Verifying Bisimulations "On the Fly"

Jean-Claude Fernandez Laurent Mounier

LGI-IMAG BP 53X 38041 GRENOBLE Cedex

Abstract

This paper describes a decision procedure for bisimulation-based equivalence relations between labeled transition systems. The algorithm usually performed in order to verify bisimulation consists in refining some initial equivalence relation until it becomes compatible with the transition relation under consideration. However, this method requires to store the transition relation explicitly, which limits it to medium-sized labeled transition systems.
The algorithm proposed here does not need to previously construct the two transition systems: the verification can be performed during their generation. Thus, the amount of memory required can be significantly reduced, and verification of larger size systems becomes possible. This algorithm has been implemented in the tool ALDÉBARAN *and has been used in the framework of verification of* LOTOS *specifications.*

1 Introduction

One of the successful approaches used for the verification of systems of communicating processes is provided by behavioural equivalence relations, which allow to compare different descriptions of a given system.
Such an approach appears to be particularly convenient for the verification of LOTOS specifications: the operational semantic of LOTOS (derived from CCS [Mil80] and CSP [BHR84]) defines a program as a labeled transition system, and thus descriptions of different abstraction levels of a given communicating system, expressed in the same formalism, can be compared by these equivalences, on labeled transition systems.
More precisely, if we note S (Specification) the more abstract description of the system and I (Implementation) the more 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 S_1 and S_2. Let $\sim_R$ be an appropriate equivalence relation (on labeled transition systems). Then, I implements S if and only if $S_1 \sim_R S_2$.
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 labeled transition system 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 labeled transition systems.
However, the main drawback of this method is that the whole labeled transition systems 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 present an alternative decision procedure for bisimulation based equivalence relations which allows to compare labeled transition systems without explicitly representing them. Thus, the verification can be done during the process of the two transition systems (verification "on the fly"). This approach is similar to the one proposed in [JJ89] and [CVWY90], which deals with "on the fly" verification of linear temporal logic properties.
A version of our algorithm for the bisimulation-based safety equivalence [Rod88] has been implemented in the tool CÆSAR-ALDÉBARAN. This tool is composed of a LOTOS compiler (CÆSAR [GS90] [Gar89]), able to generate efficiently a labeled transition system from a given specification, and an equivalence checker (ALDÉBARAN [Fer88] [Fer89]) which allows to compare and reduce labeled transition systems 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 strong bisimulation is described, and in section 4 we show how the algorithm can be adapted to safety equivalence. The results obtained when applying the usual one and our improved algorithm 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.

Definition 2.1 *A labeled transition system is a quadruplet $S = (Q, A, T, q_0)$ 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.*
- *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 $(p, a, q) \in T$.

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

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

In the following, for a labeled transition system 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 is denoted $Ex_e(q)$.

2.2 strong bisimulation

Intuitively, two states p and q are strongly bisimilar if for each state p' reachable from p by execution of an action a there is a state q', reachable from q by execution of the same action a such that p' and q' are also strongly bisimilar.
Strong bisimulation, noted $\sim$, can be defined as the intersection of a sequence of decreasing relations [Mil80]:

Definition 2.3 $\sim = \cap \sim^i$ *for* $i \in N$, *where* $\sim^i$ *are defined by:*

- $\forall p, q \in Q \,.\, p \sim^0 q$,
- $p_1 \sim^{i+1} p_2$ *if and only if* $\forall a \in A$.
  $\forall r_1 \,.\, (p_1 \xrightarrow{a}_T r_1 \Rightarrow \exists r_2 \,.\, (p_2 \xrightarrow{a}_T r_2 \wedge r_1 \sim^i r_2)) \wedge$
  $\forall r_2 \,.\, (p_2 \xrightarrow{a}_T r_2 \Rightarrow \exists r_1 \,.\, (p_1 \xrightarrow{a}_T r_1 \wedge r_1 \sim^i r_2))$.

Each equivalence relation $\sim_R$ defined on states can be extended to an equivalence relation comparing labeled transition systems in the following manner: let $S_i = (Q_i, A_\tau, T_i, q_i)$, for $i = 1, 2$ be two labeled transition systems 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 $S_1 \sim_R S_2$ if and only if $q_1 \sim_R q_2$

3 Verification of Strong Bisimulation "On the Fly"

In this section, we describe a decision procedure which allows to check if two labeled transition systems S_1 and S_2 are strongly bisimilar without explicitly constructing the two graphs. This procedure is based on the exploration of all the execution sequences of the product of the two labeled transition systems.
First, we justify the principle of the verification, and then we propose different algorithms.

3.1 Principle of the verification

We give the definition of the product $S_1 \times S_2$ between two labeled transition systems S_1 and S_2, and then we show how the existence of a strong bisimulation between these two labeled transition systems can be expressed as a simple criterion which must hold on the execution sequences of this product.

For two labeled transition systems S_1 and S_2, the labeled transition system $S_1 \times S_2$ is defined as a synchronous product of S_1 and S_2: a state (q_1, q_2) of $S_1 \times S_2$ 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, 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 noted *fail*.

Definition 3.1 *Let $S_i = (Q_i, A_i, T_i, q_{0i})$ for i = 1,2 be two labeled transition systems. We define the labeled transition system $S = S_1 \times S_2$ 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 $fail \notin (A_1 \cup A_2)$ and $\phi \notin (Q_1 \cup Q_2)$.
T and Q are defined as the smallest sets obtained by the applications of the following rules:

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

$$\frac{(q_1, q_2) \in Q,\ q_1 \xrightarrow{a}_{T_1} q_1',\ q_2 \xrightarrow{a}_{T_2} q_2'}{\{(q_1', q_2')\} \in Q, \{(q_1, q_2) \xrightarrow{a}_T (q_1', q_2')\} \in T} \qquad [R1]$$

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

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

The following proposition allows to express the non-equivalence of two labeled transition systems S_1 and S_2, in terms of the execution sequences of $S_1 \times S_2$.

Proposition 3.1 *Let $S_i = (Q_i, A_i, T_i, q_{0i})$ for i = 1,2 be two labeled transition systems, and let $S = (Q, A, T, q_0)$ be the product $S_1 \times S_2$.*
Then, $q_{01} \not\sim q_{02}$ if and only if it exists an elementary execution sequence σ of S ($\sigma \in Ex_e((q_{01}, q_{02}))$ such that:

- $\sigma = \{(q_{01}, q_{02}) = (p_0, q_0), (p_1, q_1),\ \ldots\ (p_k, q_k), fail\}$.
- $\forall i\ .\ 0 \leq i \leq k,\ p_i \not\sim^{k-i+1} q_i$

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

Lemma 3.1 *Let $S = (Q, A, T, q_0)$ be a labeled transition system. Then we have,*
$\forall k \geq 1, \forall p, q \in Q$,

$$p \not\sim^{k+1} q \ \wedge\ p \sim^k q \Rightarrow \exists a \in A\ .\ \exists p'\ .\ \exists q'\ .\ p \xrightarrow{a}_T p' \ \wedge\ q \xrightarrow{a}_T q' \ \wedge\ p' \not\sim^k q' \ \wedge\ p' \sim^{k-1} q'.$$

Moreover, if one of the two labeled transition systems is deterministic, proposition 3.1 can be improved. In the following lemma, we give an expression of the relation $\sim^k$ which holds in this case:

For a state (q_1, q_2) of $S_1 \times S_2$, $q_1 \sim^k q_2$ 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 $q_1 \sim^{k-1} q_2$.

Lemma 3.2 *Let S_1 and S_2 be two labeled transition systems, let $S = (Q, A, T, q_0)$ be the product $S_1 \times S_2$, and let us suppose that S_1 or S_2 is deterministic i.e.,*
$\forall a \in A\ .\ \forall q \in Q_i\ .\ |T_{ia}[q]| \leq 1$ for i = 1,2.
Then, it follows: $\forall (p, q) \in Q, \forall k \geq 1, p \sim^k q \Leftrightarrow$

$$\neg((p, q) \xrightarrow{\phi}_T fail) \ \wedge\ \forall a \in A\ .\ \forall (p', q')\ .\ ((p, q) \xrightarrow{a}_T (p', q') \Rightarrow p' \sim^{k-1} q').$$

From this lemma, we can deduce proposition 3.2:

Proposition 3.2 *Let S_1 and S_2 be two labeled transition systems, let $S = (Q, A, T, q_0)$ be the product $S_1 \times S_2$, and let us suppose that S_1 or S_2 is deterministic:*
Then: $S_1 \not\sim S_2 \Leftrightarrow \exists \sigma \in Ex(q_{01}, q_{02}) \,.\, \exists k > 0 \,.\, \sigma(k) = fail.$

According to this proposition, if at least one of the two labeled transition systems (S_1 or S_2) is deterministic then S_1 and S_2 are not strongly bisimilar if and only if it exists an execution sequence of $S_1 \times S_2$ containing the state *fail*.

3.2 Algorithms

We have expressed the strong bisimulation of two labeled transition systems S_1 and S_2 by means of the product $S_1 \times S_2$ in the following manner:

- If one of the two labeled transition systems is deterministic, then $S_1 \sim S_2$ if and only if the state *fail* does not belong to $S_1 \times S_2$.
- In the general case, $S_1 \sim S_2$ if and only if one cannot find an execution sequence σ of $S_1 \times S_2$ which contains the state *fail* and which is such that all the states (q_1, q_2) of σ verify $q_1 \not\sim q_2$.

In this section, we show that these verifications can both be realized by performing depth_first searches (DFS) on $S_1 \times S_2$. Consequently, the algorithms do not need to previously construct the two labeled transition systems: the states of $S_1 \times S_2$ are generated during the DFS (verification "on the fly"), but not necessarily all stored. And, what is most important, transitions need never to be stored.
We give three algorithms realizing such a verification: the first one is devoted to the deterministic case and the two others deal with the general case. For each of them, we discuss the time and memory complexity.

3.2.1 Deterministic case

According to proposition 3.2, if one of the two labeled transition systems is deterministic, the problem of the verification of $S_1 \sim S_2$ can be reduced to a simple reachability problem in the graph $S_1 \times S_2$. We give here a usual DFS algorithm (Algorithm 1).
The required data structures are the following: a stack St_1 in order to store the execution sequence which is currently analyzed and a set W, which contains all the already visited states. 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 have not yet been encountered in the currently analyzed execution sequence.
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 executed in the following manner:

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

Algorithm1

```
St1 := {(q01, q02), succ(q01, q02)}
W := {(q01, q02)}
while St1 ≠ ∅
    ((q1, q2), l) := top(St1)
    if l = ∅  { if all successors have been reached ... }
        pop(St1) { then backtrack}
    else
        choose and remove (q'1, q'2) in l
        if ¬ ((q'1, q'2) --φ-->T fail)
            if (q'1, q'2) ∉ W  { this is a new state }
                insert (q'1, q'2) in W
                push {(q'1, q'2), succ(q'1, q'2)} in St1
            endif
        else
            return FALSE  { fail is reachable ⇒ S1 ≁ S2 }
        endif
    endif
endwhile
return TRUE  { fail isn't reachable ⇒ S1 ∼ S2 }
end.
```

Notice that St_1 is needed in order to perform a complete search, but the set W is only intended to improve the efficiency of the algorithm by storing partial results, and therefore do not need to contain all the states.
Let n_i be the number of states of S_i (i = 1,2), and let n be the number of states of $S_1 \times S_2$ ($n \leq n_1 \times n_2$).
The time and memory requirements of Algorithm 1 are of order of $O(n)$. However, several graph search algorithms have been proposed allowing to improve these complexities, possibly with partial search (see for example [Hol90]). All of them can easily be adapted to this particular problem.

3.2.2 General case

According to proposition 3.1, in order to decide if S_1 and S_2 are bisimilar, it is sufficient to check whether or not it exists an execution sequence of $S_1 \times S_2$ which contains the state *fail* and which is such that all of its states (q_1, q_2) satisfy $q_1 \not\sim q_2$. Consequently, the main problem is to be able for each states (q_1, q_2) in Q, to decide if $q_1 \not\sim^i q_2$ for some i.
The proposed solution is based on the definition of $\not\sim^i$:

$$
\begin{aligned}
& q_1 \not\sim^i q_2 \Leftrightarrow \exists a \in A \,. \\
& \quad [\exists q'_1 \,.\, (q_1 \xrightarrow{a}_{T_1} q'_1 \;\wedge\; (\forall q'_2 \,.\, (q_2 \xrightarrow{a}_{T_2} q'_2 \Rightarrow q'_1 \not\sim^{i-1} q'_2) \;\vee\; T_{2a}[q_2] = \emptyset)) \vee \\
& \quad \exists q'_2 \,.\, (q_2 \xrightarrow{a}_{T_2} q'_2 \;\wedge\; (\forall q'_1 \,.\, (q_1 \xrightarrow{a}_{T_1} q'_1 \Rightarrow q'_1 \not\sim^{i-1} q'_2) \;\vee\; T_{1a}[q_1] = \emptyset))]
\end{aligned}
$$

which is equivalent to:

$$
\begin{aligned}
q_1 \not\sim^i q_2 \Leftrightarrow\ & q_1 \not\sim^1 q_2 \ \vee\ \exists a \in A\ . \\
& [\exists q_1'\ .\ (q_1 \xrightarrow{a}_{T_1} q_1' \ \wedge\ \forall q_2'\ .\ (q_2 \xrightarrow{a}_{T_2} q_2' \Rightarrow q_1' \not\sim^{i-1} q_2')) \vee \\
& \exists q_2'\ .\ (q_2 \xrightarrow{a}_{T_2} q_2' \ \wedge\ \forall q_1'\ .\ (q_1 \xrightarrow{a}_{T_1} q_1' \Rightarrow q_1' \not\sim^{i-1} q_2'))]
\end{aligned}
$$

According to this definition, it appears that the verification can be done during a DFS of the graph $S_1 \times S_2$ if:

- the relation $\not\sim^1$ can be checked.
- for each analyzed state (q_1, q_2), the result $(q_1 \sim^i q_2)$ 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 algorithm is the following:
Associated with each state (q_1, q_2), we need a bit_array M of size $|T_1[q_1]| + |T_2[q_2]|$. During the analysis of each state (q_1', q_2') in $succ(q_1, q_2)$, whenever q_1' and q_2' are found bisimilar then $M[q_1']$ and $M[q_2']$ are set to 1. Thus, when all the successors of (q_1, q_2) have been analyzed, $q_1 \sim q_2$ if and only if all the elements of M have been set to 1.
Furthermore, as we can restrict our analysis to elementary sequences (proposition 3.1), each sequence in the DFS is terminated either by a sink state, or by a state belonging already to the same sequence.
The following data structures are required:
The DFS itself is managed by a stack St_1, as in Algorithm 1. Moreover, since a bit_array is needed for each state of the current sequence, we use another stack St_2, which is the stack of these arrays. We assume that whenever a new array is pushed into St_2, then it is initialized with the value 0.
The algorithm is the following:

Algorithm2

```
St1 := {(q01, q02), succ(q01, q02)}
St2 := ∅
push into St2 a bit_array of size 2  { in order to deal with (q01, q02) }
push into St2 a bit_array of size (|T1[q01]| + |T2[q02]|)
while St1 ≠ ∅
    ((q1, q2), l) := top(St1)
    M := top(St2)
    if l ≠ ∅
        choose and remove (q1', q2') in l
        if (q1', q2') ∉ St1  { this is a new state }
            if ¬ ((q1', q2') --φ--> T fail)
                push {(q1', q2'), succ(q1', q2')} in St1
                push into St2 a bit_array of size (|T1[q1']| + |T2[q2']|)
            endif
        else  { the current sequence is terminated }
            M[q1'] := 1 ; M[q2'] := 1
        endif
    else  { l = ∅, then backtrack }
        pop(St1) ; pop(St2)
        M' := top(St2)
```

```
            if M[q'] = 1 for all q' in (T1[q1] ∪ T2[q2])  { q1 ~ q2 }
                M'[q1] := 1 ; M'[q2] := 1
            endif
        endif
    endwhile
    M := top(St2)
    if M[q01] = 1 and M[q02] = 1
        return TRUE  { q01 ~ q02 }
    else
        return FALSE  { q01 ≁ q02 }
    endif
end.
```

Let c_i be the maximum number of successors per state of S_i ($c_i = \max\{|T_{ia}[q]|$, for $a \in A_i$ and for $q \in Q_i\}$ $i = 1, 2$), and let c be the maximum number of successors per state of $S_1 \times S_2$ ($c \leq c_1 \times c_2$). Let d be the number of states of the longest elementary sequence of $S_1 \times S_2$.
The number n^* of states visited by the Algorithm 2 is then in the range of $n \leq n^* \leq c^d$ ([Hol90]). Consequently, the time requirement for Algorithm 2 is of order of $O(n^*)$, and the memory requirement is $O(d)$.

Notation:

In the sequel, we call the *status* of a state the result of the analysis of this state by an algorithm. The status of (q_1, q_2) is "$\sim$" if q_1 and q_2 are found bisimilar, and is "$\not\sim$" otherwise. $\square$.

In order to reduce the exponential complexity of Algorithm 2, the usual method would consist, as in Algorithm 1, in storing all the visited states together with their status (including those which do not belong to the current sequence). Unfortunately, this solution cannot be directly 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 status of such a state is unknown (it is not available yet).
We propose the following solution for this problem: 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.
The data structures required are two stacks St_1 and St_2 (as in Algorithm 2), and three sets, R,V, and W:
The set V is intended to mark all the visited states, the set R is used for storing all the states of the current sequence which were visited more than once, and the set W contains all the states for which the obtained status is "$\not\sim$".
We consider the function *partial_DFS*, which performs the same DFS as in Algorithm 2, but storing 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 consists in a sequence of calls of *partial_DFS* (each call increasing the set W), until the result belongs to $\{TRUE, FALSE\}$.

Algorithm3

```
W := ∅
repeat
    result := partial_DFS  { perform a DFS }
until result ∈ {TRUE, FALSE}
return result
end.
```

function partial_DFS

```
V := ∅ ; R := ∅ ; stable := false
St1 := {(q01, q02), succ(q01, q02)}
St2 := ∅
push into St2 a bit_array of size 2  { in order to deal with (q01, q02) }
push into St2 a bit_array of size (|T1[q01]| + |T2[q02]|)
while St1 ≠ ∅
    stable := true
    ((q1, q2), l) := top(St1)
    M := top(St2)
    if l ≠ ∅
        choose and remove (q'1, q'2) in l
        if (q'1, q'2) ∉ V ∪ W
            if (q'1, q'2) ∉ St1  { it's a new state }
                if ¬ ((q'1, q'2) --φ-->T fail)
                    push {(q'1, q'2), succ(q'1, q'2)} in St1
                    push into St2 a bit_array of size (|T1[q'1]| + |T2[q'2]|)
                endif
            else  { (q'1, q'2 ∈ St1) }
                insert (q'1, q'2) in R  { this state has been visited more that once }
                M[q'1] := 1 ; M[q'2] := 1
            endif
        else  { (q'1, q'2) ∈ V ∩ W (i.e, visited in a previous DFS) }
            if (q'1, q'2) ∉ W
                M[q'1] := 1 ; M[q'2] := 1  { q'1 ∼ q'2 }
            endif
        endif
    else  { l ≠ ∅ }
        pop(St1) ; pop(St2)
        insert (q1, q2) in V  { a new state has been analyzed }
        M' := top(St2)
        if M[q'] = 1 for all q' in (T1[q1] ∪ T2[q2])
            M'[q1] := 1 ; M'[q2] := 1 {q1 ∼ q2 }
        else
            insert (q1, q2) in W  { q1 ≁ q2 }
            if (q1, q2) ∈ 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
        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.
```

Proposition 3.3 *Algorithm 3 terminates, and it returns TRUE if and only if the two labeled transition systems are bisimilars.*

Proof We use the following notations :
Let DFS_i representing the ith 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 3 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 3.
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), it follows:
$R_k \cap W_k = \emptyset \ \vee \ DFS_k$ 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 labeled transition systems aren't bisimilar.

$\square$.

Let n be the number of states of $S_1 \times S_2$.
The time requirement for the function $partial_DFS$ is $O(n)$. In the worst case, as pointed out in the proof of proposition 3.3 the number of calls of this function may be n. Consequently, the theoretical time requirement for this algorithm is $O(n^2)$. In the following section, we show that, for all the examples considered, only one DFS was needed in order to obtain a reliable result. Moreover, whenever the labeled transition systems 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 and *randomly accessed memory*. As pointed out in [CVWY90], it is the size of the randomly accessed memory which is critical.
In our case, only the sets R, V and W are randomly accessed (i.e, about 3 bits per state), while the stacks St_1 and St_2 are sequentially accessed (and consequently can be implemented in secondary memory).

4 Applications

The decision procedure for the strong bisimulation given in the previous section can be adapted in order to deal with weaker equivalence relations. In this section, we describe the modifications needed in order to deal with the safety equivalence. We show that it is sufficient to transform only the definition of the product of the two labeled transition systems (and consequently the function *succ*). From this modified product, the algorithms of the previous section can be applied in order to decide of the safety equivalence of the two systems.
Then we give some results obtained by applying this algorithm to labeled transition systems generated from LOTOS specifications.

4.1 Safety Equivalence

Several weaker relations, based on the bisimulation, have been proposed in order to deal with the unobservable action $\tau \notin A$ (as observational equivalence [Mil80], branching bisimulation [GW89]).
Safety equivalence (introduced in [Rod88]) is an equivalence relation preserving *safety* properties of systems, and which is therefore interesting in connection with a temporal logic. However, the relation described in [Rod88] is based on a safety preorder, and we consider here a stronger bisimulation-based relation.
Safety equivalence, denoted $\approx_s$, can as strong bisimulation be defined as the intersection of a sequence of decreasing relations:

Definition 4.1 $\approx_s = \cap \approx_s^i$ *for* $i \in N$, *where* $\approx_s^i$ *are defined by:*

- $\forall p, q \in Q \,.\, p \approx_s^0 q$,
- $p_1 \approx_s^{i+1} p_2$ *if and only if* $\forall a \in A$.
  $\forall r_1 \,.\, (p_1 \xrightarrow{\tau^* a}_T r_1 \Rightarrow \exists r_2 \,.\, (p_2 \xrightarrow{\tau^* a}_T r_2 \wedge r_1 \approx_s^i r_2)) \wedge$
  $\forall r_2 \,.\, (p_2 \xrightarrow{\tau^* a}_T r_2 \Rightarrow \exists r_1 \,.\, (p_1 \xrightarrow{\tau^* a}_T r_1 \wedge r_1 \approx_s^i r_2))$.

As in the case of the strong bisimulation, in order to obtain an algorithm performing "on the fly" verification of safety equivalence, First we define a new product between S_1 and S_2 (noted $S_1 \times_s S_2$). We show that, the relation $\approx_s$ can then be expressed in terms of the execution sequences of this product.
The definition of $S_1 \times_s S_2$ is similar to definition 3.1: it is straightly obtained by replacing the relation $\xrightarrow{a}_{T_i}$ by $\xrightarrow{\tau^* a}_{T_i}$, for $a \neq \tau$ (i = 1,2).

Definition 4.2 *Let $S_i = (Q_i, A_i, T_i, q_{0i})$ for i = 1,2 be two labeled transition systems. We define the labeled transition system $S = S_1 \times_s S_2$ such that:*
$S = (Q, A, T, (q_{01}, q_{02}))$, with $Q \subseteq (Q_1 \times Q_2) \cup \{fail\}$, and $A = (A_1 \cap A_2) \cup \{\phi\}$, where $fail \notin (A_1 \cup A_2)$ and $\phi \notin (Q_1 \cup Q_2)$.
T and Q are the smallest sets obtained by application of the following rules:

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

$$\frac{(q_1, q_2) \in Q,\ a \in A - \{\tau\},\ q_1 \xrightarrow{\tau^* a}_{T_1} q_1',\ q_2 \xrightarrow{\tau^* a}_{T_2} q_2'}{\{(q_1', q_2')\} \in Q, \{(q_1, q_2) \xrightarrow{a}_T (q_1', q_2')\} \in T} \qquad [R1]$$

$$\frac{(q_1, q_2) \in Q,\ a \in A - \{\tau\},\ q_1 \xrightarrow{\tau^* a}_{T_1} q_1',\ \neg(q_2 \xrightarrow{\tau^* a}_{T_2} q_2')}{\{fail\} \in Q, \{(q_1, q_2) \xrightarrow{\phi}_T fail\} \in T} \qquad [R2]$$

$$\frac{(q_1, q_2) \in Q,\ a \in A - \{\tau\},\ q_2 \xrightarrow{\tau^* a}_{T_2} q_2',\ \neg(q_1 \xrightarrow{\tau^* a}_{T_1} q_1')}{\{fail\} \in Q, \{(q_1, q_2) \xrightarrow{\phi}_T fail\} \in T} \qquad [R3]$$

The following proposition is similar to proposition 3.1:

Proposition 4.1 *Let $S_i = (Q_i, A_i, T_i, q_{0i})$ for i = 1,2 be two labeled transition systems, and let $S = (Q, A, T, q_0)$ be the product $S_1 \times_s S_2$.*
Then, $q_{01} \not\approx_s q_{02}$ if and only if there exists an elementary execution sequence σ of S such that:

- $\sigma = \{(q_{01}, q_{02}) = (p_1, q_1), (p_2, q_2),\ \ldots\ (p_k, q_k), fail\}$.
- $\forall i\ .\ 1 \leq i \leq k, p_i \not\sim^{k-i} q_i$

According to this proposition, it is obvious that the algorithms obtained in the previous section can be applied in order to decide of safety equivalence of two systems. Only the function *succ* used in the algorithm has to be redefined.
Moreover, one can notice that only the states which are reachable by the relation $\xrightarrow{\tau^* a}_{T_1}$ and $\xrightarrow{\tau^* a}_{T_2}$ ($a \neq \tau$) belong to the product S. Consequently, in most examples, the number of states analyzed during a DFS (and therefore the time needed) is much smaller than $n_1 \times n_2$ (which is the theoretical upper bound).

4.2 Results

The decision procedure for safety equivalence described above has been implemented in ALDÉBARAN. In this section, we give the results obtained when applying it to LOTOS examples.
In this draft implementation, the verification is not performed "on the fly" directly from the LOTOS specifications: the labeled transition systems are previously generated and the verification phase consists then in simultaneously building the product and deciding whether or not the equivalence holds, as described in the algorithms.
Moreover, the obtained results can be compared with the classical verification procedure, also implemented in ALDÉBARAN.

Three examples are studied here: the first one is the well known scheduler described by Milner in [Mil80], the second one is an alternating bit protocol called Datalink protocol [QPF88], and the last one is a real example, the rel/REL_{fifo} protocol [SE90].
For each example, the verification was performed as follows:

- generating the labeled transition system S_1 (*Implementation*) from the LOTOS description, using CÆSAR.
- building the labeled transition system S_2 (*Specification*), representing the expected behaviour of the system.
- comparing S_1 and S_2 with respect to safety equivalence, using the usual decision procedure of ALDÉBARAN and the improved one described in this paper (Algorithm 3).

The following notations are used:

- n_i and m_i denote the number of states and transitions of the two labeled transition systems ($i = 1, 2$).
- 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. Only the verification phase is taken into account.

4.2.1 Milner's scheduler

The problem consists in designing a scheduler which ensures that N communicating processes start a given task in a cyclic way. The LOTOS specification considered has been straightly obtained from Milner's CCS solution.
The results are given for different values of N.

N	n1	m1	n2	m2	n	t1	t2
7	1345	5377	7	7	449	0:14	0:11
8	3073	13825	8	8	1025	0:46	0:34
9	6913	34561	9	9	2305	2:50	1:44
10	15361	84481	10	10	5121	13:07	5:25

4.2.2 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 which can be transmitted (noted N), labeled transition systems of different sizes can be obtained. 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.

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

4.2.3 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 [BM90]. 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 labeled transition system of 680 000 states and 1 900 000 transitions has been obtained using CÆSAR. The **Fifo** requirement has been verified by comparing (with respect to safety equivalence) this labeled transition system in which only the actions performed by one receiver were visible, with the expected behaviour 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 Algorithm 3 in less than 3 hours on a HP-9000 Workstation.

5 Conclusion

Two applications can be obtained directly from the algorithm described in this paper.
First, it can be viewed as a new decision procedure (in the usual sense) for bisimulation-based equivalence relations between labeled transition systems.
The results obtained for safety equivalence, from a draft implementation, show that, at least for

this relation, this algorithm is more efficient than the usual one. Moreover, as this algorithm requires less memory, verifications of larger size labeled transition systems become possible. Furthermore, this algorithm allows to perform "on the fly" verifications: in relation with an efficient "simulator" (i.e., a tool able to generate a labeled transition system from a LOTOS specification), it is possible to compare LOTOS specifications (with respect to an equivalence relation) without explicitly storing the whole labeled transition system. Thus, the verification could be integrated in the simulation phase, significantly reducing the amount of memory needed. Consequently, checking of real size examples could be carried out.

Acknowledgements

The authors whish to thank Joseph Sifakis, who suggested this work, for his helpful comments, and Suzanne Graf who accepted to read earlier versions of this paper.

References

[BHR84] S. D. Brookes, C.A.R Hoare, and A.W. Roscoe. Theory of Communicating Sequential Processes. *JACM*, 31(3), 1984.

[BM90] Simon Bainbridge and Laurent Mounier. *Specification and Verification of a Reliable Multicast Protocol*. Technical Report (In preparation), Hewlett-Packard Laboratories, Bristol, U.K, 1990.

[CVWY90] C. Courcoubetis, M. Vardi, P. Wolper, and M. Yannakakis. Memory Efficient Algorithms for the Verification of Temporal Properties. (to appear,1990).

[Fer88] J. C. Fernandez. *Aldébaran, Un système de vérification par réduction de processus communicants*. PhD thesis, Université de Grenoble, 1988.

[Fer89] J. C. Fernandez. *Aldébaran: A tool for verification of communicating processes*. Technical Report SPECTRE c14, LGI-IMAG Grenoble, 1989.

[Gar89] H. Garavel. *Compilation et vérification de programmes LOTOS*. PhD thesis, Université Joseph Fourier de Grenoble, 1989.

[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.

[GW89] R.J. Van Glabbeek and W.P. Weijland. *Branching time and abstraction in bisimulation semantics (extended abstract)*. CS-R 8911, Centrum voor Wiskunde en Informatica, Amsterdam, 1989.

[Hol90] Gerard J. Holzmann. *Algorithms for Automated Protocol Validation*. Technical Report 69:32-44, AT&T Technical Journal, January 1990.

[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.

[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.