A Tool Set for deciding Behavioral Equivalences *

Jean-Claude Fernandez † Laurent Mounier†

Abstract

This paper deals with verification methods based on equivalence relations between labeled transition systems. More precisely, we are concerned by two practical needs: how to efficiently minimize and compare labeled transition systems with respect to bisimulation or simulation-based equivalence relations.

First, we recall the principle of the classical algorithms for the existing equivalence relations, which are based on successive partition refinements of the state space of the labeled transition systems under consideration. However, in spite of their theoretical efficiency, the main drawback of these algorithms is that they require to generate and to store in memory the whole labeled transition systems to be compared or minimized. Therefore, the size of the systems which can be handled in practice remains limited. We propose here another approach, allowing to combine the generation and the verification phases, which is based on two algorithms respectively devoted to the comparison (*"on the fly" comparison*) and the minimization (*minimal model generation*) of labeled transition systems. Then, we present the results obtained when implementing some of these algorithms within the tool ALDÉBARAN.

1 Introduction

This paper deals with some methodological considerations on tools associated with verification methods of distributed systems. More precisely, we aim to relate our experiments with the implementation within the tool ALDÉBARAN of various decision procedures for behavioral equivalence relations.

By *verification*, we mean the comparison of a *system description*, i.e., a program, noted $\mathcal{D}$, with its *specifications*, noted $\mathcal{S}$, namely the description of its expected properties. A program semantics is defined by a congruence on a class of program models. Examples of classes of program models are Labeled Transition Systems (LTS for short), event structures, Petri nets, etc ... In this paper, we are concerned by LTS and equivalence or preorder relations between LTS. According to the formalism used for the specifications, two main verification approaches can be distinguished:

*This work was partially supported by ESPRIT Basic Research Action "SPEC" and french project research C^3

†LGI-IMAG, IMAG Campus, BP 53X, 38041 GRENOBLE cedex, FRANCE; e-mail: {fernand, mounier}@imag.imag.fr

from a certain abstraction level. Such specifications can be also expressed in terms of LTS. In this case, the verification consists in comparing the two LTS $\mathcal{D}$ and $\mathcal{S}$ with respect to a given equivalence or preorder relation. Thus, any decision procedure for these relations defines a verification method.

Logical specifications : they characterize the *global properties* of the system, such as deadlock freedom, mutual exclusion, or fairness. *Temporal logics* are suitable formalisms, since they allow to describe the whole system evolution. In this case, a formula of $\mathcal{S}$ is interpreted as a set of *computations* on $\mathcal{D}$. For example, in linear time semantics a computation is a maximal sequence, whereas in branching times semantics a computation is a tree. Then, verification consists in checking that all the computations of $\mathcal{D}$ belong to $\mathcal{S}$.

These approaches are complementary: it turns out in practice that some of the expected properties of a distributed system are easier to express using a logical formalism, and the others by giving an abstraction of the expected behavior. Moreover, a logic can *characterize* a behavioral relation: two systems are related if and only if they satisfy the same set of logical formulas. As a consequence, a practical verification method associated with such a logic consists in minimizing first the LTS $\mathcal{D}$ with respect to the corresponding behavioral relation, and then checking that all the computations of this reduced LTS belong to $\mathcal{S}$. For example, branching bisimulation is characterized by a fragment of CTL^* [NV90], and safety equivalence is characterized by a fragment of the μ-calculus [BFG*91].

The principle of the usual LTS-based verification methods is the following: first generate a LTS from the program description, and then apply the decision procedure associated with the considered specification formalism. Many automated verification tools, [dSV89,GV90,CPS90,Fer90] for behavioral specifications and [RRSV87,CES86] for logical specifications, are based on this design. However, the main drawback of this method is that the whole LTS have to be stored in memory, and therefore the size of the graphs which can be compared or minimized is limited. For ALDÉBARAN, running on a workstation, the maximal size of LTS which can be treated is of the order of one million of states. Consequently, an attractive solution would be to combine the generation and verification phases: the verification is performed during the generation, without keeping in memory the sets of states and transitions of the whole LTS ("on the fly" verification). Several algorithms have been proposed to implement such a verification method for temporal logics [JJ89,BFH90a,CVWY90].

Applying the "on the fly" approach to the verification of behavioral specification seems promising. For this purpose, we study currently the implementation of suitable algorithms for bisimulation-based equivalence and preorder relations within ALDÉBARAN, in association with the LOTOS compiler CÆSAR [GS90]. More precisely, given a LOTOS program we consider its abstract representation, which can be either a Petri net or a set of communicating automata. From this representation, we intend to cover the two practical needs:

"on the fly" LTS comparison: the algorithm consists in performing the comparison during a depth-first traversal of a product of the two transitions functions (the

this product is parametrized by the behavioral relation under consideration. This method has been successfully applyed for several relations, like strong bisimulation, strong simulation, delay bisimulation, branching bisimulation (when the specification is τ-free, i.e., without τ-action), safety equivalence and safety preorder.

minimal LTS generation: the principle is to refine a partition of the reachable state space of the abstract representation of the program, and to compute its transition relation at the same time. This method is based on symbolic computations, which impose some restrictions on the types of the values used in the LOTOS program in order to obtain an efficient implementation.

This paper is organized as follows: first we give the definitions used in the following sections. In section 3, we recall some definitions of equivalence and preorder relations. In section 4, we survey the usual methods for the minimization and comparison of LTS. In section 5, we present the principle of the "on the fly" verification methods, and we give in section 6 the state of the experiments currently realized in ALDÉBARAN.

2 Equivalences Relations

Equivalence relations for distributed systems can be defined in several ways, according to their semantics and the description formalism upon which they are based. For example, trace equivalence has been defined for automata, readiness and failures semantics for CSP [BHR84], observation bisimulation [Mil80], branching bisimulation [GW89] and strong bisimulation [Par81] for process algebra such as CCS or ACP. However, as these description formalisms can be translated in terms of LTS, all these relations can also be expressed as equivalence relations between LTS.

We will focus here on some bisimulation or simulation based equivalences defined on LTS which, when ordered by the relation "finer than", are positioned between *bisimulation equivalence* and *trace equivalence*. We first give some notations, and then we recall the definitions of the relations that we will consider in the following. Finally, we propose a practical characterization of the "non-bisimilarity" between two LTS.

2.1 Labeled Transition Systems

Let *States* be a set of states, A a set of names (of actions), τ a particular name not in A, which represents an internal or hidden action and $A_\tau = A \cup \{\tau\}$. For a set X, X^* will represent the set of finite sequences on X.

In the following, we consider a LTS $S = (Q, A_\tau, \{\xrightarrow{\alpha}\}_{\alpha \in A_\tau}, q_{\text{init}})$ where: Q is the subset of *States*, $\longrightarrow \subseteq Q \times A_\tau \times Q$ is a labeled transition relation, and q_{init} is the initial state.

We will also consider the usual pre- and post-conditions functions from 2^Q to 2^Q, where X (resp. B) is a subset of Q (resp A_τ):

$$\begin{aligned} pre_\alpha(X) &= \{q \in Q \mid \exists q' \,.\, q \xrightarrow{\alpha} q' \ \wedge \ q' \in X\} \\ post_\alpha(X) &= \{q \in Q \mid \exists q' \,.\, q' \xrightarrow{\alpha} q \ \wedge \ q' \in X\} \end{aligned}$$

$$post_B(X) = \bigcup_{\alpha \in B} post_\alpha(X)$$

$\mathcal{A}\mathrm{ct}(q)$ will denote the set of the actions which can be performed in a state q:

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

Let $\lambda \subseteq A^*$, and let $p, q \in Q$. We write $p \xrightarrow{\lambda} q$ if and only if:

$$\exists u_1 \cdots u_n \in \lambda \ \wedge \ \exists q_1, \cdots, q_{n-1} \in Q \ \wedge \ p \xrightarrow{u_1} q_1 \xrightarrow{u_2} q_2 \cdots q_i \xrightarrow{u_{i+1}} q_{i+1} \cdots q_{n-1} \xrightarrow{u_n} q.$$

Let Λ be a family of disjoint languages on A_τ.

$$\mathcal{A}\mathrm{ct}^\Lambda(q) = \{\lambda \in \Lambda \mid \exists q' \,.\, q \xrightarrow{\lambda} q'\}.$$

The set of the finite execution sequences from a state q of Q (noted $Ex(q)$) is defined as follows:

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

In the following, for a LTS S, the term *execution sequences* of S represents the set $Ex(q_{\mathrm{init}})$. Furthermore, an execution sequence is said *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 Bisimulations and Simulations

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

Definition 2.1 *(simulation) For each $R \in Q \times Q$, we define:*

$$\mathcal{I}^\Lambda(R) = \{(p_1, p_2) \mid \forall \lambda \in \Lambda \,.\, \forall q_1 \,.\, (p_1 \xrightarrow{\lambda} q_1 \Rightarrow \exists q_2 \,.\, (p_2 \xrightarrow{\lambda} q_2 \ \wedge \ (q_1, q_2) \in R))\}$$

The simulation preorder $\sqsubseteq^\Lambda$ for the language Λ is defined as the greatest fixed-point of $\mathcal{I}^\Lambda$ and the simulation equivalence is $\approx^\Lambda = \sqsubseteq^\Lambda \ \cap (\sqsubseteq^\Lambda)^{-1}$.

Definition 2.2 *(bisimulation) For each $R \in Q \times Q$, we define:*

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

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

Definition 2.3 *(branching bisimulation) For each $R \in Q \times Q$, we define:*

$$\begin{aligned}\mathcal{B}^\Lambda_{br}(R) \ = \ \{(p_1, p_2) \mid \forall \lambda \in \Lambda \,. \\ \forall q_1 \,.\, (p_1 \xrightarrow{\lambda} q_1 \Rightarrow (\lambda = \tau \ \wedge \ (q_1, p_2) \in R) \ \vee \\ (\exists q_2 q_2' \,.\, (p_2 \xrightarrow{\tau^*} q_2' \ \wedge \ q_2' \xrightarrow{\lambda} q_2 \ \wedge \ (p_1, q_2') \in R \ \wedge \ (q_1, q_2) \in R))) \\ \forall q_2 \,.\, (p_2 \xrightarrow{\lambda} q_2 \Rightarrow (\lambda = \tau \ \wedge \ (p_1, q_2) \in R) \ \vee \\ (\exists q_1 q_1' \,.\, (p_1 \xrightarrow{\tau^*} q_1' \ \wedge \ q_1' \xrightarrow{\lambda} q_1 \ \wedge \ (q_1', p_2) \in R \ \wedge \ (q_1, q_2) \in R)))\}\end{aligned}$$

est fixed-point of $\mathcal{B}^{\Lambda}_{br}$.

From these general definitions, several simulation and bisimulation relations can be defined. The choice of a class Λ corresponds to the choice of an *abstraction criterion* on the actions. The *strong simulation* and the *strong bisimulation* [Par81] are defined by $\Lambda = \{\{a\} \mid a \in A\}$, the *w bisimulation* [FM91] is the bisimulation equivalence defined by $\Lambda = \{\tau^* a \mid a \in A\}$, the *safety preorder* [BFG*91] is the simulation preorder defined by $\Lambda = \{\tau^* a \mid a \in A\}$ and the *safety equivalence* is the simulation equivalence where $\Lambda = \{\tau^* a \mid a \in A\}$. Observation equivalence [Mil80] is the bisimulation equivalence defined by $\Lambda = \tau^* \cup \{\tau^* a \tau^* \mid a \in A\}$. Delay bisimulation [NMV90] is the bisimulation equivalence defined by $\Lambda = \tau^* \cup \{\tau^* a \mid a \in A\}$. Branching bisimulation [GW89] is the branching bisimulation where $\Lambda = \{\{a\} \mid a \in A\}$.

Remark Note that when we consider the languages $\Lambda_1 = \{\tau^* a \mid a \in A\}$ or $\Lambda_2 = \{\tau^* a \tau^* a \mid a \in A\}$ or $\Lambda_3 = \Lambda_1 \cup \tau^*$ or $\Lambda_4 = \Lambda_2 \cup \tau^*$, we obtain the same preorder against bisimulations. □

Each equivalence relation R^{Λ} defined on states can be extended to an equivalence relation comparing LTS in the following manner: let $S_i = (Q_i, A_\tau, \{\xrightarrow{\alpha}\}_{\alpha \in A_\tau}, q^i_{\text{init}})$, 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 $S_1 \; R^{\Lambda} \; S_2$ if and only if $(q^1_{\text{init}}, q^2_{\text{init}}) \in R^{\Lambda}$ and $S_1 \not R^{\Lambda} \; S_2$ if and only if $(q^1_{\text{init}}, q^2_{\text{init}}) \notin R^{\Lambda}$.

2.3 Other Equivalences

In this section, we consider readiness semantics, failure semantics and other semantics of CSP. We do not present these equivalences in detail.

Definition 2.4 Failure semantics:
$(\sigma, X) \in A^* \times 2^{A_\tau}$ is a failure pair for a LTS $(Q, A_\tau, \{\xrightarrow{\alpha}\}_{\alpha \in A_\tau}, q_{\text{init}})$ if there is $q \in Q$ such that $q_{\text{init}} \xrightarrow{\sigma} q$ and $X \cap \mathcal{A}ct(q) = \emptyset$. Let $F(q)$ denote the set of failure pairs of q.
Two LTS $S_i = (Q_i, A_\tau, \{\xrightarrow{\alpha}\}_{\alpha \in A_\tau}, q^i_{\text{init}})$, for $i = 1, 2$ are failure equivalent if $F(q^1_{\text{init}}) = F(q^2_{\text{init}})$.

Definition 2.5 Readiness semantics:
$(\sigma, X) \in A^* \times 2^{A_\tau}$ is a ready pair for a LTS $(Q, A_\tau, \{\xrightarrow{\alpha}\}_{\alpha \in A_\tau}, q_{\text{init}})$ if there is $q \in Q$ such that $q_{\text{init}} \xrightarrow{\sigma} q$ and $X = \mathcal{A}ct(q)$. Let $R(q)$ denote the set of ready pairs of q.
Two LTS $S_i = (Q_i, A_\tau, \{\xrightarrow{\alpha}\}_{\alpha \in A_\tau}, q^i_{\text{init}})$, for $i = 1, 2$ are ready equivalent if $F(q^1_{\text{init}}) = F(q^2_{\text{init}})$.

Other variants of these equivalences can be found in [Gla90], in which states of transitions systems can be labeled by subset of actions.

2.4 Expressing non bisimilarity

Several formalisms have been proposed in order to express the "non bisimilarity" of two labeled transition systems (for example Hennessy-Milner Logic in [Cle90]). We present

(both denoted by R^{Π}).

Let $S_i = (Q_i, A_\tau, \{\xrightarrow{\alpha}\}_{\alpha \in A_\tau}, q^i_{\text{init}})$, for $i = 1, 2$, be two LTS. Whenever the two labeled transition systems S_1 and S_2 are not related, we define an *explanation sequence* as an execution sequence σ of a *synchronous product* $S = S_1 \times_{R_\Pi} S_2$ (see section 4 for a precise definition of this product) such that:

- All the states (p_i, q_i) belonging to σ (where p_i is a state of S_1 and q_i is a state of S_2) are not comparable against R^{Π}.
- σ is terminated by a state which is not in R^{Π}_1 (i.e, from which it clearly appears that S_1 and S_2 are not related)

Definition 2.6 *An explanation sequence of* $S_1 \not R^{\Pi} S_2$ *is an execution sequence* σ *such that:*

- $\sigma = \{(q_{01}, q_{02}) = (p_1, q_1), (p_2, q_2), ..., (p_k, q_k)\}$
- $\forall i \; . \; 0 \leq i \leq k, \; (p_i, q_i) \notin R^{\Pi}_{k-i+1}$.
- $(p_k, q_k) \notin R^{\Pi}_1$

This definition is motivated by the following propositions, which allow 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 2.1 $(q^1_{\text{init}}, q^2_{\text{init}}) \notin R^{\Pi}$ *if and only if it exists an elementary execution sequence* σ *of* S *(*$\sigma \in Ex_e(q_{01}, q_{02})$*) such that:*

- $\sigma = \{(q^1_{\text{init}}, q^2_{\text{init}}) = (p_0, q_0), (p_1, q_1), \; ... \; (p_k, q_k)\}$.
- $\forall i \; . \; 0 \leq i \leq k, \; (p_i, q_i) \notin R^{\Pi}_{k-i+1}$.

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

Lemma 2.1 *Let* $S = (Q, A, T, q_0)$ *be a labeled transition system. Then we have,*
$\forall k \geq 1, \forall p, q \in Q, \;\; (p, q) \notin R^{\Pi}_{k+1} \; \wedge \; (p, q) \in R^{\Pi}_k \Rightarrow$

$$\exists \lambda \in \Pi \; . \; \exists p' \; . \; \exists q' \; . \; p \xrightarrow{\lambda}_T p' \; \wedge \; q \xrightarrow{\lambda}_T q' \; \wedge \; (p', q') \notin R^{\Pi}_k \; \wedge \; (p', q') \in R^{\Pi}_{k-1}.$$

If one of the two labeled transition systems is deterministic, proposition 2.1 can be improved. In this case, the converse of lemma 2.1 holds: $(q_1, q_2) \in R^{\Pi}_k$ if and only if all the successors (q'_1, q'_2) of (q_1, q_2) verify $(q_1, q_2) \in R^{\Pi}_{k-1}$ and $(q_1, q_2) \in R^{\Pi}_1$.

Lemma 2.2 *Let us suppose that* S_1 *or* S_2 *is deterministic (*S_1 *if the* $(R^{\Pi}_k)_{k \geq 0}$ *are simulations).*
$\forall k \geq 1, \forall p, q \in Q,$

$$\exists \lambda \in \Pi \; . \; \exists p' \; . \; \exists q' \; . \; p \xrightarrow{\lambda}_T p' \; \wedge \; q \xrightarrow{\lambda}_T q' \; \wedge \; (p', q') \notin R^{\Pi}_k \Rightarrow (p, q) \notin R^{\Pi}_{k+1}$$

From this lemma, we can deduce proposition 2.2:

Proposition 2.2 *Let us suppose that* S_1 *or* S_2 *is deterministic (*S_1 *if the* $(R^{\Pi}_k)_{k \geq 0}$ *are simulations). Then:*
$S_1 \not R^{\Pi} S_2 \Leftrightarrow \exists \sigma \; . \; \sigma = \{(q^1_{\text{init}}, q^2_{\text{init}}) = (p_0, q_0), (p_1, q_1), \; ... \; (p_k, q_k)\} \; \wedge \; (p_k, q_k) \notin R^{\Pi}_1$.

All the proof can be found in [FM91].

The usual comparison and minimization methods of LTS with respect to an equivalence relation are based on the structure of the LTS and are independent from the generation techniques.

minimization: For a given bisimulation-based equivalence relation, a *normal-form* of a LTS S (i.e, the smallest LTS in number of states and transitions equivalent to S) can be obtained by applying the two following steps:

1. Compute a pre-normal form S' of S by transforming the transition relation of S. This step depends on the equivalence relation under consideration. Examples of transformations are various transitive closure computations of the τ-transition relation (τ-*saturation phase*), or determinization .
2. Compute the normal-form of S' with respect to strong bisimulation. This is obtained by solving the *RCP* problem (see 3.2) on the states of S' using a partition refinement algorithm. The obtained LTS is then the normal-form of S.

For the particular case of branching bisimulation, the first step can be avoided since a normal-form can be straightly obtained by solving the *GRCP* problem (see 3.3) on S.

comparison: Two LTS can be compared either by checking if their normal-form are identical or by computing the normal-form or their union and checking if the initial states of the original LTS belong or not to the same equivalence class.

In this section, we describe the classical algorithms for the computation of normal-forms with respect to the different equivalence relations presented in the previous section.

3.1 Partitions

Let $S = (Q, A_\tau, \{\xrightarrow{\alpha}\}_{\alpha \in A_\tau}, q_{init})$. We consider partitions of Q instead of equivalence relations on Q: let ρ be a partition (a set of pairwise disjoint non-empty subsets X_i such that $\bigcup_i X_i = Q$), then the induced equivalence is $p \sim_\rho q$ if and only if $\exists X_i \in \rho$ such that $p \in X_i \ \wedge \ q \in X_i$.

Lattices Let $\mathcal{P}$ be the set of partitions of Q. We consider the refinement relation, noted $\sqsubseteq$ over $\mathcal{P}$:

$$\rho \sqsubseteq \rho' \text{ if and only if } \forall X \in \rho \,.\, \exists X' \in \rho' \,.\, X \subseteq X'.$$

With this order, $\mathcal{P}$ is a complete lattice, with the greatest lower bound operator $\sqcap$ defined by:

$$\bigsqcap_i \rho_i = \{T \neq \emptyset \mid T = \bigcap_i X_i \text{ and } X_i \in \rho_i\}$$

with least upper bound,

$$\bigsqcup_i \rho_i = \bigsqcap_{\forall i, \rho_i \sqsubseteq \rho} \rho$$

class of the partition ρ containing the state q. Let $pre_{a,\rho}$ and $post_{a,\rho}$ denote the pre and post-conditions functions corresponding to a partition ρ:

$$\begin{aligned} pre_{a,\rho}(X) &= \{[q]_\rho \mid q \in pre_a(X)\} \\ post_{a,\rho}(X) &= \{[q]_\rho \mid q \in post_a(X)\} \end{aligned}$$

We also consider the lattices 2^Q or 2^{2^Q}.

Fixed-points Let F be an increasing total function, either from 2^Q to 2^Q or from 2^{2^Q} to 2^{2^Q} and let G be an increasing total function from $\mathcal{P}$ to $\mathcal{P}$. We denote by

- $\mu\pi.F(\pi)$ the least fixed-point of F with respect to the ordering $\subseteq$
- $\nu\pi.G(\pi)$ the greatest fixed-point of G with respect to the ordering $\sqsubseteq$

3.2 Strong Bisimulation

The computation of the normal-form of a LTS with respect to strong bisimulation can be expressed in terms of a partition refinement problem, known as the *Relational Coarsest Partition (RCP) problem*:

The *RCP* problem: it consists in *finding the coarsest stable refinement of an initial partition* ρ_{init} *of* Q. (A partition is stable if and only if it is an equivalence relation which is a bisimulation). The solution of this problem corresponds to the equivalence classes of Q for the strong bisimulation relation.

Kanellakis and Smolka studied first the connection between the *RCP* problem and the minimization of LTS up to strong bisimulation [KS83]. Then, an efficient algorithm was proposed by Paige and Tarjan [PT87] and implemented within the tool ALDÉBARAN [Fer90]. Its time and space complexities are respectively $O(m \log(n))$ and $O(m+n)$, where n and m are the number of states and transitions of S.

Starting with an initial partition of the state space, this algorithm proceeds by refining the current partition ρ until it becomes stable, according to the following definitions:

Splitting and refining: These functions are defined as follows:

$$\begin{aligned} split(X,Y) &= \bigsqcap_{a \in A_\tau} \{X \cap pre_a(Y), X \setminus pre_a(Y)\} \\ Ref(\rho,Y) &= \bigsqcap_{X \in \rho} split(X,Y) \\ Ref(\rho,\rho') &= \bigcup_{Y \in \rho'} Ref(\rho,Y) \end{aligned}$$

Stability: An subset X of Q is said to be stable with respect to ρ if and only if $X = Ref(\rho, X)$. A partition is stable if it is stable with respect to its elements.

The solution of the *RCP* problem consists in computing the greatest fixed-point

$$\nu\rho \,.\, \rho_{\text{init}} \sqcap Ref(\rho,\rho).$$

An algorithm in $O(mn)$ time can easily be derived. To improve this complexity, the Paige and Tarjan's idea is to keep track how blocks of the current partition are split into subblocks at each refinement step, in order to always process the "smaller half" subblock.

In [GV90], Groote and Vaandrager present a variant of the *RCP* problem: the *Generalized Relational Coarsest Partition with Stuttering (GRCP) problem*, which allow to compute the normal of a LTS with respect to branching bisimulation and stuttering equivalence. They also give an efficient algorithm, in $O(n(n+m))$ time and $O(m+n)$ space.

The *GRCP* problem: The *GRCP* problem can be stated in the same terms than the *RCP* problem. The difference lies in the notion of splitting.

As the Paige and Tarjan's one, the principle of the algorithm given in [GV90] is to refine a partition of the state space of the LTS until it becomes stable, according to the following splitting functions:

$$\begin{aligned} \mathcal{F}_a(X,Y) &= \mu Z.(X \cap pre_\tau(Z) \cup X \cap pre_a(Y)) \\ split(X,Y) &= \bigsqcap_{a \in A_\tau} \{\mathcal{F}_a(X,Y), X \setminus \mathcal{F}_a(X,Y)\} \text{ if } X \neq Y \\ split(X,X) &= \bigsqcap_{a \in A} \{\mathcal{F}_a(X,X), X \setminus \mathcal{F}_a(X,X)\} \end{aligned}$$

As for the *RCP* problem, the solution of the *GRCP* problem consists in computing the greatest fixed-point

$$\nu\rho \,.\, \rho_{\text{init}} \sqcap Ref(\rho,\rho).$$

Remark The algorithm described in [GV90] requires to suppress first the τ-cycles by finding the strongly connected components of the τ-relation. □

3.4 Observation Equivalence

The algorithm for computing the normal-form of a LTS S with respect to observation equivalence is the following:

1. First, compute the transitive closure of the τ-relation of S:
   let S' be the LTS $(Q, A_\tau, \{\overset{\alpha}{\longrightarrow}'\}_{\alpha \in A_\tau}, q_{init})$ where $q \overset{a}{\longrightarrow}' q'$ if and only if $q \overset{\tau^* a \tau^*}{\longrightarrow} q'$.
2. Then solve the *RCP* problem for S' with initial partition $\rho_{\text{init}} = \{Q\}$.

This algorithm can be implemented in $O(n^3)$ time and $O(m+n^2)$ space.

3.5 Safety equivalence

Safety equivalence is a simulation-based equivalence which preserves *Safety properties* [BFG*91]. Each equivalence class may be characterized by a formula of a fragment of a μ-calculus and conversely. The algorithm for computing the normal-form of a LTS S with respect to safety equivalences is the following:

1. Compute the pre-normal form $S' = (Q', A_\tau, \{\overset{\alpha}{\longrightarrow}\}'_{\alpha \in A_\tau}, q_{\text{init}})$ where:
   - $Q' = \{c(q) \mid q \in Q \,.\, c(q) = \{q' \mid q \overset{\tau^*}{\longrightarrow} q'\}\}$,

2. Solve the *RCP* problem for S' with initial partition $\rho_{\text{init}} = \{Q'\}$.

In practice, the sets $c(q)$ are computed by performing a depth-first search on the LTS S. Then the greatest simulation is computed on Q' and the *redundant transitions* $c(p) \xrightarrow{a}{}' c(q')$ such that $c(p) \xrightarrow{a}{}' c(q)$ and $c(q') \sqsubseteq c(q)$ are removed.

3.6 w bisimulation

This bisimulation-based relation is stronger than safety equivalence. The normal-form of a LTS with respect to this relation can be obtained in the two following way:

- either, compute the states $c(q)$, as in the safety equivalence, and the relation $c(p) \xrightarrow{a}{}' c(q)$ if $p \xrightarrow{\tau^* a} q$, and then solve the *RCP* problem with initial partition $\rho_{\text{init}} = \{Q\}$,
- or solve the *RCP* problem for the language $\Lambda = \{\tau^* a \mid a \in A\}$ with initial partition $\rho_{\text{init}} = \{Q\}$. The function *split* is then:

$$split(X, Y) = \bigsqcap_{a \in A_\tau} \{X \cap pre_{\tau^* a}(Y), X \setminus pre_{\tau^* a}(Y)\}.$$

The difference between Safety equivalence and w bisimulation only lies in the removal of redundant transitions.

3.7 Readiness and Failure Semantics

The algorithm for computing the normal-form of a LTS with respect to these equivalence proceeds as follows:

- determinization of the LTS and labeling each state by a powerset.
  Let $S' = (Q', A_\tau, \{\xrightarrow{\alpha}\}'_{\alpha \in A_\tau}, q_{\text{init}})$ the LTS obtained in this way:
  - $\{q_{\text{init}}\} \in Q'$,
  - if $X \in Q'$ then $post_a(X) \in Q'$.
  - $X \xrightarrow{a}{}' Y$ if and only if $Y = post_a(X)$.

  Let $\mathcal{L} : Q \longrightarrow 2^A$ the labeling function depending of the chosen equivalence relation (Ready, or Failure). Then, the labeling function $\mathcal{L}' : Q' \longrightarrow 2^{2^A}$ for S' is:

$$\mathcal{L}'(X) = \{\mathcal{L}(q) \mid q \in X\}.$$

- Then solve the *RCP* problem for S' with initial partition

$$\rho_{\text{init}} = \{C \mid X, Y \in Q' \,.\, X, Y \in C \text{ if and only if } \mathcal{L}'(X) = \mathcal{L}'(Y)\}.$$

In spite of their theoretical efficiency, the classical verification methods described in the previous section suffers from a practical limitation: they require to generate and to store in main memory the whole LTS of the program to be verified (i.e, its sets of states and transitions). Moreover, apart from branching and strong bisimulation, another serious drawback of these methods is the need for a "preprocessing phase" of the LTS (i.e, the computation of a pre-normal form) which turns out to be very time expensive and to increase the size of the original LTS.

However, most of the equivalence relations presented in section 2 can be dealt with using another approach. In this section, starting from an *abstract representation* of the program to be verified (i.e., a Petri net, a term of a process algebra, or a net of communicating LTS) we describe two aspects of an "on the fly" verification method:

On the fly comparison: We compare the *abstract representation* of the program and its behavioral specification (i.e, a LTS which characterize its expected behavior). Rather than translating the abstract representation into a LTS and comparing it to the specification, we construct a synchronous product between this partially translated LTS and the specification LTS. The comparison is performed during this construction, and we only need to store the states of this product.

minimal LTS generation We consider partitions of the whole state space of the abstract representation of the program (the reachable and unreachable states). For example, if $\mathcal{P}$ is a program with two variables $x_1, x_2 \in \mathbb{N}$, we will consider partitions of $\mathbb{N} \times \mathbb{N}$. Then, starting from the universal partition, we progressively refine the reachable classes of the current partition until stability.

4.1 "On the fly" comparison

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.

In the following, we consider two LTS $S_i = (Q_i, A_\tau, \{\xrightarrow{\alpha}\}_{\alpha \in A_\tau}, q^i_{\text{init}})$, for $i = 1, 2$. We use p_i, q_i, p'_i, q'_i to range over Q_i. R^Π and R^Π_k will denote either simulations or bisimulations ($R^\Pi = \bigcap_{k=0}^{\infty} R^\Pi_k$).

A synchronous product

We define the synchronous product $S_1 \times_{R_\Pi} S_2$ between the two LTS S_1 and S_2 in the following manner:

- a state (q_1, q_2) of $S_1 \times_{R_\Pi} S_2$ can perform a transition labeled by an action λ 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 λ.

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

transition labeled by λ, then the product has a transition from (q_1, q_2) to the sink state *fail*.

Definition 4.1 *We define the labeled transition system* $S = S_1 \times_{R_\Pi} S_2$ *by:*
$S = (Q, A, T, (q^1_{\text{init}}, q^2_{\text{init}}))$, *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.

$$(q^1_{\text{init}}, q^2_{\text{init}}) \in Q \qquad [R0]$$

$$\frac{(q_1, q_2) \in Q,\ \mathcal{A}\text{ct}_\Pi(q_1) = \mathcal{A}\text{ct}_\Pi(q_2),\ q_1 \xrightarrow{\lambda}_{T_1} q'_1,\ q_2 \xrightarrow{\lambda}_{T_2} q'_2}{\{(q'_1, q'_2)\} \in Q, \{(q_1, q_2) \xrightarrow{\lambda}_T (q'_1, q'_2)\} \in T} \qquad [R1]$$

$$\frac{(q_1, q_2) \in Q,\ q_1 \xrightarrow{\lambda}_{T_1} q'_1,\ T^2_\lambda[q] = \emptyset}{\{fail\} \in Q, \{(q_1, q_2) \xrightarrow{\phi}_T fail\} \in T} \qquad [R2]$$

$$\frac{(q_1, q_2) \in Q,\ q_2 \xrightarrow{\lambda}_{T_2} q'_2,\ T^1_\lambda[q] = \emptyset}{\{fail\} \in Q, \{(q_1, q_2) \xrightarrow{\phi}_T fail\} \in T} \qquad [R3 \text{ bisimulation}]$$

Let's notice that $(p_1, p_2) \xrightarrow{\phi}_T fail$ if and only if $(p_1, p_2) \notin R^\Pi_1$. According to the propositions given in section 2.4, the S_1 and S_2 are not bisimilar if and only if it exists an explanation sequence on $S_1 \times_{R_\Pi} S_2$. Similar propositions hold in case of non-similarity (see [FM91]).

Algorithms

We have expressed the bisimulation and the simulation between two labeled transition systems S_1 and S_2 in terms of the existence of a particular execution sequence of their product $S_1 \times_{R_\Pi} S_2$. We show that this verification can be realized by performing depth-first searches (DFS for short) on the labeled transition system $S_1 \times_{R_\Pi} S_2$. Consequently, the algorithm does not require to previously construct the two LTS: 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. Moreover, in the case where S_1 and S_2 are not related, explanation sequences are straightly obtained.

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 2.2, 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 ([JJ91]).

for 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: 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.

The detailed algorithm can be found in [FM91].

4.2 Minimal LTS Generation

This approach [BFH90b] combines the construction of the graph of accessibles states with its reduction by bisimulation equivalence. The termination of the algorithm imposes that the quotient of the whole space of accessible and unaccessible states by bisimulation is finite. The algorithm combines a least fixed-point (accessibility on the classes) and a greatest fixed-point (greatest bisimulation). In the classical method, the set of accessible states is computed and then, given an initial partition ρ_{init}, the current partition is refined until all the classes are stable. Another algorithm may be given, refining first the initial partition and computing the accessible classes (which are stable). In the intermediate method presented here, the set of accessible classes is computed and then refined until all the accessible classes are stable.

Let $S = (Q, A_\tau, \{\xrightarrow{\alpha}\}_{\alpha \in A_\tau}, q_{init})$ be a LTS.

successors.

$$split(X,\rho) \;=\; \bigsqcap_{Y\in\rho}\ \bigsqcap_{a\in A_\tau} \{X \cap pre_a(Y), X \setminus pre_a(Y)\}$$

Stability X is said to be stable with respect to ρ if and only if $\{X\} = split(X,\rho)$. ρ is stable if and only if it is stable with respect to itself.

Let $Stable(\rho) = \{X \in \rho \mid split(X,\rho) = \{X\}\}$.

Accessibility Let ρ be a partition of Q. We define the function Acc_ρ:

$$Acc_\rho(X) = [q_{\text{init}}] \cup \bigcup_{\alpha\in A_\tau} post_{\alpha,\rho}(X).$$

Given a partition ρ, the set of accessible states is the least fixed-point of Acc_ρ in the lattice 2^{2^Q}. However, the fact that a class belong to Acc_ρ does not imply that it contains an accessible state. This property becomes true when all the accessible classes are stable.

In [BFH90b], we propose an algorithm which compute the greatest fixed-point

$$\nu\rho\ .\ \rho_{\text{init}} \sqcap Ref(\mu\pi\ .\ Acc_\rho(\pi \cap Stable(\rho)), \rho)$$

by taking into account the stability on accessible classes. A step of the algorithm consists of choosing and refining a class, accessible from the stable classes.

4.2.1 Discussion

We can easily extend this algorithm to w-bisimulation and branching bisimulation, by modifying the definition of the function *split*:

w bisimulation

$$split(X,\rho) \;=\; \bigsqcap_{Y\in\rho}\ \bigsqcap_{a\in A_\tau} \{X \cap pre_{\tau^*a}(Y), X \setminus pre_{\tau^*a}(Y)\}.$$

Branching bisimulation

$$\begin{aligned} split(X,\rho) \;=\; & \bigsqcap_{\substack{Y\in\rho \\ X\neq Y}}\ \bigsqcap_{a\in A_\tau} \{\mathcal{F}_a(X,Y), X \setminus \mathcal{F}_a(X,Y)\} \\ & \sqcap \bigsqcap_{a\in A} \{\mathcal{F}_a(X,X), X \setminus \mathcal{F}_a(X,X)\} \end{aligned}$$

5 Experiments using the tool ALDÉBARAN

We summarize some of the results obtained when using the verification methods presented in this paper within the tool ALDÉBARAN. In particular, we compare the two distinct approaches (i.e, classical versus "on the fly") as well as the practical behavior of the algorithms associated to different relations. We first briefly present the tool ALDÉBARAN and give its current state.

ALDÉBARAN is a tool performing the minimization and comparison of labeled transition system with respect to several simulation and bisimulation-based equivalence relations. It is either used as a verification tool (in association with a LTS generator) or "internally" (for example inside another tool, as a graph minimizer). The two approaches mentioned above have been implemented within ALDÉBARAN, with respect to various relations:

classical approach: minimization and comparison algorithms have been implemented for strong bisimulation, observational equivalence, w bisimulation, safety equivalence and acceptance model equivalence, which is a variant of readiness semantics.

"on the fly approach": comparison algorithms have been implemented for strong and safety preorder, strong bisimulation, w-bisimulation, safety equivalence, delay bisimulation, and branching bisimulation when one of the LTS is τ-free (i.e, without τ actions). Theoretically branching bisimulation could also have been implemented in the general case but to obtain an efficient algorithm it was better considering this restricted case. In fact, it does not seem unrealistic in practice to consider that a specification is τ-free, and it has been shown that in this case branching bisimulation and observation equivalence coincides.

Remark

- The comparison algorithm which is currently implemented does not process yet the LOTOS program description "on the fly": as in the classical method, the two LTS are previously generated and the comparison phase consists in simultaneously building the LTS product and checking for the existence of an explanation sequence as described in section 4.
- The minimal model generation algorithm is still under implementation.

□

In addition, in both approaches *diagnostic features* are computed by ALDÉBARAN when the two LTS under comparison are not related: in the classical approach the set of all the explanation sequences is given, whereas in the "on the fly" approach one explanation sequence is exhibited.

5.2 Experiments

Two examples are discussed here: the first one is the well known scheduler described by Milner in [Mil80], and the second one is an alternating bit protocol called Datalink protocol [QPF88]. For each example, we proceed as follows:

- generating the labeled transition system S_1 from a LOTOS description, using CÆSAR.
- building the labeled transition system S_2, representing the expected behavior of the system.
- comparing S_1 and S_2 with respect to w-bisimulation, branching bisimulation and safety preorder using the "on the fly" algorithm.

classical algorithm.

The times have been obtained on a SUN 4 SparcStation using the *times*() UNIX standard function. Only the verification phase is taken into account. In each table, the first value represents the *system* time, whereas the second one represents the *user* time.

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 which can be found in [Mil80]. The results are given for different values of N.

The first table contains the sizes of the LTS obtained from the LOTOS program:

N	number of states	number of transitions
8	3073	13825
9	6913	34561
10	15361	84481
11	33793	202753
12	73729	479233

Sizes of the LTS obtained from the LOTOS programs

Using the classical minimization algorithm, due to memory shortage the LTS cannot be processed when $N > 8$ for observation equivalence, and when $N > 11$ for w bisimulation. For smaller LTS, the system and user times obtained are the following:

N	Paige Tarjan		Transitive Closure	
8	0.017	4.417	1.533	134.200
9	0.250	15.333	8.417	918.917

Minimization with respect to Observation Equivalence

N	Paige Tarjan		Transitive Closure	
8	0.000	0.533	0.067	4.367
9	0.000	1.650	0.250	20.350
10	0.050	5.333	0.617	111.517
11	0.433	7.600	2.633	581.017

Minimization with respect to w Bisimulation

Using the "on the fly" algorithm, the comparison can be carried out up to 12 cyclers:

N	Branching Bisimulation		w Bisimulation		Safety Preorder	
8	0.150	0.950	0.050	1.650	0.133	0.483
9	0.300	2.850	0.217	5.900	0.317	1.233
10	1.033	7.300	0.633	21.800	1.217	2.933
11	2.783	15.333	1.767	83.950	2.117	6.883
12	7.283	39.300	6.750	341.283	9.150	16.383

Comparison using the "on the fly" approach

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), labeled transition systems of different sizes can be obtained. The sizes of the LTS are the following:

N	number of states	number of transitions
40	27281	40320
50	42101	62400
70	81341	120960
80	105761	157440
90	133380	198719
100	167459	249672

Sizes of the LTS obtained from the LOTOS programs

Again, for memory shortage reasons, the LTS cannot be minimized when $N > 50$ with respect to observation equivalence and w bisimulation using the classical approach. For smaller LTS, the times obtained are the following:

N	Paige	Tarjan	Transitive	Closure
40	0.150	7.417	0.080	3.317
50	2.133	13.650	1.000	2.650

Minimization with respect to Observation Equivalence

N	Paige	Tarjan	Transitive	Closure
40	0.033	4.050	0.017	1.950
50	0.117	7.667	0.033	2.983

Minimization with respect to w Bisimulation

Using the "on the fly" approach, the comparison can be carried out up to 100 messages and more:

N	Branching	Bisimulation	w	Bisimulation	Safety	Preorder
40	0.717	3.350	0.117	0.250	0.600	2.000
50	1.017	5.250	0.200	2.033	1.000	3.083
70	2.233	9.950	0.483	3.850	1.817	6.050
80	4.033	5.600	0.450	6.117	2.750	9.583
90	4.933	6.600	0.633	6.700	3.433	9.817
100	5.783	28.250	0.850	8.500	1.217	2.933

Comparison using the "on the fly" approach

6 Conclusion

In this paper, we have presented an overview of the methods implemented in ALDÉBARAN. ALDÉBARAN is a part of a tool set including CÆSAR and CLÉOPÂTRE [Ras91]. CÆSAR is a

for branching-time logic specifications, including a verification module and a diagnostic module.

Initially, we have dealt with classical method in ALDÉBARAN: computation of a normal form for LTS with respect to a given equivalence, followed by a minimization or a comparison up to strong bisimulation. The limitation of this method is now well known.

From this limitation, we experiment now algorithms presented in 4. However, we do not yet combine generation algorithm and verification algorithm in the current implementation, since the method is applied to LTS already constructed. We want to combine the generation phase, (i.e., from Petri Nets to LTS), of the LOTOS compiler CÆSAR with verification algorithms of ALDÉBARAN.

For this purpose, a depth first search generation algorithm can be combined with the algorithm constructing the partial product. Thus, larger LOTOS programs could be carried out. In particular, there is no restriction on the data types.

Minimal LTS generation can also be implemented. This method may require symbolic computations in order to determine the *pre* and *post* functions, inclusion and intersection of classes. Such symbolic computations are reasonably achievable in the boolean case, (i.e., LOTOS description with boolean value). We think that this method is also suitable for other simple data types.

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.

[BFH90a] A. Bouajjani, J. C. Fernandez, and N. Halbwachs. *On the verification of safety properties*. Tech. report, Spectre L 12, IMAG, Grenoble, march 1990.

[BFH90b] A. Bouajjani, J.C. Fernandez, and N. Halbwachs. Minimal model generation. In *Workshop on Computer-aided Verification*, to appear in LNCS, Springer Verlag, june 1990.

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

[CES86] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic spec ifications. *TOPLAS*, 8(2), 1986.

[Cle90] R. Cleaveland. On automatically distinguishing inequivalent processes. In *Workshop on Computer-Aided Verification*, june 1990.

[CPS90] R. Cleaveland, J. Parrow, and B. Steffen. The concurrency workbench. 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.

INRIA, Sophia Antipolis, 1989.

[Fer90] J. C. Fernandez. An implementation of an efficient algorithm for bisimulation equivalence. *Science of Computer Programming*, 13(2-3), May 1990.

[FM91] J.-C. Fernandez and L. Mounier. "on the fly" verification of behavioural equivalences and preorders. In *Workshop on Computer-aided Verification*, To appear, july 1–4 1991.

[Gla90] R.J. van Glabbeek. *The Linear Time - Branching Time Spectrum*. Technical Report CS-R9029, Centre for Mathematics and Computer Science, 1990.

[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.

[GV90] Jan Friso Groote and Frits Vaandrager. *An Efficient Algorithm for Branching Bisimulation and Stuttering Equivalence*. CS-R 9001, Centrum voor Wiskunde en Informatica, Amsterdam, January 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.

[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.

[JJ91] Claude Jard and Thierry Jéron. Bounded-memory algorithms for verification on-the-fly. In K. G. Larsen, editor, *Proceedings of the 3rd Workshop on Computer-Aided Verification (Aalborg, Denmark)*, July 1991. to appear.

[KS83] P. Kanellakis and S. Smolka. Ccs expressions, finite state processes and three problems of equivalence. In *Proceedings ACM Symp. on Principles of Distribued Computing*, 1983.

[Mil80] R. Milner. A calculus of communication systems. In *LNCS 92*, Springer Verlag, 1980.

[NMV90] R. De Nicola, U. Montanari, and F.W. Vaandrager. *Back and Forth Bisimulations*. CS-R 9021, Centrum voor Wiskunde en Informatica, Amsterdam, 1990.

[NV90] R. De Nicola and F. Vaandrager. Three logics for branching bisimulation. In *Proc. of Fifth Symp. on Logic in Computer Science*, Computer Society Press, 1990.

Conference on Theorical Computer Science, Springer Verlag, 1981. LNCS 104.

[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.

[Ras91] A. Rasse. Error diagnosis in finite communicating systems. In *Workshop on Computer-aided Verification*, To appear, july 1–4 1991.

[RRSV87] J.L. Richier, C. Rodriguez, J. Sifakis, and J. Voiron. *Xesar: A Tool for Protocol Validation. User's Guide*. LGI-Imag, 1987.