A^*$, we write $p \xrightarrow{\lambda}_T q$ iff $\exists u_1 \cdots u_n \in \lambda \wedge \exists q_1, \cdots, q_{n-1} \in Q \wedge p \xrightarrow{u_1}_T q_1 \cdots q_{n-1} \xrightarrow{u_n}_T q$. Note that for each LTS considered in this paper, Q always matches with the set of states *reachable* by T from q_0: $Q = \{p \mid \exists \sigma \in A^* \,.\, q_0 \xrightarrow{\sigma}_T p\}$.
– $\mathcal{A}\mathrm{ct}(p)$ is the set of labels that can be performed from p:

$$\mathcal{A}\mathrm{ct}(p) = \{a \in A \mid \exists q \in Q \,.\, p \xrightarrow{a}_T q\}.$$

A rough characterization of a process behavior consists in considering the set of its observable execution sequences. For this purpose we define the language of an LTS as the set of finite observable label sequences that can be obtained from its initial state.

Definition 2. Let $S = (Q, A, T, q_0)$ be an LTS. For a given $p \in Q$, the (observable) language associated to p on S is defined as follows:

$$\mathcal{L}(p) = \{\sigma \mid \sigma = a_0 a_1 \cdots a_{n-1} \;\wedge\; p \overset{\tau^* a_0}{\longrightarrow} p_1 \overset{\tau^* a_1}{\longrightarrow} p_2 \cdots \overset{\tau^* a_{n-1}}{\longrightarrow} p_n\}$$

The (observable) language of S is then defined as the language associated to its initial state: $\mathcal{L}(S) = \mathcal{L}(q_o)$. □

Finally, we also introduce a particular operator, allowing to abstract away a given set of labels on an LTS by renaming them into the special τ label.

Definition 3. Let $S = (Q, A, T, q_0)$ be an LTS and G a set of label ($G \subseteq \mathcal{A}$). The *abstraction* $S[G]$ of S with respect to G is the LTS (Q, A', T', q_0) where

$$A' = \begin{cases} A & \text{if } A \subseteq G \\ (A \cap G) \cup \{\tau\} & \text{otherwise} \end{cases}$$

$$T' = \{(p, a, q) \mid (p, a, q) \in T \;\wedge\; a \in G\} \cup \{(p, \tau, q) \mid (p, a, q) \in T \;\wedge\; a \notin G\}.$$

□

Several equivalence relations have been proposed in the literature for comparing two LTSs. They mainly differ by the underlying behavior notion they are based on (e.g. a set of execution sequences versus an execution tree), together with the abstraction criterion used to handle the internal τ action.

We consider here a well-known family of behavioral equivalences, the *bisimulation* relations. First, we recall the general definition of these relations. Then we indicate how most of the classical bisimulations used in the verification framework are derived from this definition.

In the rest of the section we consider two LTSs $S_i = (Q_i, A_i, T_i, q_{0i})_{i=(1,2)}$ and Λ a family of disjoint languages on $\mathcal{A}_\tau^*$ ($\Lambda \subseteq 2^{\mathcal{A}_\tau^*}$).

Definition 4. For each relation $R \in Q_1 \times Q_2$, we define:

$$\begin{aligned}\mathcal{B}_\Lambda(R) = \{(p_1, p_2) \mid \forall \lambda \in \Lambda,\\ &(\forall q_1 \,.\, (p_1 \xrightarrow{\lambda}_{T_1} q_1 \;\Rightarrow\; \exists q_2 \,.\, (p_2 \xrightarrow{\lambda}_{T_2} q_2 \;\wedge\; (q_1, q_2) \in R))) \;\wedge\\ &(\forall q_2 \,.\, (p_2 \xrightarrow{\lambda}_{T_2} q_2 \;\Rightarrow\; \exists q_1 \,.\, (p_1 \xrightarrow{\lambda}_{T_1} q_1 \;\wedge\; (q_1, q_2) \in R)))\}\end{aligned}$$

The bisimulation equivalence $\sim^\Lambda$ for the language Λ is defined as the greatest fixed-point of $\mathcal{B}_\Lambda$. □

Following definition 4, LTSs S_1 and S_2 are said Λ-bisimilar (also noted $S_1 \sim^\Lambda S_2$) if and only if their initial states are related by $\sim^\Lambda$. This general definition allows to define most of usual bisimulation relations. The choice of a family Λ corresponds to the choice of an *abstraction criterion* on the labels: *strong bisimulation* $\sim$ ([Par81]) is obtained when $\Lambda = \{\{a\} \mid a \in A\}$, *observation equivalence* $\approx_o$ ([Mil80]) is obtained when $\Lambda = \tau^* \cup \{\tau^* a \tau^* \mid a \in A\}$, *delay bisimulation* $\approx_d$ ([NMV90]) is obtained when $\Lambda = \tau^* \cup \{\tau^* a \mid a \in A\}$, etc[1].

For each behavioral relation R, the *quotient* of a given LTS S with respect to R can be intuitively defined as the smallest LTS (in number of states) R-equivalent to S. Such a quotient will be noted S/R in the sequel. Moreover, for the bisimulation relations, the quotient of an LTS can be uniquely defined and computed rather efficiently for medium-sized LTSs (see for instance [PT87,KS90,Fer90,GV90]).

Finally, all these relations can be compared each other with respect to the inclusion, and thus ordered in the binary relation lattice. In particular, it is generally admitted that strong bisimulation is the finest relation for behavior comparison and it is therefore considered here as the "identity" relation between LTSs.

2 Compositional state space generation

We now turn back to the problem of generating the global LTS representing the behavior of a system of communicating processes. First, we present the

[1] *branching bisimulation* [vGW89], however, cannot be derived from this general definition.

language used throughout this paper to express such systems, and then we describe a compositional approach to perform the state space generation.

2.1 Composition expressions

The language we chose to describe systems of communicating processes is build from two LOTOS operators, namely the *parallel composition* operator ($||_G$)[2] and the *hiding* operator (`hide` G `in` $\ldots$), both parametrized by a label set G. Moreover, sequential *elementary processes* are represented by identifiers S.

The abstract syntax of *composition expressions* EXP is then the following:

$$\text{EXP} ::= \text{EXP} \ \ ||_G \ \ \text{EXP} \mid \texttt{hide}\ G\ \texttt{in}\ \text{EXP} \mid \text{S}$$

Composition expressions can be viewed as an intermediate form for the compilation of a LOTOS program into an LTS. More precisely, our objective is to express a LOTOS program as parallel compositions of many possible elementary processes. Although very simple, it will be shown in section 5 that the syntax proposed here fulfills this objective, and allows to deal with non trivial programs. Moreover, this syntax could easily be extended with other LOTOS operators (e.g. the so-called *enable* and *disable* operators, `>>` and `[>`) without significantly modifying the results presented in the following sections.

It now remains to define more precisely the parallel and hiding operators used in composition expressions.

Informally, $S_1 \ ||_G \ S_2$ is the LTS obtained by parallel composition of LTSs S_1 and S_2 with *rendez-vous* synchronization on the labels belonging to G. Transitions whose label does not belong to G are performed independently by the two LTSs according to the interleaving semantics.

Definition 5. Let $S_i = (Q_i, A_i, T_i, q_{0_i})_{i=(1,2)}$ be two LTSs, and G a label set ($G \subseteq \mathcal{A}$). Then, $S_1 \ ||_G \ S_2$ is the LTS $(Q, A_1 \cup A_2, T, (q_{0_1}, q_{0_2}))$ where $Q \subseteq Q_1 \times Q_2$ and T are the smallest sets verifying:

$$(q_{0_1},\ q_{0_2}) \ \in \ Q \qquad \text{[C0]}$$

$$\frac{(q_1,\ q_2) \in Q,\ q_1 \xrightarrow{a}_{T_1} q_1',\ q_2 \xrightarrow{a}_{T_2} q_2',\ a \in G}{(q_1',\ q_2') \ \in \ Q,\ (q_1,\ q_2) \xrightarrow{a}_{T} (q_1',\ q_2')} \qquad \text{[C1]}$$

$$\frac{(q_1,\ q_2) \in Q,\ q_1 \xrightarrow{a}_{T_1} q_1',\ a \notin G}{(q_1',\ q_2) \ \in \ Q,\ (q_1,\ q_2) \xrightarrow{a}_{T} (q_1',\ q_2)} \qquad \text{[C2]}$$

$$\frac{(q_1,\ q_2) \in Q,\ q_2 \xrightarrow{a}_{T_2} q_2',\ a \notin G}{(q_1,\ q_2') \ \in \ Q,\ (q_1,\ q_2) \xrightarrow{a}_{T} (q_1,\ q_2')} \qquad \text{[C3]}$$

[2] representing both the `||`, `|||` and `|[...]|` operators of LOTOS.

□

Note that this parallel composition operator is commutative but *not* associative in the general case (i.e., when the synchronization sets are not fixed).

For a given LTS S, $\texttt{hide}\ G\ \texttt{in}\ S$ is the LTS obtained from S by renaming each label belonging to G into the internal τ label.

Definition 6. Let $S = (Q, A, T, q_0)$ be an LTS, and G a label set ($G \subseteq \mathcal{A}$). $\texttt{hide}\ G\ \texttt{in}\ S$ is the LTS $S[\mathcal{A} \setminus G]$. □

The hiding operator can be partially distributed over parallel composition as follows:

$$\texttt{hide}\ H\ \texttt{in}\ (S_1 \,||_G\, S_2) \sim \texttt{hide}\ (H \cap G)\ \texttt{in}\ ((\texttt{hide}\ (H \setminus G)\ \texttt{in}\ S_1) \,||_G\, (\texttt{hide}\ (H \setminus G)\ \texttt{in}\ S_2))$$

Using definition 5 and 6 we are now able to give the semantics of a composition expression E in terms of LTSs. To this purpose we (inductively) define the function *sem* associating to each composition expression the LTS representing its behavior:

$$\begin{aligned} \mathrm{sem}\,(E_1 \,||_G\, E_2) &= \mathrm{sem}\,(E_1) \,||_G\, \mathrm{sem}\,(E_2) \\ \mathrm{sem}\,(\texttt{hide}\ G\ \texttt{in}\ E) &= \texttt{hide}\ G\ \texttt{in}\,\mathrm{sem}\,(E) \\ \mathrm{sem}\,(S) &= S. \end{aligned}$$

Finally, all the bisimulation relations mentioned in the previous section are *congruences* with respect to parallel composition and hiding operators. More precisely, it easy to check that if a language set Λ is such that each λ in Λ is included in $(\tau^* \cup \{\tau^* a \tau^* \mid a \in A\})$, then, for all LTSs S_1, S_2 and S:

$$\begin{aligned} S_1 \sim_\Lambda S_2 &\;\Rightarrow\; (S_1 \,||_G\, S) \sim_\Lambda (S_2 \,||_G\, S) \\ S_1 \sim_\Lambda S_2 &\;\Rightarrow\; (\texttt{hide}\ G\ \texttt{in}\ S_1) \sim_\Lambda (\texttt{hide}\ G\ \texttt{in}\ S_2) \end{aligned}$$

2.2 A compositional approach for state space generation

As already mentioned in the introduction, an automatic method to formally verify a system described by a composition expression E consists in generating the LTS $\mathrm{sem}\,(E)$, or, more efficiently, the quotient of this LTS with respect to a suitable equivalence relation R (where R is supposed to preserve the properties under verification).

Furthermore, instead of generating first the overall LTS $\mathrm{sem}\,(E)$ and then reducing it modulo relation R, using an incremental approach seems much more attractive. Such an approach can be sketched as follows: starting from the elementary processes S_i of E, $\mathrm{sem}\,(E)$ is obtained following a *bottom-up* strategy by replacing each sub-expression of E by the quotient modulo R of its associated LTS. This approach can be formalized as a recursive function

CompGen (for *Compositional Generation*), inductively defined in the following manner:

$$\begin{aligned}
\mathrm{CompGen}\,(E_1 \mathbin{||_G} E_2) &= (\mathrm{CompGen}\,(E_1) \mathbin{||_G} \mathrm{CompGen}\,(E_2))/R \\
\mathrm{CompGen}\,(\texttt{hide}\ G\ \texttt{in}\ E) &= (\texttt{hide}\ G\ \texttt{in}\ \mathrm{CompGen}\,(E))/R \\
\mathrm{CompGen}\,(S) &= (\mathrm{sem}\,(S))/R.
\end{aligned}$$

Provided that R is a congruence with respect to parallel composition and hiding operators, and provided that the quotient of an LTS modulo this relation is unique, it is clear that for any composition expression E, $\mathrm{CompGen}\,(E)$ is equal to $(\mathrm{sem}\,(E))/R$.

Unfortunately, this straightforward approach is not always sufficient in practice since unnecessary large intermediate LTSs may be generated. Indeed, sub-expressions are considered outside of their context (the remaining part of the initial composition expression), and therefore many constraints on their effective behavior are not taken into account during their generation. Consequently, the LTSs associated to such sub-expressions by function CompGen may contain lots of useless execution sequences forbidden by the context, that will disappear only in forthcoming parallel compositions [3].

A solution to this problem has been already formulated by [GS90] and [CK93] for a CSP-like parallel composition operator. Their approach can be summarized as follows:

– the context constraints of a sub-expression is a set of (allowed) execution sequences, and it can be represented by an LTS called the *interface*;
– LTSs generated from sub-expressions of E are "restricted" LTSs, in which forbidden execution sequences have been cut off according to their associated interface.

In order to formalize this solution in our framework, we first need to define more precisely the notion of *environment* of a sub-expression E' in a composition expression E. Intuitively, this is the set of parallel composition operations applied to E' in E (hiding operators are not included in the environment since only parallel compositions may restrict the behavior of a given sub-expression):

Definition 7. Let E be a composition expression. The set of sub-expressions of E is given by the function $\mathrm{SubExp} : \mathrm{Exp} \to 2^{\mathrm{Exp}}$, defined in the usual way:

$$\begin{aligned}
\mathrm{SubExp}\,(E_1 \mathbin{||_G} E_2) &= \{E_1 \mathbin{||_G} E_2\} \cup \mathrm{SubExp}\,(E_1) \cup \mathrm{SubExp}\,(E_2) \\
\mathrm{SubExp}\,(\texttt{hide}\ G\ \texttt{in}\ E) &= \{\texttt{hide}\ G\ \texttt{in}\ E\} \cup \mathrm{SubExp}\,(E) \\
\mathrm{SubExp}\,(S) &= \{S\}
\end{aligned}$$

[3] Note that this is particularly the case for programs written following the so-called *constraint oriented* specification style [VSSB91].

For any sub-expression E' of E, the environment of E' in E is given by the function $\text{Env} : \text{E}\textsc{xp} \times \text{E}\textsc{xp} \rightarrow 2^{\text{E}\textsc{xp} \times 2^{\mathcal{A}}}$, where

$$\begin{aligned}\text{Env}\,(E', E) = \{(E_i, G_i) \mid \exists E'_i, E''_i \in \text{SubExp}\,(E)\ .\\ ((E''_i = E'_i \mathbin{||_{G_i}} E_i) \ \vee\ (E''_i = E_i \mathbin{||_{G_i}} E'_i)) \ \wedge\ (E' \in \text{SubExp}\,(E'_i))\}\end{aligned}$$

$\square$

Using this definition we are able to take into account the context constraints within a compositional generation. More precisely, the basic idea is to replace each sub-expression E' of E by the Lts $\Psi(E', \text{Env}\,(E', E))$, where the transformation Ψ satisfies the following requirements:

R1: it *restricts* the behavior of E' according to its environment, i.e.,

$$\mid \Psi(E', \text{Env}\,(E', E)) \mid \ \leq \ \mid \text{sem}\,(E') \mid$$

R2: it *preserves* the behavior of the initial expression when a sub-expression E' is replaced by its corresponding Ψ-transformation, i.e.,

$$\text{sem}\,(E[\Psi(E', \text{Env}\,(E', E))/E']) \sim \text{sem}\,(E)$$

R3: it can be computed *on-the-fly*, i.e., $\Psi(E', \text{Env}\,(E', E))$ can be obtained without generating $\text{sem}\,(E')$ first.

Finally, it remains to propose a suitable transformation Ψ, satisfying the desired requirements. In [CK93], this transformation is built from the parallel composition operator itself. However, this solution requires to determinize first the interface Lts, which may exponentially increase its size. Moreover, even with a deterministic interface, requirement **R1** is not ensured. Regarding [GS90], transformation Ψ is built from a new operator, called the *projection*, able to restrict a composition expression even from a non deterministic interface. We adopt here this later approach, defining a similar operator in the framework of Lotos parallel composition.

3 Compositional generation under context constraints

In this section, we give first the definition of a new operator between Ltss named the *semi-composition*. Then, we show that it ensures all the requirements given in the previous section, and how to introduce it in a composition expression.

Definition 8. Let $S_i = (Q_i, A_i, T_i, q_{0_i})_{i=(1,2)}$ be two Ltss, and G a label set ($G \subseteq \mathcal{A}$).

Let $S' = (Q', A', T', q'_0)$ be the Lts $S_1 \mathbin{||_G} S_2$. We denote by $S_1 \rceil\!|_G\, S_2$ the Lts (Q, A_1, T, q_0) resulting from the semi-composition of S_1 by S_2 and defined as follows:

$$\begin{aligned}
&Q = \{(p_1, X) \mid p_1 \in Q_1 \;\wedge\; X = \{p_2 \mid (p_1, p_2) \in Q'\}\}\\
&T = \{((p_1, X_1), a, (p_2, X_2)) \mid\\
&\qquad (p_1, q_1) \xrightarrow{a}_{T'} (p_2, q_2) \;\wedge\; q_1 \in X_1 \;\wedge\; q_2 \in X_2 \;\wedge\; p_1 \xrightarrow{a}_{T_1} p_2\}\\
&q_0 = \{(q_{0_1}, X_0) \mid X_0 = \{p_2 \mid (q_{0_1}, p_2) \in Q'\}\}
\end{aligned}$$

For any $(p_1, X) \in Q$, set X is unique, which ensures the correction of this definition. $\square$

According to this definition, the LTS resulting from the semi-composition of S_1 by S_2 is clearly a sub-LTS of S_1. Consequently, the semi-composition never increases the number of states and transitions of its first operand, which ensures the requirement **R1**.

This semi-composition operator can be introduced in a composition expression in order to reduce the size of intermediate LTSs, as expressed by proposition 9 :

Proposition 9. *Let S_1 and S_2 be two* LTS*s and G a label set ($G \subseteq \mathcal{A}$). Then, the following relation holds:*

$$S_1 \;||_G\; S_2 \;\sim\; (S_1 \;\rceil|_G\; S_2) \;||_G\; S_2$$

$\square$

The second operand (S_2) of the semi-composition will represent the context constraints applied on the first operand and is named the *interface.* However, it is not necessary to use this whole LTS in order to restrict a sub-expression. Indeed, it can be performed by considering only the set of execution sequences of the interface defined on the synchronization set G (in particular the branching structure of the interface is irrelevant). Proposition 10 formalizes this property:

Proposition 10. *Let S_1 and S_2 be two* LTS*s and G a label set ($G \subseteq \mathcal{A}$). For any* LTS *S_2' such that $\mathcal{L}(S_2'[G]) = \mathcal{L}(S_2[G])$ we have:*

$$(S_1 \;\rceil|_G\; S_2) \sim (S_1 \;\rceil|_G\; S_2').$$

$\square$

Contrarily to the one considered in [GS90] and [CK93], the parallel composition operator we use is not associative. Consequently, the propagation of the semi-composition operator through the parallel composition needs a property of (partial) distribution:

Proposition 11. *Let S_1, S_2 and S be three* LTS*s and G a label set ($G \subseteq \mathcal{A}$). Then,*

$$(S_1 \;||_E\; S_2) \;||_G\; S \;\sim\; ((S_1 \;\rceil|_{G_1}\; S) \;||_E\; (S_2 \;\rceil|_{G_2}\; S)) \;||_G\; S$$

where label sets G_1 and G_2 are defined as follows:

$$G_1 = G \cap (E \cup (\mathcal{A}ct(S_1) \setminus \mathcal{A}ct(S_2)))$$
$$G_2 = G \cap (E \cup (\mathcal{A}ct(S_2) \setminus \mathcal{A}ct(S_1)))$$

□

Finally, we also use the following property in order to propagate context constraints through the hiding operator:

Proposition 12. *Let S_1 and S_2 be two* LTS*s and G_1 and G_2 two label sets. Then,*

$$(\texttt{hide } G_1 \texttt{ in } S_1) \,||_{G_2}\, S_2 \;\sim\; (\texttt{hide } G_1 \texttt{ in } (S_1 \,\rceil||_{G_2 \setminus G_1}\, S_2)) \,||_{G_2}\, S_2$$

□

It now remains to show more formally how this semi-composition operator can be used to implement the Ψ transformation.

In the general case, let E be a composition expression, E' a sub-expression of E, and (E_i, G_i) an element of Env (E', E). According to definition 7 there exists a sub-expression E''_i of E such that $E''_i \,||_{G_i}\, E_i$ (or $E_i \,||_{G_i}\, E''_i$) belongs to SubExp (E) and $E' \in$ SubExp (E''_i). Then, E' can be restricted up to (E_i, G_i) and we define:

$$\Psi(E', \{(E_i, G_i)\}) = E' \,\rceil||_{G'i}\, E'_i$$

where $\mathcal{L}(\text{sem}\,(E'_i)[G'_i]) = \mathcal{L}(\text{sem}\,(E_i)[G'_i])$ and G'_i is a subset of G_i, depending on the syntactic path between E''_i to E'. More precisely, $G'_i = \Phi_{E'}(E''_i, G_i)$ where function $\Phi_{E'}$ is inductively defined according to propositions 11 and 12 :

$$\Phi_{E'}(E', X) = X$$
$$\Phi_{E'}(E_1 \,||_G\, E_2, X) = \begin{cases} \Phi_{E'}(E_1, X \cap (G \cup \mathcal{A}\text{ct}(E_1) \setminus \mathcal{A}\text{ct}(E_2))) \\ \qquad\qquad\qquad\qquad \text{if } E' \in \text{SubExp}\,(E_1) \\ \Phi_{E'}(E_2, X \cap (G \cup \mathcal{A}\text{ct}(E_2) \setminus \mathcal{A}\text{ct}(E_1))) \\ \qquad\qquad\qquad\qquad \text{if } E' \in \text{SubExp}\,(E_2) \end{cases}$$
$$\Phi_{E'}(\texttt{hide } G \texttt{ in } E, X) = \Phi_{E'}(E, X \setminus G)$$

We have shown that the semi-composition operator can be used to build a Ψ-transformation verifying requirements **R1** and **R2** of the previous section, and that this transformation allows to restrict automatically a sub-expression according to a part of its environment. However, several problems live on. For instance:

- the interface (*i.e.* $sem(E'_i)[G'_i]$) has to be small enough to be generated;
- the semi-composition have to be restrictive (i.e. G'_i not empty);

– It is not always possible to restrict a sub-expression using its whole environment in a single (semi-composition) operation.

Consequently these results may be unsufficient in some practical cases. In the next section, we propose an alternative solution in which the user can express by himself the context constraints, and then (partially) avoid these problems.

4 Compositional generation with user given interfaces

The idea of using user-supplied interfaces to represent the context constraints associated to a sub-expression is not original: it is the basis of the work described in [GS90], and it has also been applied in [CK95]. However, our objective in this section is to show how this solution can be adapted to Lotos composition expressions, and to propose a general framework in which both user given and computed interfaces can be used.

The main problem arising when user given informations are used in a verification framework is to ensure that, even if such informations are erroneous, they cannot lead to an incorrect result. A practical way to solve this problem is therefore to try verifying these informations as well, and to conclude only when the answer is positive. To this purpose, we follow the approach proposed in [GS90][4]. Intuitively, this approach can be summarized as follows:

– if a sub-expression E' is restricted with respect to a user given interface, the synchronizations "refused" by this interface are recorded;
– when E' is composed with its "real" environment (the rest of the composition expression) it is easy to verify if these synchronizations really had to be refused.

To formalize this approach we need to extend the notion of Lts used so far by adding a binary predicate $\uparrow$. Its intuitive meaning is to associate to each state a label set for which a synchronization has been refused during the generation of this Lts, and such that this refusal has not been justified (yet).

Definition 13. An Extended Labeled Transition System (Elts for short) is a 5-tuple $(Q, A, T, q_0, \uparrow)$ where (Q, A, T, q_0) is an Lts and $\uparrow$ is a predicate over $Q \times A$.

In the following we note $p \uparrow a$ iff $(p, a) \in \uparrow$, and $p \uparrow \epsilon$ iff $\not\exists a \in \mathcal{A}_\tau \,.\, p \uparrow a$.

Moreover, an Elts S is said *valid*, and we note valid (S), iff it has not been obtained from unjustified refused synchronizations: $(\forall p \in Q \,.\, p \uparrow \epsilon)$. Consequently, "standard" Ltss can be simply viewed as valid Eltss. □

In the rest of the section we consider a label set $G \subseteq \mathcal{A}$ and two Eltss $S_i = (Q_i, A_i, T_i, q_{0_i}, \uparrow_i)_{i=(1,2)}$.

[4] and also in [CK95] using a different formalism.

From definition 13, we extend the parallel composition and hiding operators for ELTSs. In particular, since the parallel composition operator is used only to compose a sub-expression with a part of its "real" environment, then, for any action a belonging to the synchronization set G, $(q_1, q_2) \uparrow a$ holds iff $q_1 \uparrow a$ holds (an unjustified a-synchronization holds on state q_1) and $a \in \mathcal{A}\mathrm{ct}(q_2)$ (a is not refused by q_2), or vice-versa.

The exact definition of these operators is then the following:

Definition 14. $S_1 \,||_G\, S_2$ is the ELTS $(Q, A, T, q_0, \uparrow)$ where Q, A, T and q_0 are obtained from definition 5, and $\uparrow$ is the smallest set verifying:

$$\frac{q_1 \uparrow_1 a,\ a \notin G}{(q_1,\ q_2) \uparrow a} \qquad \frac{q_2 \uparrow_2 a,\ a \notin G}{(q_1,\ q_2) \uparrow a}$$

$$\frac{q_1 \uparrow_1 a,\ q_2 \xrightarrow{a}_{T_2} q_2',\ a \in G}{(q_1,\ q_2) \uparrow a} \qquad \frac{q_1 \xrightarrow{a}_{T_1} q_1',\ q_2 \uparrow_2 a,\ a \in G}{(q_1,\ q_2) \uparrow a}$$

$$\frac{q_1 \uparrow_1 a,\ q_2 \uparrow_2 a,\ a \in G}{(q_1,\ q_2) \uparrow a}$$

`hide` G `in` S_1 is the ELTS $(Q, A, T, q_0, \uparrow)$ where Q, A, T and q_0 are obtained from definition 6, and

$$\uparrow = \{(p_1, a) \mid p_1 \uparrow_1 a \ \wedge \ a \notin G\} \cup \{(p_1, \tau) \mid p_1 \uparrow_1 a \ \wedge \ a \in G\}$$

$\square$

Similarly, the semi-composition operator also has to be extended to ELTSs. Let us recall that this operator allows to restrict a sub-expression E', whose semantics is now expressed by an ELTS, with respect to a set of execution sequences – the interface – represented by a "standard" LTS and a synchronization set. Depending on the nature of this interface (i.e., user-supplied or automatically computed on the initial composition expression), we distinguish between two semi-composition operators:

– an "user one" (noted $\rceil|?$), which updates predicate $\uparrow$ on sem (E') by *adding* the labels corresponding to synchronizations refused by the interface;

– an "exact one" (still noted $\rceil|$), which updates predicate $\uparrow$ on sem (E') by *removing* the labels corresponding to synchronizations refused by the interface (this can be viewed as an anticipation, provided that E' will be composed with the part of the environment corresponding to this exact interface).

More formally, these operators are defined as follows:

Definition 15. $S_1 \rceil|_G S_2$ is the ELTS $(Q, A, T, q_0, \uparrow)$ where Q, A, T and q_0 are defined according to definition 8 and $\uparrow$ is obtained from $\uparrow_1$ as follows:

$$\uparrow = \{((p_1, X), a) \mid ((p_1, X) \in Q \wedge p_1 \uparrow_1 a) \wedge (a \in G \Rightarrow (\exists p \in X \,.\, a \in \mathcal{A}\mathrm{ct}(p)))\}$$

Similarly, $S_1 \rceil|?_G S_2$ is the ELTS $(Q, A, T, q_0, \uparrow)$ where Q, A, T and q_0 are defined according to definition 8 and $\uparrow$ is obtained as follows:

$$\uparrow = \{((p_1, X), a) \mid (p_1 \uparrow_1 a) \vee (a \in G \wedge a \in \mathcal{A}\mathrm{ct}(p_1) \wedge (\forall p_2 \in X \,.\, a \notin \mathcal{A}\mathrm{ct}(p_2)))\}$$

□

The validity of a compositional generation under user-given interfaces is established by proposition 16, which states that whenever an ELTS obtained from such an interface is *valid*, then this interface can be considered as correct.

Proposition 16. *Let S_1 and S_2 be two* ELTS*s, I an* LTS*, G and X two label sets and $S = (S_1 \rceil|?_X I) \,||_G\, S_2$.*

$$valid(S) \Rightarrow (S \sim S_1 \,||_G\, S_2)$$

□

More generally, according to proposition 16, if E is a composition expression and E' a sub-expression of E, then E' can be replaced in E by ELTS $(E' \rceil|?_X I)$ whenever the resulting ELTS is *valid*:

$$\mathrm{valid}\,(\mathrm{sem}\,(E[(E' \rceil|?_X I)/E'])) \Rightarrow (\mathrm{sem}\,(E[(E' \rceil|?_X I)/E']) \sim \mathrm{sem}\,(E))$$

In such case, $\Psi(E', \mathrm{Env}\,(E', E))$ is therefore simply expressed by $E' \rceil|?_X I$.

Finally, to obtain a practical approach for compositional generation, it also remains to extend the behavioral relations $\sim_\Lambda$ to ELTSs. Intuitively, these extensions $\sim_\Lambda^\uparrow$ must verify three properties: to preserve the original relation $\sim_\Lambda$ ($\sim_\Lambda^\uparrow \subseteq \sim_\Lambda$), to be a congruence with respect to (extended) operators of a composition expression, and to preserve the *valid* predicate between equivalent ELTSs.

It can be checked that the following extension of a Λ-bisimulation relation satisfies these criteria for the language sets Λ mentioned in section 1:

Definition 17. For each relation $R \in Q_1 \times Q_2$, we define:

$$\begin{aligned}\mathcal{B}_\Lambda^\uparrow(R) = \{(p_1, p_2) \mid \forall \lambda \in \Lambda,\\ &(\forall a \in \mathcal{A}_\tau \,.\, (p_1 \Uparrow a) \Leftrightarrow (p_2 \Uparrow a)) \wedge\\ &(\forall q_1 \,.\, (p_1 \xrightarrow{\lambda}_{T_1} q_1 \Rightarrow \exists q_2 \,.\, (p_2 \xrightarrow{\lambda}_{T_2} q_2 \wedge (q_1, q_2) \in R))) \wedge\\ &(\forall q_2 \,.\, (p_2 \xrightarrow{\lambda}_{T_2} q_2 \Rightarrow \exists q_1 \,.\, (p_1 \xrightarrow{\lambda}_{T_1} q_1 \wedge (q_1, q_2) \in R)))\}\end{aligned}$$

where $p \Uparrow a \equiv (\exists q \,.\, p \xrightarrow{\tau^*} q \wedge q \uparrow a)$. The extension $\sim_\Lambda^\uparrow$ of bisimulation equivalence $\sim_\Lambda$ is defined as the greatest fixed-point of $\mathcal{B}_\Lambda^\uparrow$. □

5 Application

This compositional generation method has been implemented within the Cadp toolbox and experimented on several Lotos programs. We briefly describe this implementation, and we give some experimental results.

5.1 The Cadp toolbox

Cadp (Cæsar/Aldébaran Development Package) is a toolbox for protocol engineering [FGK+96]. Its main functionality is to allow formal verification of both behavioral and logical specifications, following the model-based approach. This toolbox includes several components, and in particular:

– the Lotos compiler Cæsar, able to translate a Lotos program into an *explicit* Lts;
– the equivalence checker Aldébaran, able to compare or minimize Ltss up to various bisimulation relations.
– the Open-Cæsar environment, able to compile either a Lotos program or a composition expression into an *implicit* Lts (i.e., a set of C functions allowing an on-the-fly exploration of this Lts).

Two new components have been integrated in Cadp to allow compositional generation from Lotos programs:

– the Projector tool, implementing the semi-composition algorithms that can be found in [KM97], and developed within the Open-Cæsar environment;
– a compositional generation tool, which takes as input an equivalence relation and composition expression extended with semi-composition operators (see example ine section 5.2), and which generates an Unix *shell-script* containing the corresponding calls to Cæsar, Aldébaran and Projector.

5.2 Experimental results

We give the experimental results obtained when applying the compositional generation method on two realistic Lotos examples: an *atomic multicast protocol* [SE90], requiring user-given interfaces, and a *leader election algorithm* [GM96], that could be handled automatically.

The rel/Rel protocol [SE90] aims to support atomic communications between a transmitter and several receivers, in spite of an arbitrary number of failures from the stations involved in the communications. We focus here on a version of this protocol which preserves the order of the messages sent by the transmitter (its Lotos specification is given in [BM91]).

This protocol is built on a *transport* layer which provides a reliable message transmission between any pair of stations. In case of crash, stations are supposed to adopt a *fail-silent behavior*: they stop any message emission, and they silently discard any received message.

The rel/REL protocol is based on a *two phases commit* algorithm: the transmitter sends two successive copies of the message to all receivers; each message being uniquely identified, and an additional label indicates whether it is a first or a second copy. On receipt of a first copy, a station S waits for the second one. If it does not arrive before the expiration of a delay, then S assumes that the transmitter crashed and that some of the receivers may have not received any copy of the message. Then, S relays the transmitter and multicasts the two copies of the message, using itself the rel/REL protocol. However, to reduce the network traffic, a station stops to relay as soon as a second copy of the message is received from the transmitter or from any other receiver.

If we consider a transmitter `Trans`, and three receiving stations `Rec1`, `Rec2` and `Rec3` the composition expression derived from the LOTOS program describing this protocol is the following:

$$E = ((\texttt{Rec2}\ \ ||_{\{R23,R32\}}\ \texttt{Rec3})\ ||_{\{R12,R13,R21,R31\}}\ \texttt{Rec1})\ ||_{\{RT1,RT2,RT3\}}\ \texttt{Trans}$$

Note that the hiding operators have been omitted, since they are automatically distributed over parallel compositions by the compositional generation tool (see section 2.1).

A brutal application of the compositional generation method on this expression leads to several comments. First, the LTSs representing receivers $\texttt{Rec}_i$ are too large to be generated, and only LTS sem(`Trans`) can be obtained. Moreover, this later happens to be insufficient to restrict the receivers, i.e., LTS (sem($\texttt{Rec}_i$) $\rceil|_{RTi}$ sem(`Trans`)) is still too large. Therefore, user given interfaces $\mathcal{I}_i$ are necessary to express the constraints provided by the *whole* environment of each station $\texttt{Rec}_i$ (the transmitter *and* the other receivers). Finally, each parallel composition occurring in E is also systematically restricted with respect to the constraints provided by the transmitter. The resulting composition expression is then the following:

$$\begin{aligned} E' = (((((&\texttt{Rec2}\ \rceil|?_{\{RT2,R12,R32\}}\ \mathcal{I}_2)\ ||_{\{R23,R32\}}\ (\texttt{Rec3}\ \rceil|?_{\{RT3,R13,R23\}}\ \mathcal{I}_3))\\ &\rceil|_{\{RT2,RT3\}}\ \texttt{Trans})\ ||_{\{R12,R21,R13,R31\}}\ (\texttt{Rec1}\ \rceil|?_{\{RT1,R21,R31\}}\ \mathcal{I}_1))\\ &\rceil|_{\{RT1,RT2,RT3\}}\ \texttt{Trans})\ ||_{\{RT1,RT2,RT3\}}\ \texttt{Trans}\end{aligned}$$

Intuitively, interfaces $\mathcal{I}_i$ can be obtained by examining the constraints imposed by their environment on message sequences received by each station $\texttt{Rec}_i$. In particular, due to the message order preservation, one can assume that every messages are always received by the station in the order they have been sent. Then, using a global knowledge of the protocol, it becomes possible to write a LOTOS program describing a superset of such sequences for each station $\texttt{Rec}_i$, and thus to obtain suitable LTSs $\mathcal{I}_i$ [5].

When applying our compositional generation tool on expression E' the following intermediate LTSs are generated (each of them corresponding to a generation step):

$$\begin{aligned} S1_i &= \text{sem}\,(\texttt{Rec}_i) \;\rceil\!|?_{G_i}\; \mathcal{I}_i \\ S2 &= (S1_2 \;||_{\{R23,R32\}}\; S1_3) \;\rceil\!|_{\{RT2,RT3\}}\; \text{sem}\,(\texttt{Trans}) \\ S3 &= (S1_1 \;||_{\{R12,R21,R13,R31\}}\; S2) \;\rceil\!|_{\{RT1,RT2,RT3\}}\; \text{sem}\,(\texttt{Trans}) \\ \text{sem}\,(E') &= S3 \;||_{\{RT1,RT2,RT3\}}\; \text{sem}\,(\texttt{Trans}) \end{aligned}$$

The following table lists the size of these LTSs (in number of states and transitions), before and after reduction modulo strong bisimulation (remember that each LTS is systematically reduced after its generation), when three different messages are sent by the transmitter:

	before reduction		**after reduction**	
	states	transitions	states	transitions
$S1_i$	16694	108407	1121	15114
$S2$	95041	1284922	44195	551902
$S3$	854302	6144825	200795	1418989
sem (E')	898638	5893476	193991	1550623

According to these figures, none of the intermediate LTSs overcomes one million of states, and the resulting LTS sem $(E')/\sim$ is less than 200 000 states, which is quite manageable for verification purposes. Moreover, the whole generation process completed in a few hours on a SUN SS 20 workstation. The application on this same example of a symbolic generation method (based on a BDD encoding of the composition expression), leads to an LTS sem (E) containing about 200 million of states (represented itself by a BDD), obtained in one week of computations using the same workstation.

More generally, we summarize in the following table the results obtained for the two main applications we considered. We adopt here the terminology proposed in [GS90]: the "apparent size" of the application is the number of states of the LTS S obtained using a symbolic generation method, its "real size" is the number of states of S/R[6] and its "algorithmic size" is the number of states of the largest LTS generated using our compositional approach.

[5] Let us notice that these interfaces rely on a *correct* functioning of the protocol under verification, which may seem paradoxical. In fact, the *valid* predicate allows to justify *a posteriori* the correction of this hypothesis.

[6] where R is the "extended" strong bisimulation relation $\sim^{\uparrow}$ for the rel/REL protocol and the branching bisimulation for the Leader Election algorithm.

application	"apparent size"	"algorithmic size"	"real size"
rel/REL 2 stations	249 357	9 717	4 085
rel/REL 3 stations	178 519 776	898 638	193 991
Leader Election 4 stations	502 788 448	1 232	5
Leader Election 5 stations	?	45 760	6

It is quite clear on this two examples that compositional generation allows to largely avoid the "apparent complexity" of the program, and even to remain sometimes close to its "real complexity" as in the rel/REL example.

Conclusion

We have proposed in this paper a generalization of the results presented in [GS90] and [CK93] for applying a compositional generation method to LOTOS programs. Although many other works have been already carried out on compositional verification and compositional generation (an interesting classification can be found in [GLS96]), only a few of them – to our knowledge – have been applied to large examples in order to make a fair comparison with other "advanced" verification techniques.

The integration within the CADP toolbox of the compositional generation method described in this paper, and its evaluation on non-trivial case studies, have shown its interest in a verification framework. In particular, this approach allowed to significantly improve the capabilities of the toolbox for the two examples presented in this paper, providing better results than other verification methods implemented in CADP (such as on-the-fly verification and symbolic *minimal model generation* [FKM93]). Nevertheless, this is not true for all the examples we considered and this work still needs to be carried on.

First of all, it appears in practice that, even with a good knowledge of the program, it is not always possible for the user to provide suitable interfaces. Therefore, their automatic computation should be improved. A possible way could be to consider composition expression between LTSs extended with state variables, and thus making possible the use of some abstract interpretation techniques (since interfaces may not be necessarily represented by LTSs).

Besides, the Ψ transformation we consider preserves strong bisimulation (requirement **R2** in section 2). In fact, this requirement is too strong if the relation R under consideration is a coarsest relation (which is often the case in practice). Therefore, parametrizing this transformation with an equivalence relation could lead to further restrictions during the semi-composition, and thus reducing even more the size of intermediate LTSs.

Furthermore, the choice of a suitable strategy to decide which sub-expressions have to be dealt with during the compositional generation is also an important problem from the user point of view. Even if automatically providing an optimal strategy is certainly not manageable, some heuristics could be proposed to assist him.

Finally, compositional generation could also be extended to LTSs communicating with other mechanisms than *rendez-vous*, for instance such as *fifo* channels.

Acknowledgements: the authors are grateful to Susanne Graf for her numerous comments about a previous version of this work, and to the anonymous referees for their helpful suggestions.

References

[Arn89] André Arnold. MEC: A System for Constructing and Analysing Transition Systems. In Joseph Sifakis, editor, *Proceedings of the 1st Workshop on Automatic Verification Methods for Finite State Systems (Grenoble, France)*, volume 407 of *Lecture Notes in Computer Science*, pages 117–132. Springer Verlag, June 1989.

[BM91] Simon Bainbridge and Laurent Mounier. Specification and Verification of a Reliable Multicast Protocol. Technical Report HPL-91-163, Hewlett-Packard Laboratories, Bristol, U.K., October 1991.

[CK93] S.C. Cheung and J. Kramer. Enhancing Compositional Reachability Analysis with Context Constraints. In *Proceedings of the 1st ACM International Symposium on the Foundations of Software Engineering*, pages 115–125, Los Angeles, California, December 1993.

[CK95] S.C. Cheung and J. Kramer. Compositional Reachability Analysis of Finite-State Distributed Systems with User-Specified Constraints. In *Proceedings of SIGSOFT'95*, 1995.

[CPS89] R. Cleaveland, J. Parrow, and B. Steffen. The Concurrency Workbench. In J. Sifakis, editor, *Proceedings of the 1st Workshop on Automatic Verification Methods for Finite State Systems (Grenoble, France)*, volume 407 of *Lecture Notes in Computer Science*, pages 24–37. Springer Verlag, June 1989.

[Fer90] Jean-Claude Fernandez. An Implementation of an Efficient Algorithm for Bisimulation Equivalence. *Science of Computer Programming*, 13(2–3):219–236, May 1990.

[FGK$^+$96] J.C. Fernandez, H. Garavel, A. Kerbrat, R. Mateescu, L. Mounier, and M. Sighireanu. CADP: A Protocol Validation and Verification Toolbox. In Rajeev Alur and Thomas A. Henzinger, editors, *Proceedings of the 8th Conference on Computer-Aided Verification (New Brunswick, New Jersey, USA)*, August 1996.

[FKM93] J.C. Fernandez, A. Kerbrat, and L. Mounier. Symbolic Equivalence Checking. In C. Courcoubetis, editor, *Proceedings of the 5th Workshop on Computer-Aided Verification (Heraklion, Greece)*, volume 697 of *Lecture Notes in Computer Science*. Springer Verlag, June 1993.

[GLS96] S. Graf, G. Lüttgen, and B. Steffen. Compositional Minimisation of Finite State Systems using Interface Specifications. *Formal Aspects of Computation*, 3, 1996. appeared as Passauer Informatik Bericht MIP-9505.

[GM96] Hubert Garavel and Laurent Mounier. Specification and Verification of various Distributed Leader Election Algorithms for Unidirectional Ring Networks. *Science of Computer Programming*, 1996. Special issue on Industrially Relevant Applications of Formal Analysis Techniques. Full version available as INRIA Research Report 2986.

[GS90] Susanne Graf and Bernhard Steffen. Compositional Minimization of Finite State Processes. In *Workshop on Computer-Aided Verification*, Rutgers, USA, June 1990. DIMACS, R.P. Kurshan and E.M. Clarke.

[GV90] Jan Friso Groote and Frits Vaandrager. An Efficient Algorithm for Branching Bisimulation and Stuttering Equivalence. In M. S. Patterson, editor, *Proceedings of the 17th ICALP (Warwick)*, volume 443 of *Lecture Notes in Computer Science*, pages 626–638. Springer Verlag, 1990.

[Hoa78] C. A. R. Hoare. Communicating Sequential Processes. *Communications of the ACM*, 21(8):666–677, August 1978.

[Hol91] Gerard J. Holzmann. *Design and Validation of Computer Protocols.* Software Series. Prentice Hall, 1991.

[ISO88] ISO/IEC. LOTOS — A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour. International Standard 8807, International Organization for Standardization — Information Processing Systems — Open Systems Interconnection, Genève, September 1988.

[KS90] P. Kanellakis and S. Smolka. CCS Expressions, Finite State Processes and Three Problems of Equivalence. *Information and Computation*, 86(1), May 1990.

[Mil80] Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science.* Springer Verlag, 1980.

[NMV90] Rocco De Nicola, Ugo Montanari, and Frits Vaandrager. Back and Forth Bisimulations. CS R9021, Centrum voor Wiskunde en Informatica, Amsterdam, May 1990.

[Par81] David Park. Concurrency and Automata on Infinite Sequences. In Peter Deussen, editor, *Theoretical Computer Science*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer Verlag, March 1981.

[PT87] Robert Paige and Robert E. Tarjan. Three Partition Refinement Algorithms. *SIAM Journal of Computing*, 16(6):973–989, December 1987.

[RS90] Valérie Roy and Robert de Simone. Auto/Autograph. In R. P. Kurshan and E. M. Clarke, editors, *Proceedings of the 2nd Workshop on Computer-Aided Verification (Rutgers, New Jersey, USA)*, volume 3 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 477–491. AMS-ACM, June 1990.

[SE90] Santosh K. Shrivastava and Paul. D. Ezhilchelvan. rel/REL: A Family of Reliable Multicast Protocol for High-Speed Networks. Technical Report, University of Newcastle, Dept. of Computer Science, U.K, 1990.

[Val96] Antti Valmari. *Compositionality in State Space Verification.* In *Application and Theory of Petri Nets*, volume 1091 of *Lecture Notes in Computer Science*, pages 29–56. Springer Verlag, June 1996.

[vGW89] R. J. van Glabbeek and W. P. Weijland. Branching-Time and Abstraction in Bisimulation Semantics (extended abstract). CS R8911, Centrum voor Wiskunde en Informatica, Amsterdam, 1989. Also in proc. IFIP 11th World Computer Congress, San Francisco, 1989.

[VSSB91] C. Vissers, G. Scollo, M. van Sinderen, and E. Brinksma. On the use of specification styles in the design of distributed systems. *Theoretical Computer Science*, 89(1):179–206, October 1991.