COMPOSITIONALITY FOR IMPROVING MODEL CHECKING

Antonella Santone
Dipartimento di Ingegneria dell'Informazione, Università di Pisa, Italy
santone@iet.unipi.it

Abstract

Model checking is an automatic technique for verifying finite state systems: in this approach, properties are expressed in a temporal logic and systems are modelled as transition systems. A main problem of model checking is *state explosion*: very complex systems are often represented by transition systems with a prohibitive number of states. The primary cause of this problem is the parallel composition of interacting processes. Many techniques have been proposed to attack this problem, among them *compositional techniques*. These techniques reduce state explosion exploiting the natural decomposition of complex systems into processes. In this paper we present a formula-based compositional rule that allows us to deduce a property of a parallel composition of processes by checking it only on a component process.

Keywords: model checking, compositionality, temporal logic, state explosion.

1. INTRODUCTION

Model checking is an automatic technique for verifying finite state systems: in this approach, properties are expressed in a temporal logic and systems are modelled as transition systems. A model checker accepts two inputs, a transition system and a temporal formula, and returns "true" if the system satisfies the formula and "false" otherwise. Several efficient model checkers have been developed, see for example [5, 10]. A main problem of the model checking technique is *state explosion*: very complex systems are often represented by transition systems with a prohibitive number of states. The primary cause of this problem is the parallel composition of interacting processes. The problem occurs because the number of states of the global transition system is exponential in the number of component processes.

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35533-7_26

Tommaso Bolognesi and Diego Latella (Eds.), *Formal Methods for Distributed System Development.*

Many techniques have been proposed to attack this problem. While all these techniques cannot solve the problem in general, they provide significant improvements in performance. These techniques can be roughly divided into two categories. The works in the first category generate the whole transition system, corresponding to the concurrent system, but encode it *symbolically* to reduce the memory size [6, 18]. The works belonging to the second category avoid the generation of the whole transition system; local model checking [9, 23], on-the fly techniques [12, 14], abstractions [3, 7] and compositional reasoning [1, 8, 13, 15] belong to this category.

In our work we follow a compositional approach. Compositional techniques reduce state explosion exploiting the natural decomposition of complex systems into processes. Many finite state systems are composed of multiple processes running in parallel. The goal of this approach is to verify the processes individually and then piece together the results to conclude that the original system is correct.

In this paper we present a formula-based compositional rule that allows us to deduce a property of a parallel composition of processes by checking it only on a component process. Suppose that we want to show that a property φ holds for a system composed of two parallel processes P and Q, with synchronisation on the set of actions ω. Suppose that the transition system of the global system is too large to be handled by the existing verification environments. The first step of our approach consists of proving whether P (or equivalently Q) satisfies that property. This can be done by using standard model checkers, since the size of a single component is obviously less than that of the global system. If the first step succeeds, then we check that Q does not alter the satisfaction of φ when composed in parallel with P. This is done by checking a suitable relation between P and Q, based both on the formula φ and on the set ω of communication actions. If this second step succeeds too, then we can conclude that the original system satisfies φ, without generating the global transition system. This compositional method can significantly reduce the state explosion problem arising in the direct model checking method. Moreover it can be automated and it is completely transparent to the user. Other works based on compositional reasoning (see for example [15, 21]) require more user intervention, although they can achieve better results.

In this paper we use the LOTOS [4] language to specify concurrent and distributed systems, and a particular temporal logic, called *selective-μ-calculus*, to express properties. The selective-μ-calculus has been defined by the author and others in [2, 3]. We have chosen this logic since it is

particularly suited to be used in a compositional verification approach. However the method we present is general, since it is applicable, with suitable modifications, with other logic, for example standard modal μ-calculus; moreover it can be used with different formal specification languages.

The paper is organised as follows. Some notions related to LOTOS are recalled in Section 2, while the selective-μ-calculus is introduced in Section 3. Our compositional approach is described in Section 4. Considerations are made in Section 5, while comparisons with related works are given in Section 6.

2. BASIC LOTOS

Basic LOTOS is the version of LOTOS [4] without value-passing; thus it is used to describe synchronisation aspects of the system. We assume that the reader is familiar with Basic LOTOS, which is widely used in the specification of concurrent and distributed systems, and so we recall only some main concepts. The reader can refer to [4] for further details. From now on we write LOTOS instead of Basic LOTOS. A LOTOS program is defined as:

```
process ProcName := P
      where E
endproc
```

where P is a *process*, `ProcName := P` is a *process declaration* and $\mathcal{E}$ is a *process environment*, i.e. a set of process declarations. A process is the composition, by means of a set of operators, of a finite set $\mathcal{A} = \{i, a, b, ...\}$ of atomic *actions*. The action i is called the *unobservable action*. For the sake of simplicity, we do not consider all operators which are allowed in LOTOS. The simplified syntax of a LOTOS process is the following:

$$P ::= \texttt{stop} \mid \alpha; P \mid P[]P \mid P|[S]|P \mid \texttt{hide } S \texttt{ in } P \mid P[f] \mid X$$

where X ranges over a set of process names, α ranges over $\mathcal{A}$, $S \subseteq \mathcal{A} - \{i\}$ and $f : \mathcal{A} \to \mathcal{A}$ is an action relabelling function, with the property that $f(i) = i$. We call $\mathcal{P}$ the processes generated from P. By `stop` we denote the empty process. The operators to build processes are *action prefix* $(a; P)$, *choice* $(P_1[]P_2)$, *parallel composition* $(P_1|[S]|P_2)$, *hiding* (`hide` S `in` P), *relabelling*, $(P[f])$ and *process instantiation* (X).

Given a set $\mathcal{E}$ of process declarations, the standard *operational semantics* is given by a relation $\longrightarrow_{\mathcal{E}} \subseteq \mathcal{P} \times \mathcal{A} \times \mathcal{P}$. $\longrightarrow_{\mathcal{E}}$ ($\longrightarrow$ for short) is the least relation defined by the rules in Table 1. In Table 1 the symmetric rules for choice and parallel composition are not shown.

$\alpha \in \mathcal{A},\ l \in \mathcal{A} - \{i\}$

pre $\dfrac{}{\alpha; P \xrightarrow{\alpha} P}$ **choice** $\dfrac{P_1 \xrightarrow{\alpha} P_1'}{P_1 \,[]\, P_2 \xrightarrow{\alpha} P_1'}$

inst $\dfrac{P \xrightarrow{\alpha} P'}{X \xrightarrow{\alpha} P'}\ X := P \in \mathcal{E}$ **rel** $\dfrac{P \xrightarrow{\alpha} P'}{P[f] \xrightarrow{f(\alpha)} P'[f]}$

par $\dfrac{P_1 \xrightarrow{\alpha} P_1'}{P_1 \,|[S]|\, P_2 \xrightarrow{\alpha} P_1' \,|[S]|\, P_2}\ \alpha \notin S$

com $\dfrac{P_1 \xrightarrow{\alpha} P_1',\ P_2 \xrightarrow{\alpha} P_2'}{P_1 \,|[S]|\, P_2 \xrightarrow{\alpha} P_1' \,|[S]|\, P_2'}\ \alpha \in S$

hide$_1$ $\dfrac{P \xrightarrow{\alpha} P'}{\texttt{hide}\ S\ \texttt{in}\ P \xrightarrow{\alpha} \texttt{hide}\ S\ \texttt{in}\ P'}\ \alpha \notin S$

hide$_2$ $\dfrac{P \xrightarrow{l} P'}{\texttt{hide}\ S\ \texttt{in}\ P \xrightarrow{i} P'}\ l \in S$

Table 1 Standard operational semantics of Basic LOTOS

If $\delta \in \mathcal{A}^*$ and $\delta = \alpha_1 \ldots \alpha_n, n \geq 1$, we write $P \xrightarrow{\delta} Q$ to mean $P \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_n} Q$ and $\alpha_1, \ldots, \alpha_n$ is called a *finite computation* of P. For the empty sequence λ of actions we have $P \xrightarrow{\lambda} P$ for every $P \in \mathcal{P}$; clearly also λ is a finite computation of P.

A *labelled transition systems* $\mathcal{T}$ is a quadruple $(S, T, \longrightarrow, s_0)$, S is a set of states, T is a set of transition labels, $s_0 \in S$ is the initial state and $\longrightarrow \subseteq S \times T \times S$. The *standard transition system* for a LOTOS process P, denoted by $\mathcal{S}(P)$, is the transition system $(\mathcal{P}, \mathcal{A}, \longrightarrow, P)$. A LOTOS process P is finite, if the standard transition system for P is finite.

We now define a function that associates a process to a set of actions:

Definition 1 $(\mathcal{L}(P))$ *Let P be a LOTOS process and $\mathcal{E}$ be a set of process declarations. The sort of P with $\mathcal{E}$ is the set $\mathcal{L}_{\mathcal{E}}(P) \subseteq \mathcal{A}$ defined as*

the least solution of the following recursive definition:

$$
\begin{array}{lcl}
\mathcal{L}_{\mathcal{E}}(\texttt{stop}) & = & \emptyset \\
\mathcal{L}_{\mathcal{E}}(\alpha.P) & = & \begin{cases} \mathcal{L}_{\mathcal{E}}(P) \cup \{\alpha\} & \text{if } \alpha \neq i \\ \mathcal{L}_{\mathcal{E}}(P) & \text{if } \alpha = i \end{cases} \\
\mathcal{L}_{\mathcal{E}}(P[]Q) & = & \mathcal{L}_{\mathcal{E}}(P|[S]|Q) \quad = \quad \mathcal{L}_{\mathcal{E}}(P) \cup \mathcal{L}_{\mathcal{E}}(Q) \\
\mathcal{L}_{\mathcal{E}}(P[f]) & = & \{f(\alpha) : \alpha \in \mathcal{L}_{\mathcal{E}}(P)\} \\
\mathcal{L}_{\mathcal{E}}(\texttt{hide } S \texttt{ in } P) & = & \mathcal{L}_{\mathcal{E}}(P) - S \\
\mathcal{L}_{\mathcal{E}}(X) & = & \mathcal{L}_{\mathcal{E}}(P) \quad \text{if } X := P \in \mathcal{E}
\end{array}
$$

When clear from the context, we use $\mathcal{L}(P)$ in place of $\mathcal{L}_{\mathcal{E}}(P)$.

From now on, without loss of generality, we consider the parallel composition of two processes P and Q with synchronisation of the actions common to both of their sorts (P $|[\mathcal{L}(P) \cap \mathcal{L}(Q)]|$ Q).

3. SELECTIVE-μ-CALCULUS

The selective-μ-calculus, introduced by the author and others in [2, 3], is a branching temporal logic to express behavioural properties of systems. It is equi-expressive to μ-calculus [16, 22], but it differs from it in the definition of the modal operators. The basic characteristic of the selective-μ-calculus is that each formula guides the definition of a reduced transition system on which the formula can be equivalently checked.

The syntax of the selective-μ-calculus is the following, where K and R range over sets of actions, while Z ranges over a set of variables:

$$\varphi ::= \texttt{tt} \mid \texttt{ff} \mid Z \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid [K]_R \varphi \mid \langle K \rangle_R \varphi \mid \nu Z.\varphi \mid \mu Z.\varphi$$

The satisfaction of a formula φ by a state s of a transition system, written $s \models \varphi$, is defined as follows:

each state satisfies `tt` and no state satisfies `ff`; a state satisfies $\varphi_1 \vee \varphi_2$ ($\varphi_1 \wedge \varphi_2$) if it satisfies φ_1 or (and) φ_2. $[K]_R \varphi$ and $\langle K \rangle_R \varphi$ are the selective modal operators:

$[K]_R \varphi$ is satisfied by a state which, for every performance of a sequence of actions not belonging to $R \cup K$, followed by an action in K, evolves to a state obeying φ.

$\langle K \rangle_R \varphi$ is satisfied by a state which can evolve to a state obeying φ by performing a sequence of actions not belonging to $R \cup K$, followed by an action in K.

As in standard μ-calculus, a fix-point formula has the form $\mu Z.\varphi$ ($\nu Z.\varphi$) where μZ (νZ) *binds* free occurrences of Z in φ. An occurrence of Z is free if it is not within the scope of a binder μZ (νZ). A formula is *closed* if it contains no free variables. $\mu Z.\varphi$ is the least fix-point of the recursive equation $Z = \varphi$, while $\nu Z.\varphi$ is the greatest one.

A transition system $\mathcal{T}$ satisfies a formula φ, written $\mathcal{T} \models \varphi$, if and only if $P \models \varphi$, where P is the initial state of $\mathcal{T}$. A LOTOS process P satisfies φ if $\mathcal{S}(P)$ satisfies φ.

$P \not\models \mathtt{ff}$		
$P \models \mathtt{tt}$		
$P \models \varphi \wedge \psi$	iff	$P \models \varphi$ and $P \models \psi$
$P \models \varphi \vee \psi$	iff	$P \models \varphi$ or $P \models \psi$
$P \models [K]_R \varphi$	iff	$\forall P'.\forall \alpha \in K.P \stackrel{\alpha}{\Longrightarrow}_{K \cup R} P'$ implies $P' \models \varphi$
$P \models \langle K \rangle_R \varphi$	iff	$\exists P'.\exists \alpha \in K.P \stackrel{\alpha}{\Longrightarrow}_{K \cup R} P'$ and $P' \models \varphi$
$P \models \nu Z.\varphi$	iff	$P \models \nu Z^n.\varphi$ for all n
$P \models \mu Z.\varphi$	iff	$P \models \mu Z^n.\varphi$ for some n

where, for each n, $\nu Z^n.\varphi$ and $\mu Z^n.\varphi$ are defined as:

$$\nu Z^0.\varphi = \mathtt{tt} \qquad \mu Z^0.\varphi = \mathtt{ff}$$
$$\nu Z^{n+1}.\varphi = \varphi[\nu Z^n.\varphi/Z] \qquad \mu Z^{n+1}.\varphi = \varphi[\mu Z^n.\varphi/Z]$$

where the notation $\varphi[\psi/Z]$ indicates the substitution of ψ for every free occurrence of the variable Z in φ.

Table 2 Satisfaction of a closed formula by a state

The precise definition of the satisfaction of a closed formula φ by a state of a transition system $\mathcal{T} = (\mathcal{S}, \mathcal{A}, \longrightarrow, Q)$ is given in Table 2. It uses the transition relation $\Longrightarrow_I$, parametric with respect to $I \subseteq \mathcal{A}$, which ignores all non-interesting actions (i.e. those in $\mathcal{A} - I$).

Definition 2 ($\Longrightarrow_I$ relation) *Let P be a LOTOS process and $I \subseteq \mathcal{A}$, we define the relation $\Longrightarrow_I \subseteq \mathcal{P} \times I \times \mathcal{P}$ such that, for each $\alpha \in I$, $P, Q \in \mathcal{P}$:*

$$P \stackrel{\alpha}{\Longrightarrow}_I Q \quad \textit{iff} \quad P \stackrel{\delta\alpha}{\longrightarrow} Q, \textit{ where } \delta \in (\mathcal{A} - I)^*$$

By $P \stackrel{\alpha}{\Longrightarrow}_I Q$ we express the fact that it is possible to pass from P to Q by performing a (possibly empty) sequence of actions not belonging to I and then the action α in I. In the following, Q is called an *α-derivative of P with $\Longrightarrow_I$* (or *α-derivative* for short). Note that $\Longrightarrow_{\mathcal{A}} = \longrightarrow$.

Example 3 *Consider the LOTOS program in Table 3 that describes a coffee and tea dispenser in which the user first reads the instructions (action* `read_instr`*) and then he/she inserts a coin (action* `coin`*) if the* `Dispenser` *is ready to accept a coin (action* `ready`*). Afterwards, the user decides (action* `deciding`*) either to collect immediately a cup of coffee (action* `coffee`*); or to insert another coin before collecting, in case, a cup of tea (action* `tea`*). However, if the user does not insert the second coin quickly, i.e. he/she inserts the second coin after a timeout, internal to the* `Dispenser`*, (action* `timeout`*), the user is left without neither tea nor coffee.*

```
process System := Dispenser |[coin,coffee,tea]| User
    where E
endproc
```

$\mathcal{E}$ is composed of the following process declarations:

```
Dispenser := ready;coin;(timeout;(coin;Dispenser [] coffee;Dispenser)
                         []
                         coffee;Dispenser
                         []
                         coin;(tea;Dispenser [] Dispenser))

User := read_instr;coin;deciding;(coffee;User
                                  []
                                  coin;(User [] tea;User))
```

Table 3 A coffee and tea dispenser

$\mathcal{S}$(`System`) *has* 13 *states and* 22 *transitions, while* $\mathcal{S}$(`Dispenser`) *has* 5 *states and* 9 *transitions and* $\mathcal{S}$(`User`) *has* 5 *states and* 7 *transitions.*

To explain the use of the selective operators, we give some examples of selective-μ-calculus formulae, which are satisfied by `System`.

$\varphi_1 = \langle \texttt{tea} \rangle_{\{\texttt{timeout}\}}\, \texttt{tt}$: *"it is possible to get a cup of tea if the second coin has been inserted quickly, i.e. the timeout has not occurred".*

$\varphi_2 = [\texttt{coffee}]_{\{\texttt{coin}\}}\, \texttt{ff}$: *"a cup of coffee cannot be obtained if a coin has not been inserted".*

$\varphi_3 = \langle \texttt{coin} \rangle_{\{\texttt{coin}\}} \big(\langle \texttt{coffee} \rangle_{\emptyset} \texttt{tt} \wedge [\texttt{tea}]_{\{\texttt{coin}\}} \texttt{ff} \big)$: *"after the user has inserted a coin, it is possible to get a cup of coffee and a cup of tea cannot be collected if another coin has not been inserted".*

4. COMPOSITIONAL MODEL CHECKING

Compositional model checking is a method for reducing the complexity of temporal logic model checking in systems composed of many parallel processes. The goal is to check a property on a single component of a system and then deduce that the original system satisfies that property. In general, a property satisfied by a process could be not preserved when that process is composed with another process. For example: a; *stop* satisfies $\langle a \rangle_{\{b\}}$ tt, while a; *stop* $|[a]|$ b; a; *stop* does not satisfy $\langle a \rangle_{\{b\}}$ tt, since there exists no path from the initial state ending with a and not containing b. The problem is to find sufficient conditions under which a property of a process remains true in a parallel composition involving that process.

In this section we define a formula-based compositional rule that allows us to deduce a property of a parallel composition by checking it only on a parallel component. In particular, if we want to verify that a process of the form $P|[\omega]|Q$ has a property φ, we reduce this task to: 1) check whether P satisfies φ; and 2) check that Q does not alter the satisfaction of φ when composed in parallel with P, over the set of actions ω. To solve the first point we use the standard model checkers which we can found in all existing verification environments [5, 10]. To solve the second point we check whether P and Q are related by a suitable relation, based both on the formula φ and on the set of communication actions ω. A main point of this paper is the definition of this relation, which represents a sufficient condition for the preservation of the satisfaction of a formula.

For the sake of clarity, we first explore our approach considering selective-μ-calculus formulae containing only $\langle K \rangle_R$ modal operators, from now on referred to as $\langle\rangle s\mu$ formulae. We use the coffee and tea dispenser of Example 3 to explain our approach. Let us suppose that we want to check whether System satisfies the $\langle\rangle s\mu$ formula:

$$\varphi_1 = \langle \texttt{tea} \rangle_{\{\texttt{timeout}\}} \texttt{tt}.$$

It is easy to show that the process Dispenser satisfies φ_1. If we want to deduce that also the complete system (Dispenser $|[\omega]|$ User) satisfies φ_1 (with $\omega = \{\texttt{coin}, \texttt{coffee}, \texttt{tea}\}$) we must check that User does not alter the satisfaction of φ_1; this is ensured if:

- `User` does not block the execution of `Dispenser`: whenever `Dispenser` executes an action in ω, `User` can perform that action and so the parallel composition can perform the action `tea`.
- `User` does not execute `timeout` before `Dispenser` performs `tea`.

In general, if $P \models \varphi$, the actions relevant to preserve the satisfaction of a $\langle\rangle s\mu$ formula, when P is composed in parallel with another process, are both the communication actions and those ones belonging to the set $\mathcal{R}(\varphi)$ of all actions occurring in the set R of the modal operators $\langle K\rangle_R$ appearing in φ. In order to express the above conditions, we introduce the notion of $\langle\omega,\rho\rangle$-simulation.

Definition 4 ($\langle\omega,\rho\rangle$-simulation) *Let P, Q be two LOTOS processes and $\omega, \rho \subseteq \mathcal{A}$.*

- *A $\langle\omega,\rho\rangle$-simulation, $\mathcal{S}$, is a relation on $\mathcal{P} \times \mathcal{P}$ such that $P\ \mathcal{S}\ Q$ implies:*

$$P \stackrel{a}{\Longrightarrow}_{\omega} P' \text{ implies } Q \stackrel{a}{\Longrightarrow}_{\omega\cup\rho} Q' \text{ for some } Q' \text{ with } P'\ \mathcal{S}\ Q'.$$

- *We say that Q $\langle\omega,\rho\rangle$-simulates P (write $P \leq^{\omega}_{\rho} Q$) iff there exists a $\langle\omega,\rho\rangle$-simulation $\mathcal{S}$ containing the pair (P,Q).*

In other words, Q $\langle\omega,\rho\rangle$-simulates P means that, if P becomes the process P', after performing a sequence of actions which are not communication actions, followed by an action α in ω, then Q performs α, without executing an action in ρ, and becomes a process which $\langle\omega,\rho\rangle$-simulates P'. Note that Q can have more computations than P.

Let φ be a $\langle\rangle s\mu$ formula, we give the following compositional rule:

$$\langle\rangle\textbf{rule}\quad \frac{P \models \varphi, \quad P \leq^{\omega}_{\mathcal{R}(\varphi)} Q}{P\ |[\omega]|\ Q \models \varphi}$$

Recall the coffee and tea dispenser described in Example 3; it holds that:

- `Dispenser` $\models \varphi_1$: $\langle \texttt{tea}\rangle_{\{\texttt{timeout}\}}\ \texttt{tt}$;
- `Dispenser` $\leq^{\omega}_{\{\texttt{timeout}\}}$ `User` where $\omega = \{\texttt{coin}, \texttt{coffee}, \texttt{tea}\}$.

Thus we can deduce that `Dispenser` $|[\omega]|$ `User` $\models \varphi_1$.

With this approach we can check the formula only on $\mathcal{S}$(`Dispenser`). We remark that $\mathcal{S}$(`Dispenser`) has only 5 states, while $\mathcal{S}$(`System`) has 13 states. On the other hand, we have to determine whether it exists

a $\langle \omega, \{\texttt{timeout}\}\rangle$-simulation between **Dispenser** and **User**: we discuss this problem in Section 5. Note that also **User** satisfies φ_1, but **User** $\not\leq_{\rho}^{\omega}$ **Dispenser**, and so the $\langle\rangle$**rule** cannot be applied in this direction. Since the $\langle \omega, \rho\rangle$-simulation is not symmetric, if the application of the rule in one direction fails, we can apply the rule in the other direction. The choice of the direction is made arbitrarily, but some remarks concerning the sorts of the processes can be useful. For example, if we want to check whether $P \; |[\omega]| \; Q$ satisfies a formula φ, containing the modal operator $\langle\alpha\rangle_R$, and $\alpha \notin \mathcal{L}(P)$, while $\alpha \in \mathcal{L}(Q)$, then we clearly choose Q to check φ on it and then we verify that P does not alter the satisfaction of φ (i.e. $Q \leq_{\mathcal{R}(\varphi)}^{\omega} P$).

Finally, note that the above compositional rule is not complete; in fact it is possible that a process P satisfies a $\langle\rangle$sμ formula φ, Q does not $\langle \omega, \mathcal{R}(\varphi)\rangle$-simulates P, but $P \; |[\omega]| \; Q$ satisfies φ.

Let us now consider selective-μ-calculus formulae containing only $[K]_R$ modal operators, in the following referred to as []sμ formulae. Recall the coffee and tea dispenser of Example 3. Let us suppose that we want to check whether **System** satisfies the []sμ formula:

$$\varphi_2 = [\texttt{coffee}]_{\{\texttt{coin}\}} \, \texttt{ff}.$$

It is easy to show that the process **Dispenser** satisfies φ_2. If we want to deduce that also

$$\texttt{Dispenser} \; |[\{\texttt{coin}, \texttt{coffee}, \texttt{tea}\}]| \; \texttt{User} \models \varphi_2$$

we must check that **User** does not alter the satisfaction of φ_2. A sufficient condition to ensure this fact is that **User** and **Dispenser** behave in the same way with respect to the action **coffee**.

In general, if $P \models \varphi$ and φ is a []sμ formula, the actions relevant to preserve the satisfaction of φ, when P is composed in parallel with another process, are only those ones belonging to the set $\mathcal{K}(\varphi)$ of all actions occurring in the set K of the modal operators $[K]_R$ appearing in φ. In order to express the above condition, we introduce the notion of σ-bisimulation.

Definition 5 (σ-bisimulation) *Let P, Q be two LOTOS processes and $\sigma \subseteq \mathcal{A}$.*

- *A σ-bisimulation, $\mathcal{B}$, is a relation on $\mathcal{P} \times \mathcal{P}$ such that $P \; \mathcal{B} \; Q$ implies:*

- $P \stackrel{\alpha}{\Longrightarrow}_\sigma P'$ *implies* $Q \stackrel{\alpha}{\Longrightarrow}_\sigma Q'$ *for some* Q' *and if* $Q \stackrel{\alpha}{\Longrightarrow}_\sigma Q'$ *then* $P' \; \mathcal{B} \; Q'$; *and*
- $Q \stackrel{\alpha}{\Longrightarrow}_\sigma Q'$ *implies* $P \stackrel{\alpha}{\Longrightarrow}_\sigma P'$ *for some* P' *and if* $P \stackrel{\alpha}{\Longrightarrow}_\sigma P'$ *then* $P' \; \mathcal{B} \; Q'$.

- *We say that* P *and* Q *are* σ*-bisimilar (write* $P \sim_\sigma Q$*) iff there exists a* σ*-bisimulation* $\mathcal{B}$ *containing the pair* (P, Q).

$P \sim_\sigma Q$ means that, if P performs an action in σ and becomes the process P', then also Q can perform the same action and becomes a process which is σ-bisimilar to P'. Moreover, each α-derivative of P is σ-bisimilar to all α-derivatives of Q. Similarly with P and Q interchanged.

Suppose that P satisfies a $[]\mathrm{s}\mu$ formula φ. Note that the actions belonging to the set R of the modal operators $[K]_R$ occurring in φ are not relevant to preserve the satisfaction of φ, when P is composed in parallel with another process Q. In fact a computation containing an action in K but preceded by an action in R does not alter the truth value of a formula. Further, also the communication actions are not relevant to preserve the satisfaction of φ: if P performs an action α in K and Q blocks the execution of such action, then the parallel composition cannot perform α and consequently the formula is vacuously true.

Let φ be a $[]\mathrm{s}\mu$ formula, we give the following compositional rule:

$$[]\textbf{rule} \quad \frac{P \models \varphi, \quad Q \sim_{\mathcal{K}(\varphi)} P}{P \; |[\omega]| \; Q \models \varphi}$$

Recall the coffee and tea dispenser described in Example 3; it holds that:

- $\texttt{Dispenser} \models \varphi_2 = [\texttt{coffee}]_{\{\texttt{coin}\}} \, \texttt{ff}$;
- $\texttt{User} \sim_{\{\texttt{coffee}\}} \texttt{Dispenser}$.

Thus we can deduce that $\texttt{Dispenser} \; |[\{\texttt{coin}, \texttt{coffee}, \texttt{tea}\}]| \; \texttt{User} \models \varphi_2$.

Now we consider the full selective-μ-calculus formulae. Again, if we want to check that $P \; |[\omega]| \; Q \models \varphi$, we check φ on P and we check whether P and Q are related by a $\langle \omega, \varphi \rangle$-relation, which, in this case, is obtained combining $\langle \omega, \rho \rangle$-simulation with σ-bisimulation in the following way:

Definition 6 ($\langle \omega, \rho, \sigma \rangle$-relation) *Let* P, Q *be two LOTOS processes and* $\omega, \rho, \sigma \subseteq \mathcal{A}$.

- *A $\langle \omega, \rho, \sigma \rangle$-relation, $\mathcal{R}$, is a relation on $\mathcal{P} \times \mathcal{P}$ such that $P\ \mathcal{R}\ Q$ implies:*
  - $P \stackrel{\alpha}{\Longrightarrow}_{\omega} P'$ *implies* $Q \stackrel{\alpha}{\Longrightarrow}_{\omega \cup \rho} Q'$ *for some* Q' *with* $P'\ \mathcal{R}\ Q'$; *and*
  - $P \stackrel{\alpha}{\Longrightarrow}_{\sigma} P'$ *implies* $Q \stackrel{\alpha}{\Longrightarrow}_{\sigma} Q'$ *for some* Q' *and if* $Q \stackrel{\alpha}{\Longrightarrow}_{\sigma} Q'$ *then* $P'\ \mathcal{R}\ Q'$; *and*
  - $Q \stackrel{\alpha}{\Longrightarrow}_{\sigma} Q'$ *implies* $P \stackrel{\alpha}{\Longrightarrow}_{\sigma} P'$ *for some* P' *and if* $P \stackrel{\alpha}{\Longrightarrow}_{\sigma} P'$ *then* $P'\ \mathcal{R}\ Q'$.
- *We say that P $\langle \omega, \rho, \sigma \rangle$-relates Q iff there exists a $\langle \omega, \rho, \sigma \rangle$-relation $\mathcal{R}$ containing the pair (P, Q).*

Let φ be a selective-μ-calculus formula and $\omega \subseteq \mathcal{A}$, in the following we say that two processes P and Q are related by a $\langle \omega, \varphi \rangle$-relation (write $P \bowtie^{\omega}_{\varphi} Q$) if and only if a $\langle \omega, \rho, \sigma \rangle$-relation exists containing the pair (P, Q), where $\mathcal{R}(\varphi) = \rho$ and $\mathcal{K}(\varphi) = \sigma$.

We can now give the complete compositional rule:

$$\textbf{compositional\_rule} \quad \frac{P \models \varphi, \quad P \bowtie^{\omega}_{\varphi} Q}{P\ |[\omega]|\ Q \models \varphi}$$

Recall the coffee and tea dispenser described of Example 3. Let us suppose that we want to check whether `System` satisfies the selective-μ-calculus formula:

$$\varphi_3 = \langle \texttt{coin} \rangle_{\{\texttt{coin}\}} \left(\langle \texttt{coffee} \rangle_{\emptyset}\, \texttt{tt} \wedge [\texttt{tea}]_{\{\texttt{coin}\}}\, \texttt{ff} \right).$$

Applying the compositional rule, we can deduce that the property holds on the complete system by checking such property only on `Dispenser` and by showing that `Dispenser` $\bowtie^{\omega}_{\varphi_3}$ `User`.

We now present the main result of the paper, stating the soundness of the compositional rule.

Theorem 1 (main) *Let P, Q be two LOTOS processes and φ a formula of the selective-μ-calculus.*

$$P \models \varphi \ \textit{ and } \ P \bowtie^{\omega}_{\varphi} Q \ \textit{ implies } \ P\ |[\omega]|\ Q \models \varphi$$

Proof. By induction on the structure of the formula.

The soundness of $\langle\rangle$**rule** and $[]$**rule** follows by Theorem 1.

5. DISCUSSION

The compositional rule described in the previous section can significantly reduce the state explosion problem arising in the direct model checking method, since we can check a formula φ only on a parallel component. However, to complete the verification, we have to check whether there exists a $\langle \omega, \varphi \rangle$-relation between two processes P and Q composed in parallel over the set of actions ω. Thus, a crucial point is to find an efficient algorithm to determine whether $P \bowtie_{\varphi}^{\omega} Q$. In literature, efficient algorithms exist for computing bisimulation equivalences [19], among them there is the algorithm of Paige-Tarjan [20]. We can adapt this algorithm to compute the $\langle \omega, \varphi \rangle$-relation. In the worst case, we obtain an algorithm whose complexity, in space, is $\mathcal{O}(n + m)$, where n is the size of the transition system for P and m is the size of the transition system for Q. As a consequence, we obtain a reduction of the state space since, in the worst case, the size of the parallel composition of P and Q may be equal to the product of the sizes of the two components.

It is important to note that the compositional approach presented in this paper can be applied to standard μ-calculus too. As proved in [3], it holds that: $\langle K \rangle \varphi = \langle K \rangle_{\mathcal{A}} \varphi$ and $[K] \varphi = [K]_{\mathcal{A}} \varphi$. As a consequence, we can apply our compositional rule, taking into account that, given a μ-calculus formula φ, $\mathcal{R}(\varphi)$ coincides with the whole set $\mathcal{A}$ if it exists a $\langle\rangle$ operator in φ. Moreover, our approach can handle also infinite processes and we are investigating how the method could work for systems composed of more than two processes.

We are actually developing a tool to check whether there exists a $\langle \omega, \varphi \rangle$-relation between two processes P and Q composed in parallel over the set of actions ω. The tool can be included, for example, in the the Concurrency Workbench environment [10], which is a verification system for process algebra description languages.

6. RELATED WORK

We now compare our method with some related works based on compositional reasoning. In [8] a technique based on the use of *interface processes* is proposed. This technique attempts to minimise the global transition system by focusing on the communication among the component processes. The method considers the set of actions used in the interface between two components and minimises the system by eliminating events that are not related to communication actions. More precisely, assume that P_1 and P_2 communicate using a set of actions ω. Then P_1 can only observe the behaviour of P_2 through ω. Thus P_2 can

be replaced by an equivalent process A_2 (called the *interface process for* P_2) which is indistinguishable from P_2 with respect to ω. Our approach differs from the above one since for several reasons, among them: suppose that ω is almost the whole set $\mathcal{L}(P_2)$, the method proposed in [8] does not improve the efficiency of model checking, since the size of A_2 is not significantly less than the size of P_2. Instead our approach can be applied also in this situation, reducing in many cases the complexity of model checking. Moreover, in [8], after finding the interface process, the model checking is always applied to a parallel composition of two processes (although less than the original one) which is the main cause of state explosion; this is avoided in the solution proposed in this paper. On the other hand, the method proposed in [8] can achieve better results than ours, when the set of communication actions is small.

Assume-guarantee reasoning [21] is a semi-automatic technique that verifies each component separately. The behaviour of each component depends on the behaviour of the rest of the system, i.e. its environment. Because of this, the user must specify properties that the environment has to satisfy in order to guarantee the correctness of the component. These properties are *assumed.* If these assumptions are satisfied, the component will satisfy other properties, called *guarantees.* By combining the set of assume/guarantee properties in an appropriate way, it is possible to demonstrate the correctness of the entire system without constructing the global transition system. In order to automate this type of reasoning, in [13] a simulation preorder has been defined which preserves the satisfaction of formulae of a subset of CTL [11]. Our approach differs from the above one since that one uses a subset of a general logic (CTL) and cannot be applied to infinite state systems. Moreover, in the assume-guarantee approach the state explosion problem is replaced with another problem: the *assumption explosion problem*, i.e. assume/guarantee properties are expressed by complex formulae. However, for some kind of formulae, the method described in [13] is more efficient than ours.

In [15] it is proved that, for a given property φ that a system composed of two concurrent processes must satisfy, there exist properties φ_1, φ_2 that the processes in the composition must satisfy. Nevertheless, no method is given to deduce such properties; this is full responsibility of the human verifier of the system. Our approach differs from the above one since it can be completely automated.

In [1] the parallel composition operator was eliminated basically by encoding one of the component of the parallel composition into the formula, which is checked on the other components. In the worst case this re-

sults in an exponential blow-up in the size of the formula, and the total complexity remains the same as for non-compositional model checking.

We remark that, unlike all other approaches discussed above, the effectiveness of our method depends on the efficiency of the algorithm to decide whether two parallel processes are related by a $\langle \omega, \varphi \rangle$-relation.

Acknowledgments

The author wish to thank Nicoletta De Francesco and Gigliola Vaglini for their useful comments and suggestions.

References

[1] H.R. Andersen, G. Winskel. Compositional Checking of Satisfaction. *Formal Methods in System Design, 1(4)*, 1992.

[2] R. Barbuti, N. De Francesco, A. Santone, G. Vaglini. Selective Mu-Calculus: New Modal Operators for Proving Properties on Reduced Transition Systems. In *Proceedings of FORTE X/PSTV XVII '97*. Chapman & Hall, 1997. 519-534

[3] R. Barbuti, N. De Francesco, A. Santone, G. Vaglini. Selective Mu-Calculus and Formula-Based Abstractions of Transition Systems. *Journal of Computer and System Sciences, 59(3)*, 1999. 537-556.

[4] T. Bolognesi, E. Brinksma. Introduction to ISO Specification Language LOTOS. *Comp. Networks and ISDN Systems, 14*, 1987. 25-59.

[5] A. Bouali, S. Gnesi, S. Larosa. The integration Project for the JACK Environment. *Bulletin of the EATCS, 54*, 1994. 207-223.

[6] J. Burch, E. Clarke, K. McMillan, D. Dill, L. Hwang. Symbolic Model Checking: 10^{20} States and Beyond. In *Proceedings of Fifth Annual IEEE Symposium on Logic in Computer Science*, 1990. 428-439.

[7] E.M. Clarke, O.Grumberg, D.E. Long. Model Checking and Abstraction. *ACM Transactions on Programming Languages and Systems, 16(5)*, 1994. 1512-1542.

[8] E.M. Clarke, D.E. Long, K.L. McMillan. Compositional Model Checking. In *Proceedings of the Fourth Annual IEEE Symposium on Logic in Computer Science*, 1989. 353-362.

[9] R. Cleveland. Tableau-based Model Checking in the Propositional Mu-Calculus. *Acta Informatica, 27*, 1990. 725–747.

[10] R. Cleaveland, S. Sims. The NCSU Concurrency Workbench. In *Proceedings of the Eighth International Conference on Computer-*

Aided Verification (CAV'96), Lecture Notes in Computer Science 1102, 1996. 394–397.

[11] E.A. Emerson, J.Y. Halpern. "Sometimes" and "Not Never" Revisited: On Branching Time Versus Linear Time. *Journal of ACM, 33(1)*, 1986. 151–178.

[12] J.C. Fernandez, L. Mounier. Verifying Bisimulation on the fly. In *Proceedings of the Third International Conference on Formal Description Techniques, FORTE'90*, 1990.

[13] O. Grumberg, D.E. Long. Model Checking and Modular Verification. *ACM Transactions on Programming Languages and Systems, 16(3)*, 1994. 843-871.

[14] C. Jard, T. Jéron. Bounded-memory Algorithms for Verification on-the-fly. In *Proceedings of the Third International Conference on Computer-Aided Verification (CAV'91)*, Lecture Notes in Computer Science 575, 1991. 192-201.

[15] R. Kaivola. Compositional Model Checking for Linear-Time Temporal Logic. In *Proceedings of the Fourth International Conference on Computer-Aided Verification (CAV'92)*, Lecture Notes in Computer Science 663, 1991. 248–259.

[16] D. Kozen. Results on the Propositional Mu-Calculus. *Theoretical Computer Science, 27*, 1983. 333–354.

[17] E. Madelaine, D. Vergamini. Finiteness Conditions and Structural Construction of Automata for all Process Algebras. In *Proceedings of 2nd Workshop on Computer-Aided Verification. DIMACS Technical Report 90-31*, 1990.

[18] K. McMillan. *Symbolic Model Checking.* Boston: Kluwer Academic Publishers, 1993.

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

[20] R. Paige, R. Tarjan. Three Partition Refinement Algorithms. *SIAM Journal of Computing, 16(6)*, 1987. 973–989.

[21] A. Pnueli. In Transition for Global to Modular Temporal Reasoning about Programs. *Logics and Models of Concurrent Systems.* NATO ASI Series. Series F, Computer and System Sciences, 13. Springer-Verlag, 1984.

[22] C. Stirling. An Introduction to Modal and Temporal Logics for CCS. In *Concurrency: Theory, Language, and Architecture*, Lecture Notes in Computer Science 391, 1989.

[23] C. Stirling, D. Walker. Local Model Checking in the Modal Mu-Calculus. *Theoretical Computer Science, 89*, 1991. 161-177.