"On the Fly" Verification of Behavioural Equivalences and Preorders

Jean-Claude Fernandez Laurent Mounier
IMAG–LGI BP53X 38041 GRENOBLE Cedex

Abstract

This paper describes decision procedures for bisimulation and simulation relations between two transition systems. The algorithms proposed here do not need to previously construct them: the verification can be performed during their generation. In addition, a diagnosis is computed when the two transitions systems are not equivalent.

1 Introduction

One of the successful approaches used for the verification of systems of communicating processes is provided by behavioral equivalence and preorder relations, which allow to compare different descriptions of a given system. More precisely, if we note S (Specification) the most abstract description of the system and I (Implementation) the most detailed one, it is possible to check whether I is in fact an implementation of S in the following manner: from S and I, generate two Labeled Transition Systems (LTS for short) S_1 and S_2. Let R be an appropriate equivalence relation or preorder relation on LTS. Then, I implements S if and only if $S_1 R S_2$.

Among the different equivalence relations which have been proposed, *bisimulations* appear to be the most attractive ones: these equivalences have a suitable semantics, are well defined, and for each of them a normal form exists which is minimal in number of states and transitions. An efficient algorithm [PT87] allows to compute the normal form of a LTS S for the strong bisimulation relation. This algorithm consists in refining a partition of its states until it becomes "compatible" with its transition relation. If n is the number of states of S, and m is the cardinality of its transition relation, then the time requirement for this algorithm is $O(m \log(n))$. Thus, an efficient decision procedure for the equivalence of two transition systems consists in computing the normal form of the union of the LTS.

Other equivalence relations are based on simulation preorders like *safety equivalence* [Rod88], which characterizes exactly safety properties [BFG*91]. In this case, I implements S if and only if $S_1 R S_2$ **and** $S_2 R S_1$. A decision procedure for safety equivalence is based on the Paige & Tarjan algorithm [Fer89].

However, the main drawback of these methods is that the whole LTS have to be stored (i.e, the sets of states and transitions). Consequently, the size of the graphs which can be compared is limited, and this limit is easily reached when verifying real examples.

In this paper we extend the decision procedure for bisimulation equivalence relation, presented in [FM90], to simulation based equivalence or preorder. In fact, we show that it is sufficient to define a particular synchronous product between two LTS parametrized by a simulation or a bisimulation. Thus, the

verification can be done during the process of the two transition systems ("on the fly" verification). In addition, in the case where two LTS are not comparable under the relation R, we produce as a *diagnosis* an execution sequence which leads in a *failure state*. This approach is similar to the one proposed in [JJ89], [BFH90] and [CVWY90], which deals with "on the fly" verification of linear temporal logic properties.

A version of our algorithm for a weaker bisimulation, for safety equivalence and for simulation preorder have been implemented in the tool ALDÉBARAN which allows to compare and reduce LTS with respect to several equivalence relations (strong bisimulation, observational equivalence [Mil80], acceptance model equivalence [GS86] and safety equivalence).

The paper is organized as follows: in section 2 we give the definitions used in the following pages, in section 3 the verification method for simulations and bisimulations is described, in section 4 we give the algorithm, and in section 5 we show how it can be adapted to provide a diagnostic. The results obtained when applying the usual algorithm and our improved one are also compared in this section.

2 Definitions

2.1 Labeled Transition Systems

Let *States* be a set of states, A a set of names (of actions), and τ a particular name of A, which represents an internal or hidden action. For a set X, X^* will represent the set of finite sequences on X.

Definition 2.1 *A LTS is a tuple $S = (Q, A, T, q_0)$ where: Q is the subset of States reachable from q_0 with respect to T, A is a set of actions (or labels), $T \subseteq Q \times A \times Q$ is a labeled transition relation, and q_0 is the initial state.*

For each label a and each state q, we consider the image set: $T^a[q] = \{q' \in Q \mid (q, a, q') \in T\}$.
We also use the notation $p \xrightarrow{a}_T q$ for $(p, a, q) \in T$. We consider the set of the actions which can be performed in a state q: $\mathcal{A}\mathrm{ct}(q) = \{a \in A \mid \exists q' \in Q \,.\, q \xrightarrow{a} q'\}$.

Definition 2.2 *Let $S = (Q, A, T, q_0)$ be a LTS and q a state of Q.*
The set of the finite execution sequences from q (noted $Ex(q)$) is defined as follows:

$$Ex(q) = \{\sigma \in Q^* \,.\, \sigma(0) = q \ \wedge \ \forall i \,.\, 0 \leq i < |\sigma| - 1, \ \exists a_i \in A \,.\, \sigma(i) \xrightarrow{a_i}_T \sigma(i+1)\}.$$

In the following, for a LTS S, the term *execution sequences* of S represents the set $Ex(q_0)$ (where q_0 is the initial state of S). Furthermore, an execution sequence is *elementary* if and only if all its states are distinct. The subset of $Ex(q)$ containing the elementary execution sequences of a state q will be noted $Ex_e(q)$.

2.2 Equivalences and Preorders

We recall the definition of the simulation and the bisimulation relations.

Notation 1 *Let $\lambda \subseteq A^*$, and let $p, q \in Q$. We write $p \xrightarrow{\lambda}_T q$ if and only if:*
$\exists u_1 \cdots u_n \in \lambda \ \wedge \ \exists q_1, \cdots, q_{n-1} \in Q \ \wedge \ p \xrightarrow{u_1}_T q_1 \xrightarrow{u_2}_T q_2 \cdots q_i \xrightarrow{u_{i+1}}_T q_{i+1} \cdots q_{n-1} \xrightarrow{u_n}_T q.$

$T^{\lambda}[q] = \{q' \in Q \mid q \xrightarrow{\lambda}_T q'\}$. *Let* Π *be a family of disjoint languages on* A.
$\mathcal{A}ct_{\Pi}(q) = \{\lambda \in \Pi \mid \exists q' \,.\, q \xrightarrow{\lambda} q'\}$.

Definition 2.3 *(simulation) Let* Π *be a family of disjoint languages on* A. *We define inductively a family of simulations* R_k^{Π} *by:*

$$\begin{aligned} R_0^{\Pi} &= Q \times Q \\ R_{k+1}^{\Pi} &= \{(p_1,p_2) \mid \forall \lambda \in \Pi \,.\, \forall q_1 \,.\, (p_1 \xrightarrow{\lambda}_T q_1 \;\Rightarrow\; \exists q_2 \,.\, (p_2 \xrightarrow{\lambda}_T q_2 \;\wedge\; (q_1,q_2) \in R_k^{\Pi}))\} \end{aligned}$$

The simulation preorder for Π is $\sqsubseteq^{\Pi} = \bigcap_{k=0}^{\infty} R_k^{\Pi}$, the simulation equivalence is $\approx^{\Pi} = \bigcap_{k=0}^{\infty} (R_k^{\Pi} \cap R_k^{\Pi^{-1}})$.

Definition 2.4 *(bisimulation) Let* Π *be a family of disjoint languages on* A. *We define inductively a family of bisimulations* R_k^{Π} *by:*

$$\begin{aligned} R_0^{\Pi} &= Q \times Q \\ R_{k+1}^{\Pi} &= \{(p_1,p_2) \mid \forall \lambda \in \Pi \,.\, \forall q_1 \,.\, (p_1 \xrightarrow{\lambda}_T q_1 \;\Rightarrow\; \exists q_2 \,.\, (p_2 \xrightarrow{\lambda}_T q_2 \;\wedge\; (q_1,q_2) \in R_k^{\Pi})) \\ &\qquad\qquad \forall q_2 \,.\, (p_2 \xrightarrow{\lambda}_T q_2 \;\Rightarrow\; \exists q_1 \,.\, (p_1 \xrightarrow{\lambda}_T q_1 \;\wedge\; (q_1,q_2) \in R_k^{\Pi}))\} \end{aligned}$$

The bisimulation equivalence for Π is $\sim^{\Pi} = \bigcap_{k=0}^{\infty} R_k^{\Pi}$.

Remark 1 From these general definitions, several simulation and bisimulation relations can be defined. The choice of a class Π corresponds to the choice of an *abstraction criterion* on the actions. The *strong simulation* and the *strong bisimulation* are defined by $\Pi = \{\{a\} \mid a \in A\}$, the *w-bisimulation* is the bisimulation equivalence defined by $\Pi = \{\tau^* a \mid a \in A \;\wedge\; a \neq \tau\}$, the *safety preorder* is the simulation preorder defined by $\Pi = \{\tau^* a \mid a \in A \wedge a \neq \tau\}$ and the *safety equivalence* is the simulation equivalence where $\Pi = \{\tau^* a \mid a \in A \;\wedge\; a \neq \tau\}$.

Each equivalence relation R^{Π} defined on states can be extended to an equivalence relation comparing LTS in the following manner: let $S_i = (Q_i, A_\tau, T_i, q_i)$, for $i = 1,2$ be two LTS such that $Q_1 \cap Q_2 = \emptyset$ (if it is not the case, this condition can be easily obtained by renaming). Then we define $S_1 \; R^{\Pi} \; S_2$ if and only if $(q_1,q_2) \in R^{\Pi}$ and $S_1 \not R^{\Pi} S_2$ if and only if $(q_1,q_2) \notin R^{\Pi}$.

3 Verification of Simulations and Bisimulations "On the Fly"

In this section, we describe the principle of a decision procedure which allows to check if two LTS S_1 and S_2 are similar or bisimilar without explicitly constructing the two graphs. We define the product $S_1 \times_{R^{\Pi}} S_2$ between two LTS S_1 and S_2, and then we show how the existence of R^{Π} between these two LTS can be expressed as a simple criterion which must hold on the execution sequences of this product. In the rest of the section, we consider two LTS $S_i = (Q_i, A_i, T_i, q_{0i})$, for $i = 1,2$. We use p_i, q_i, p_i', q_i' to range over Q_i. We use R^{Π} and R_k^{Π} to denote either simulations or bisimulations ($R^{\Pi} = \bigcap_{k=0}^{\infty} R_k^{\Pi}$).

The LTS $S_1 \times_{R^{\Pi}} S_2$ is defined as a synchronous product of S_1 and S_2: a state (q_1,q_2) of $S_1 \times_{R^{\Pi}} S_2$ can perform a transition labeled by an action a if and only if the state q_1 (belonging to S_1) **and** the state q_2 (belonging to S_2) can perform a transition labeled by a. Otherwise,

- in the case of a simulation, if **only** the state q_1 can perform a transition labeled by a, then the product has a transition from (q_1, q_2) to the sink state noted *fail*.
- in the case of a bisimulation, if **only one** of the two states (q_1 or q_2) can perform a transition labeled by a, then the product has a transition from (q_1, q_2) to the sink state *fail*.

Definition 3.1 *We define the LTS* $S = S_1 \times_{R^\Pi} S_2$ *by:*
$S = (Q, A, T, (q_{01}, q_{02}))$, *with* $Q \subseteq (Q_1 \times Q_2) \cup \{fail\}$, $A = (A_1 \cap A_2) \cup \{\phi\}$, *and* $T \subseteq Q \times A \times Q$, *where* $\phi \notin (A_1 \cup A_2)$ *and* $fail \notin (Q_1 \cup Q_2)$.
T *and* Q *are defined as the smallest sets obtained by the applications of the following rules:* $R0$, $R1$ *and* $R2$ *in the case of a simulation,* $R0$, $R1$, $R2$ *and* $R3$ *in the case of a bisimulation.*

$$(q_{01}, q_{02}) \in Q \qquad [R0]$$

$$\frac{(q_1, q_2) \in Q,\ \mathcal{A}\mathrm{ct}_\Pi(q_1) = \mathcal{A}\mathrm{ct}_\Pi(q_2),\ q_1 \xrightarrow{\lambda}_{T_1} q_1',\ q_2 \xrightarrow{\lambda}_{T_2} q_2'}{\{(q_1', q_2')\} \in Q, \{(q_1, q_2) \xrightarrow{\lambda}_T (q_1', q_2')\} \in T} \qquad [R1]$$

$$\frac{(q_1, q_2) \in Q,\ q_1 \xrightarrow{\lambda}_{T_1} q_1',\ T^2_\lambda[q] = \emptyset}{\{fail\} \in Q, \{(q_1, q_2) \xrightarrow{\phi}_T fail\} \in T} \qquad [R2]$$

$$\frac{(q_1, q_2) \in Q,\ q_2 \xrightarrow{\lambda}_{T_2} q_2',\ T^1_\lambda[q] = \emptyset}{\{fail\} \in Q, \{(q_1, q_2) \xrightarrow{\phi}_T fail\} \in T} \qquad [R3 \text{ bisimulation}]$$

Let's notice that $(p_1, p_2) \xrightarrow{\phi}_T fail$ if and only if $(p_1, p_2) \notin R^\Pi_1$.

The following proposition allows to express that S_1 and S_2 are not comparable against R^Π in terms of the execution sequences of $S_1 \times_{R^\Pi} S_2$.

Proposition 3.1 *Let* $S = (Q, A, T, q_0)$ *be the product* $S_1 \times_{R^\Pi} S_2$. *Then,* $(q_{01}, q_{02}) \notin R^\Pi$ *if and only if it exists an elementary execution sequence* σ *of* S *(*$\sigma \in Ex_e(q_{01}, q_{02})$*) such that:*

- $\sigma = \{(q_{01}, q_{02}) = (p_0, q_0), (p_1, q_1),\ ...\ (p_k, q_k), fail\}$.
- $\forall i\ .\ 0 \leq i \leq k,\ (p_i, q_i) \notin R^\Pi_{k-i+1}$.

The proof of this proposition is based on the following lemma:

Lemma 3.1 *Let* $S = (Q, A, T, q_0)$ *be a LTS. Then we have,*
$\forall k \geq 1, \forall p, q \in Q,\ \ (p, q) \notin R^\Pi_{k+1}\ \wedge\ (p, q) \in R^\Pi_k \Rightarrow$
$\exists \lambda \in \Pi\ .\ \exists p'\ .\ \exists q'\ .\ p \xrightarrow{\lambda}_T p'\ \wedge\ q \xrightarrow{\lambda}_T q'\ \wedge\ (p', q') \notin R^\Pi_k\ \wedge\ (p', q') \in R^\Pi_{k-1}$.

Proof (By induction on k). We consider the case where the family $(R^\Pi_k)_{k \geq 0}$ are simulations. Let $(p, q) \notin R^\Pi_{k+1}$ and $(p, q) \in R^\Pi_k$. From $(p, q) \notin R^\Pi_{k+1}$, we have

$$\exists \lambda \in \Pi\ .\ \exists p'\ .\ p \xrightarrow{\lambda}_T p'\ \wedge\ (\forall q"\ .\ q \xrightarrow{\lambda}_T q"\ \Rightarrow\ (p', q") \notin R^\Pi_k).$$

Since $(p, q) \in R^\Pi_k$, $\exists q'\ .\ q \xrightarrow{\lambda}_T q'\ \wedge\ (p', q') \in R^\Pi_{k-1}$. The lemma is the conjunction of these two facts. □

In order to obtain the proof of proposition 3.1, we choose the least $k \geq 0$ such that $(p, q) \notin R^\Pi_{k+1}$ and $(p, q) \in R^\Pi_k$, then we construct the sequence $(p_0, q_0), ..., (p_k, q_k)$ such that

1. $(p_0, q_0) = (p, q)$,
2. $(p_i, q_i) \xrightarrow{\lambda_i}_T (p_{i+1}, q_{i+1})$, with $(p_i, q_i) \notin R^{\Pi}_{k-i+1}$ and $(p_i, q_i) \in R^{\Pi}_{k-i}$ for $0 \leq i < k$ (see lemma 3.1).
3. $(p_k, q_k) \notin R^{\Pi}_1$, which is equivalent to $(p_k, q_k) \xrightarrow{\phi}_T fail$.

If one of the two LTS is deterministic, proposition 3.1 can be improved. In this case, if S_2 (or S_1, if the $(R^{\Pi}_k)_{k\geq 0}$ are bisimulations) the converse of the lemma 3.1 holds: For a state (q_1, q_2) of $S_1 \times_{R^{\Pi}} S_2$, $(q_1, q_2) \in R^{\Pi}_k$ if and only if *fail* is not a successor of (q_1, q_2) and all the successors (q'_1, q'_2) of (q_1, q_2) verify $(q_1, q_2) \in R^{\Pi}_{k-1}$.

Lemma 3.2 *Let $S = (Q, A, T, q_0)$ be the product $S_1 \times_{R^{\Pi}} S_2$ and let us suppose that S_2 is deterministic (or S_1 if the $(R^{\Pi}_k)_{k\geq 0}$ are bisimulations).*
$\forall k \geq 1, \forall p, q \in Q,\ \exists \lambda \in \Pi\ .\ \exists p'\ .\ \exists q'\ .\ p \xrightarrow{\lambda}_T p' \ \wedge\ q \xrightarrow{\lambda}_T q' \ \wedge\ (p', q') \notin R^{\Pi}_k \ \Rightarrow\ (p, q) \notin R^{\Pi}_{k+1}$

From this lemma, we can deduce proposition 3.2:

Proposition 3.2 *Let $S = (Q, A, T, q_0)$ be the product $S_1 \times_{R^{\Pi}} S_2$ and let us suppose that S_2 is deterministic (or S_1 if the $(R^{\Pi}_k)_{k\geq 0}$ are bisimulations). Then:*
$S_1 \not R^{\Pi} S_2 \Leftrightarrow \exists \sigma \in Ex(q_{01}, q_{02})\ .\ \exists k > 0\ .\ \sigma(k) = \text{fail}$.

Proof We only show the $\Leftarrow$ direction. The $\Rightarrow$ direction is a consequence of the proposition 3.1.
Let k such that $\sigma(k) = fail$. Then we have : $\{(q_{01}, q_{02}),\ \ldots\ (p_k, q_k), fail\} \in Ex(q_{01}, q_{02}$. Thus, $(p_k, q_k) \notin R^{\Pi}_1$. According to the lemma 3.2, $(p_{k-1}, q_{k-1}) \notin R^{\Pi}_2$, and so on. Thus, $(q_{01}, q_{02}) \notin R^{\Pi}_{k+1}$. □

According to this proposition, if at least one of the two LTS S_1 or S_2 (resp. S_2) is deterministic then S_1 and S_2 are not bisimilar (resp. similar) if and only if it exists an execution sequence of $S_1 \times_{R^{\Pi}} S_2$ containing the state *fail*.

4 Algorithms

In the previous section, we have expressed the bisimulation and the simulation between two LTS S_1 and S_2 in terms of the existence of a particular execution sequence of their product $S_1 \times_{R^{\Pi}} S_2$. Now we show that this verification can be realized by performing depth-first searches (DFS for short) on the LTS $S_1 \times_{R^{\Pi}} S_2$. Consequently, the algorithm does not require to construct the two LTSpreviously : the states of $S_1 \times_{R^{\Pi}} S_2$ are generated during the DFS ("on the fly" verification), but not necessarily all stored. And the most important is that transitions do not have to be stored.

We note n_1 (resp. n_2) the number of states of S_1 (resp. S_2), and n the number of states of $S_1 \times_{R^{\Pi}} S_2$ ($n \leq n_1 \times n_2$). We describe the algorithm considering the two following cases:

Deterministic case: if R^{Π} represents a simulation (resp. a bisimulation) and if S_2 (resp. either S_1 or S_2) is deterministic, then, according to proposition 3.2, it is sufficient to check whether or not the state *fail* belongs to $S_1 \times_{R^{\Pi}} S_2$, which can be easily done by performing a usual DFS of $S_1 \times_{R^{\Pi}} S_2$. The verification is then reduced to a simple reachability problem in this graph. Consequently, if we store all the visited states during the DFS, the time and memory complexities of this decision procedure are $O(n)$. Several memory efficient solutions exist to manage such a DFS ([Hol89]).

General case: in the general case, according to the proposition 3.1, we have to check the existence of an execution sequence σ of $S_1 \times_{R^\Pi} S_2$ which contains the state *fail* and which is such that for all states (q_1, q_2) of σ, $(q_1, q_2) \notin R_k^\Pi$ for a certain k. According to the definition of R_k^Π, this verification can be done during a DFS as well if:

- the relation R_1^Π can be checked.
- for each visited state (q_1, q_2), the result $(q_1, q_2) \in R_k^\Pi$ is synthesized for its predecessors in the current sequence (the states are then analyzed during the back tracking phase).

More precisely, the principle of the general case algorithm is the following: if R^Π is a simulation (resp. a bisimulation) we associate with each state (q_1, q_2) a bit_array M of size $|T_1[q_1]|$ (resp. $|T_1[q_1]|+|T_2[q_2]|$). During the analysis of each successor (q_1', q_2') of (q_1, q_2), whenever it happens that $(q_1', q_2') \in R^\Pi$ then $M[q_1']$ (resp. $M[q_1']$ and $M[q_2']$) is set to 1. Thus, when all the successors of (q_1, q_2) have been analyzed, $(q_1, q_2) \in R^\Pi$ if and only if all the elements of M have been set to 1.

As in the deterministic case algorithm, to reduce the exponential time complexity of the DFS the usual method would consist in storing all the visited states (including those which do not belong to the current sequence) together with the result of their analysis (i.e, if they belong or not to R^Π). Unfortunately, this solution cannot be straightly applied:

During the DFS, the states are analyzed in a postfixed order. Consequently, it is possible to reach a state which has already been visited, but not yet analyzed (since the visits are performed in a prefixed order). Therefore, the result of the analysis of such a state is unknown (it is not available yet). We propose the following solution for this problem:

Notation 2 We call the *status* of a state the result of the analysis of this state by the algorithm. The status of (q_1, q_2) is "$\sim$" if $(q_1, q_2) \in R^\Pi$, and is "$\not\sim$" otherwise.

Whenever a state already visited but not yet analyzed (i.e, which belongs to the stack) is reached, then we assume its status to be "$\sim$". If, when the analysis of this state completes (i.e, when it is popped), the obtained status is "$\not\sim$", then a TRUE answer from the algorithm is not reliable (a wrong assumption was used), and another DFS has to be performed. On the other hand, a FALSE answer is always reliable.

Consequently, the following data structures are required:

- A stack St_1, to store the states already visited of the current execution sequence. Each element of St_1 is a couple $((p, q), l)$, where (p, q) is a state and l the list of its direct successors which remains to explore.
- A stack St_2, to store the bit_arrays associated to each state of the current execution sequence. We assume that whenever a new array is pushed into St_2, then it is initialized with the value 0.
- a set V, to mark all the visited states.
- a set R, to store all the states of the current sequence visited more than once.
- a set W, to store all the states for which the obtained status is "$\not\sim$".

The list of all direct successors of a state (p, q) is obtained by the function *succ*:

$$succ(p, q) = \{(a, (p', q')) \,.\, p \xrightarrow{a}_{T_1} p' \ \wedge \ q \xrightarrow{a}_{T_2} q'\}.$$

$succ(p, q)$ can be incrementally computed in the following manner:

- calculate the direct successors of p and q applying the transition rules of the description language of S_1 and S_2.
- calculate the direct successors of (p, q), applying the rules given in definition 3.1.

We also consider the function $partial\_DFS$, which performs a DFS storing all the visited states and analyzing only the states which do not belong to $V \cup W$. The result returned by this function may be TRUE, FALSE or UNRELIABLE. The algorithm then consists in a sequence of calls of $partial\_DFS$ (each call increasing the set W), until the result belongs to $\{TRUE, FALSE\}$.

The algorithm dealing with the bisimulation relation is the following:

Algorithm

$W := \emptyset$
repeat
  $result := partial\_DFS$ { perform a DFS }
until $result \in$ {TRUE, FALSE}
return $result$
end.

function partial_DFS

$V := \emptyset$; $R := \emptyset$; $stable :=$ false
$St_1 := \{(q_{01}, q_{02}), succ(q_{01}, q_{02})\}$
$St_2 := \emptyset$
push into St_2 a bit_array of size 2 { in order to deal with (q_{01}, q_{02}) }
push into St_2 a bit_array of size $(|T_1[q_{01}]| + |T_2[q_{02}]|)$ **(1)**
while $St_1 \neq \emptyset$
  $stable :=$ true
  $((q_1, q_2), l) := \text{top}(St_1)$
  $M := \text{top}(St_2)$
  if $l \neq \emptyset$
    choose and remove (q'_1, q'_2) in l
    if $(q'_1, q'_2) \notin V \cup W$
      if $(q'_1, q'_2) \notin St_1$ { it's a new state }
        if $\neg\ ((q'_1, q'_2) \xrightarrow{\phi}_T fail)$
          push $\{(q'_1, q'_2), succ(q'_1, q'_2)\}$ in St_1
          push into St_2 a bit_array of size $(|T_1[q'_1]| + |T_2[q'_2]|)$ **(1)**
        endif
      else { $(q'_1, q'_2 \in St_1)$ }
        insert (q'_1, q'_2) in R { this state has been visited more that once }
        $M[q'_1] := 1$; $M[q'_2] := 1$ **(2)**
      endif
    else { $(q'_1, q'_2) \in V \cup W$ (i.e, visited in a previous DFS) }
      if $(q'_1, q'_2) \notin W$
        $M[q'_1] := 1$; $M[q'_2] := 1$ **(2)** { $q'_1 \sim q'_2$ }
      endif
    endif
  else { $l \neq \emptyset$ }

```
                pop(St_1) ; pop(St_2)
                insert (q_1, q_2) in V  { a new state has been analyzed }
                M' := top(St_2)
                if M[q'] = 1 for all q' in (T_1[q_1] ∪ T_2[q_2]) (3)
                        M'[q_1] := 1 ; M'[q_2] := 1 {q_1 ~ q_2 } (2)
                else
                        insert (q_1, q_2) in W  { q_1 ≁ q_2 }
                        if (q_1, q_2) ∈ R
                                stable := false  { we assumed a wrong status }
                        endif
                endif
        endif
    endwhile
    M := top(St_2)
    if M[q_01] ≠ 1 and M[q_02] ≠ 1 (4)
        return FALSE  { q_01 ≁ q_02 }
    else
        if stable
            return TRUE  { q_01 ~ q_02 }
        else
            return UNRELIABLE  { another DFS has to be performed }
        endif
    endif
end.
```

The algorithm dealing with the simulation is straightly obtained by replacing:

(1) by push into St_2 a bit_array of size $(|T_1[q_{01}]|)$

(2) by $M[q'_1] := 1$

(3) by if $M[q'] = 1$ for all q' in $T_1[q_1]$

(4) by if $M[q_{01}] = 1$

Proposition 4.1 *Algorithm terminates, and it returns TRUE if and only if the two LTS are bisimilars.*

Proof We use the following notations: let DFS_i representing the i^{th} execution of the function *partial_DFS*, and let R_i (resp. W_i) representing the set R (resp. W) at the end of DFS_i. When DFS_i terminates, the following property holds:

$stable = False \Leftrightarrow R_k \cap W_k \neq \emptyset \quad (1)$

Algorithm terminates: From (1), $\forall i$. DFS_i returns UNRELIABLE $\Leftrightarrow$
$\exists (q_1, q_2) \in Q$. $((q_1, q_2) \in W_i \cap R_i)$.

Moreover, as during DFS_i the states of W_{i-1} aren't pushed, we also have:
$\forall i$. $\forall (q_1, q_2) \in Q$. $((q_1, q_2) \in R_i \Rightarrow (q_1, q_2) \notin W_{i-1})$.

From these two assertions, we can deduce :
$\forall i$. DFS_i returns UNRELIABLE $\Rightarrow$
$\exists (q_1, q_2) \in Q$. $((q_1, q_2) \in W_i \ \wedge \ (q_1, q_2) \notin W_{i-1})$.

Consequently, the set W increases strictly ($\forall i \ . \ W_i \subset W_{i+1}$) and, as Q is finite, it exists a k such that DFS_k doesn't return UNRELIABLE, which ensures the termination of Algorithm. Moreover, the number of calls to the function *partial_DFS* is less or equal to n.

It remains to prove the correctness. Let DFS_k be the last DFS performed. From (1), $R_k \cap W_k = \emptyset \ \vee \ DFS_k$ returns FALSE. But,

- if $R_k \cap W_k = \emptyset$, then all the assumptions made during DFS_k are correct. Consequently, the obtained result is correct too.
- Whenever the status of a state is unknown, it's assumed to be $\sim$. Thus, the relation computed by the algorithm contains the relation $\sim$ (it's a weaker relation). It follows that if the algorithm returns FALSE then the LTS aren't bisimilar.

□.

The time requirement for the function *partial_DFS* is $O(n)$. In the worst case, as pointed out in the proof of proposition 4.1 the number of calls of this function may be n. Consequently, the theoretical time requirement for this algorithm is $O(n^2)$. In practice, it turns out that only 1 or 2 DFS are required to obtain a reliable result. Moreover, whenever the LTS are not bisimilar, the time requirement is always $O(n)$.

In both cases, the memory requirement for the algorithm is $O(n)$. However, the data structures required can be divided into *sequentially accessed* memory (St_1 and St_2) and *randomly accessed* memory (R, V and W). Furthermore, as it is not critical to store **all** the already visited states, memory efficient implementations can be found for the set V, like hash-based caches.

5 Applications and Results

From this general algorithm several decision procedures for bisimulation and simulation based relations have been implemented in the tool ALDÉBARAN, like strong and w-bisimulation, strong simulation, safety preorder and safety equivalence. However, as it is the case for the Paige & Tarjan algorithm, such decision procedures are really useful in a verification tool – from a user's point of view – only if they allow to build a *diagnosis* whenever the two LTS are not related. We show how the previous algorithm has been modified in order to allow this computation. Then, we give some results obtained when applying it to the verification of LOTOS specifications.

Remark 2 In this draft implementation, the verification is not performed "on the fly" straightly from the LOTOS specifications: the LTS are previously generated and the verification phase consists in simultaneously building the LTS product and deciding whether or not they are related, as described in the algorithm. Thus, the obtained results can be compared with the classical verification procedure (based on the Paige & Tarjan algorithm) already implemented in ALDÉBARAN.

5.1 Diagnosis

Several formalisms have been proposed in order to express the "non bisimulation" of two LTS (for example Hennessy-Milner Logic in [Cle90]). We present here a more intuitive solution, suitable either for bisimulation or simulation relations (both denoted by R^{Π}): whenever the two LTS S_1 and S_2 are not related, we build an *explanation sequence* consisting of an execution sequence σ of $S_1 \times_{R^{\Pi}} S_2$

terminated by a failure state (p_k, q_k) which is not in R_1^Π (i.e, from which it clearly appears that S_1 and S_2 are not related) and such that for each (p_i, q_i) of σ, $(p_i, q_i) \notin R^\Pi$.

Definition 5.1 *Let S_1 and S_2 be two LTS. An explanation sequence of $S_1 \not R^\Pi S_2$ is an execution sequence σ of $S_1 \times_{R^\Pi} S_2$ such that:*

- $\sigma = \{(q_{01}, q_{02}) = (p_1, q_1), (p_2, q_2), ..., (p_k, q_k)\}$
- $\forall i \; . \; 0 \leq i \leq k, \; (p_i, q_i) \notin R^\Pi_{k-i+1}$.
- $(p_k, q_k) \notin R_1^\Pi$

In fact, the explanation sequences are exactly the execution sequences which are looked for during the verification phase, see proposition 3.1.

We show how such a sequence can be obtained (and therefore printed) without modifying the time and memory complexities of the previous algorithm:

deterministic case: Obviously, when a state *fail* is reached during the DFS of $S_1 \times_{R^\Pi} S_2$ the stack St_1 contains an *explanation sequence* (proposition 3.2).

general case: In this case, the sequence has to be explicitly built during the verification phase. In the previous algorithm, all the visited states (p, q) of $S_1 \times_{R^\Pi} S_2$ which do not belong to R^Π are inserted in the set W. To obtain an *explanation sequence*, it is then sufficient to modify the algorithm in the following manner: whenever a new state is inserted in W, it is linked with one of its successor already in W (which always exists). Thus, if the initial state of the product belongs to W (i.e, the two LTS are not related), an *explanation sequence* is straightly available from its associated linked list.

5.2 Results

Two examples are studied here: the first one is an alternating bit protocol called Datalink protocol [QPF88], and the second one is a more realistic example, the rel/REL_{fifo} protocol [SE90]. For each example, the verification was performed as follows:

- generating the LTS S_1 (*Implementation*) from the LOTOS description, using the LOTOS compiler CÆSAR [GS90].
- building the LTS S_2 (*Specification*), representing the expected behavior of the system.
- comparing S_1 and S_2 with respect to w-bisimulation or safety equivalence, using both the usual decision procedure of ALDÉBARAN and the improved one described in this paper.

5.2.1 Datalink protocol

The Datalink protocol is an example of an alternating bit protocol. The LOTOS specification provided to CÆSAR is described in [QPF88]. By varying the number of the different messages (noted N), LTS of different sizes can be obtained. These LTS have been compared, with respect to w-bisimulation, with the LTS describing the expected behavior of the protocol. However, for $N > 40$, the memory required by the classical decision procedure of ALDÉBARAN becomes too large, and consequently the verification can no longer be performed with this procedure.

The following notations are used:

- n_i and m_i denote the number of states and transitions of the two LTS ($i = 1, 2$).
- n denotes the number of states of the product which have been effectively analyzed.
- $t1$ is the time needed by the usual decision procedure of ALDÉBARAN.
- $t2$ is the time needed by the decision procedure described in this paper.

The times given here are elapsed times, obtained on a SUN 3-80 Workstation.

N	n1	m1	n2	m2	n	t1	t2
20	7241	10560	41	440	1661	0:24	0:19
30	15661	23040	60	930	3691	0:57	0:55
40	27281	40320	80	1640	6521	2:07	1:45
50	42101	62400	101	2600	10151	—	2:27
60	60121	89280	121	3720	14581	—	3:42
70	81341	120960	140	4970	19811	—	6:42
80	105761	157440	161	6560	25841	—	9:23

5.2.2 rel/REL_{fifo} protocol

This algorithm has also been used for the verification of a "real" protocol, rel/REL_{fifo} ([SE90]), carried out in Hewlett-Packard Laboratories [MB90]. This *reliable multicast protocol* provides the following service:

Atomicity: If a multicast from a transmitter is received by a functioning receiver, then all the other functioning receivers will also receive it, even if the transmitter crashes during the multicast.

Fifo: All the multicasts from the same transmitter are received by the functioning receivers in the order of the multicasts were made.

This protocol has been modeled in LOTOS, and a LTS of 680 000 states and 1 900 000 transitions has been generated by CÆSAR. The **Fifo** requirement has been verified by comparing (with respect to safety equivalence) this LTS in which only the actions performed by one receiver were visible, with the expected behavior of a single receiver. Although the size of the graphs prevented a verification by using the Paige & Tarjan algorithm, this comparison was carried out by using the algorithm described in this paper in less than 3 hours on a HP-9000 Workstation.

6 Conclusion

Several applications can be obtained from the algorithm described in this paper.

First, it can be viewed as a new decision procedure (in the usual sense) for bisimulation equivalence, simulation equivalence and simulation preoders between LTS.

The results obtained, from a draft implementation in ALDÉBARAN, show that this algorithm can be more efficient than the usual one. As this algorithm requires less memory, verifications of larger LTS become possible.

Moreover, the diagnosis capability of this decision procedure is very useful from the user's point of view for the specification of communicating processes (as a debugging tool for a sequential language).

But one of the major improvement provided by this algorithm is that "on the fly" verification of bisimulation and simulation relations are allowed. In this framework, our project is to modify the LOTOS compiler CÆSAR to compare LOTOS specifications (with respect to these relations) without explicitly storing the whole LTS of the LOTOS specifications. Consequently, checking of real size examples could be carried out.

References

[BFG*91] A. Bouajjani, J.C. Fernandez, S. Graf, C. Rodriguez, and J. Sifakis. Safety for Branching Time Semantics. In *18th ICALP*, july 1991.

[BFH90] A. Bouajjani, J. C. Fernandez, and N. Halbwachs. *On the verification of safety properties.* Tech. report, Spectre L 12, IMAG, Grenoble, march 1990.

[Cle90] R. Cleaveland. On Automatically Distinguishing Inequivalent Processes. In *Workshop on Computer-Aided Verification*, june 1990.

[CVWY90] C. Courcoubetis, M. Vardi, P. Wolper, and M. Yannakakis. Memory Efficient Algorithms for the Verification of Temporal Properties. In *Workshop on Computer-Aided Verification*, june 1990.

[Fer89] J. C. Fernandez. *Aldébaran: A tool for verification of communicating processes.* Tech. report Spectre C14, LGI-IMAG Grenoble, 1989.

[FM90] J. C. Fernandez and L. Mounier. Verifying Bisimulations on the Fly. In *Proceedings of the Third International Conference on Formal Description Techniques FORTE'90 (Madrid, Spain)*, pages 91–105, North-Holland, November 1990.

[GS86] S. Graf and J. Sifakis. *Readiness Semantics for Regular Processes with Silent Action.* Technical Report Projet Cesar RT-3, LGI-IMAG Grenoble, 1986.

[GS90] Hubert Garavel and Joseph Sifakis. Compilation and Verification of LOTOS Specifications. In L. Logrippo, R. L. Probert, and H. Ural, editors, *Proceedings of the 10th International Symposium on Protocol Specification, Testing and Verification (Ottawa)*, IFIP, North-Holland, Amsterdam, June 1990.

[Hol89] Gerard J. Holzmann. Algorithms for Automated Protocol Validation. In *Proceedings of the 1st International Workshop on Automatic Verification Methods for Finite State Systems (Grenoble, France)*, Springer Verlag, jun 1989.

[JJ89] Claude Jard and Thierry Jeron. On-Line Model-Checking for Finite Linear Temporal Logic Specifications. In *International Workshop on Automatic Verification Methods for Finite State Systems, LNCS 407*, Springer Verlag, 1989.

[MB90] Laurent Mounier and Simon Bainbridge. *Specification and Verification of a Reliable Multicast Protocol.* Technical Report (In preparation), Hewlett-Packard Laboratories, Bristol, U.K, 1990.

[Mil80] R. Milner. A Calculus of Communication Systems. In *LNCS 92*, Springer Verlag, 1980.

[PT87] R. Paige and R. Tarjan. Three Partition Refinement Algorithms. *SIAM J. Comput., No. 6*, 16, 1987.

[QPF88] Juan Quemada, Santiago Pavón, and Angel Fernández. Transforming LOTOS Specifications with LOLA: The Parametrized Expansion. In Kenneth J. Turner, editor, *Proceedings of the 1st International Conference on Formal Description Techniques FORTE'88 (Stirling, Scotland)*, pages 45–54, North-Holland, Amsterdam, September 1988.

[Rod88] C. Rodriguez. *Spécification et validation de systèmes en XESAR.* PhD thesis, Institut National Polytechnique de Grenoble, 1988.

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