Symbolic Equivalence Checking *

J.C. Fernandez, A. Kerbrat and L. Mounier

VERIMAG, B.P. 53 X, 38041 - Grenoble Cedex, France

Abstract. We describe the implementation, within ALDEBARAN of an algorithmic method allowing the generation of a minimal labeled transition system from an abstract model ; this minimality is relative to an equivalence relation. The method relies on a symbolic representation of the state space. We compute the minimal labeled transition system using the Binary Decision Diagram structures to represent the set of equivalence classes. Some experiments are presented, using a model obtained from LOTOS specifications.

1 Introduction

Program analysis is a part of process design whose purpose is to verify statically dynamic properties of the run time behaviour of a program. In this general framework, we are interested in the verification of behavioural properties on a concurrent program specified in the LOTOS language [ISO87]. For this purpose, a possible approach is to translate the program and the properties to be verified into suitable abstract models, and to check the equivalence of these models under some abstraction criteria. The abstract models considered in this work are labeled transition systems, which provide behavioural descriptions of the programs and of the properties. The equivalence relations chosen to compare these models are bisimulation relations, defined by [Mil80], which are now widely used in the context of concurrent program verification.
A bisimulation relation on a state space may be viewed either as a partition (the set of its classes) or as a binary relation. According to the choice of the definition, two algorithmic families can be considered to perform the equivalence checking. One of them [PT87,KS90,BFH*92,LY92] is based on refinement principle: *given an initial partition, find the coarsest partition stable with respect to the transition relation.* The other is based on a cartesian product traversal from the initial state [FM91a, GLZ89]. These algorithms are both applied on the whole state graph, and they require an explicit enumeration of this state space. This approach leads to the well-known *state explosion problem.* A possible solution is to reduce the state graph before performing the check. Classical reduction algorithms already exist [Fer90, KS90,PT87], but they can be applied only when the whole state space has been computed, which limits their interest. In [BFH90], another reduction algorithm is presented, namely the *Minimal Model Generation Algorithm* (MMG, for short). This algorithm allows the minimization of the graph *during* its generation, thus avoiding in part the state explosion problem. The main goal of this paper is to present some implementation issues of this algorithm within the verification toolbox

* This work was partially supported by ESPRIT Basic Research Action "SPEC"

Cæsar-Aldébaran [FGM*92]. Several bisimulation relations are considered, such as strong bisimulation, weak bisimulation and branching bisimulation.
The paper is organized as follows: first, we recall the definitions and notations used throughout the paper, together with the MMG Algorithm. Then we present the model used for our experiments, i.e., a net of transition systems represented by means of Binary Decision Diagrams, BDDs for short. Finally, we give some results obtained from the implementation of the algorithm within *Aldébaran*.

2 Definitions and Notations

We recall here the main definitions related to Labeled Transition Systems (LTS, for short) and bisimulation relations, together with the associated notations.
Throughout the paper, we consider a LTS $S = (Q, A_\tau, \{\xrightarrow{a}\}_{a \in A_\tau}, q_{init})$ where: Q is a set of program *states*, $A_\tau = A \cup \{\tau\}$ is a set of *action names*, where τ is a distinguished name representing an *internal* action, $\rightarrow \subseteq Q \times A_\tau \times Q$ is the *transition relation* of the program, and q_{init} is the *initial state* of the program, i.e., a distinguished element of Q.
Associated with the LTS S, we introduce the following notations:
$\mathcal{P}$ represents the lattice of partitions of Q:

- it is ordered by the refinement relation $\sqsubseteq$: $\rho \sqsubseteq \rho'$ iff $\forall X \in \rho, \exists X' \in \rho' \,.\, X \subseteq X'$
- its greatest lower bound operator is $\sqcap$: $\bigsqcap_i \rho_i = \{T \neq \emptyset \mid T = \bigcap_i X_i \text{ and } X_i \in \rho_i\}$

The pre- and post-condition functions from 2^Q to 2^Q, for a given $a \in A_\tau$, are defined as usual:

$$pre^a(X) = \{q \in Q \mid \exists q' \in X \text{ such that } q \xrightarrow{a} q'\}$$
$$post^a(X) = \{q \in Q \mid \exists q' \in X \text{ such that } q' \xrightarrow{a} q\}$$

We denote by $[q]_\rho$ the class of the partition ρ containing the state q. Let $pre^a_\rho, post^a_\rho$ the pre- and post-condition functions corresponding to a partition ρ. These functions are overloaded as follows:

from 2^Q to $\mathcal{P}$: $pre^a_\rho(X) = \{[q]_\rho \mid q \in pre^a(X)\}$, $post^a_\rho(X) = \{[q]_\rho \mid q \in post^a(X)\}$
from $\mathcal{P}$ to $\mathcal{P}$: $pre^a_\rho(\rho') = \bigcup\{pre^a_\rho(X) \mid X \in \rho'\}$, $post^a_\rho(\rho') = \bigcup\{post^a_\rho(X) \mid X \in \rho'\}$

Intuitively, bisimulations relations are intended to compare LTS from a behavioural point of view: two LTS are bisimilar if and only if they represent the same behaviour, observed from a given abstract level.
Let Λ be a set of disjoint languages on A and $\lambda \in \Lambda$.
We write $p \xrightarrow{\lambda} q$ if and only if:
$\exists u_1 \cdots u_n \in \lambda \ \wedge \ \exists q_1, \cdots, q_{n-1} \in Q \ \wedge \ p \xrightarrow{u_1} q_1 \xrightarrow{u_2} q_2 \cdots q_i \xrightarrow{u_{i+1}} q_{i+1} \cdots q_{n-1} \xrightarrow{u_n} q$.
A relation $R \subseteq Q \times Q$ is a *bisimulation* iff it is symmetric and satisfies the following property:

$$(p_1, p_2) \in R \ \wedge \ p_1 \xrightarrow{\lambda} q_1 \ \Rightarrow \ \exists q_2 \,.\, (p_2 \xrightarrow{\lambda} q_2 \ \wedge \ (q_1, q_2) \in R)$$

From this general definition, one can obtain several relations by varying the value of the Λ parameter. We will consider in this work the *strong bisimulation* relation, noted $\sim$, obtained for $\Lambda = \{\{a\} \mid a \in \mathcal{A}_\tau\}$, and the *weak bisimulation* relation, noted $\sim_w$, obtained for $\Lambda = \{\tau^* a \mid a \in \mathcal{A}\}$. Note that this last relation differs from Milner's "observational equivalence" ([Mil80]).
We will also consider a third relation, called *branching bisimulation* ([GW89]), noted $\sim_b$, which can be viewed as another alternative to observational equivalence:
A relation $R \subseteq Q \times Q$ is a *branching bisimulation* if it is symmetric and satisfies the following property:
$(p_1, p_2) \in R \ \wedge \ p_1 \xrightarrow{\alpha} q_1 \ \Rightarrow \ (\alpha = \tau \ \wedge \ (q_1, p_2) \in R) \ \vee$
$(\exists q_2 q_2' \, . \, (p_2 \xrightarrow{\tau^*} q_2' \ \wedge \ q_2' \xrightarrow{\alpha} q_2 \ \wedge \ (p_1, q_2') \in R \ \wedge \ (q_1, q_2) \in R))$
Let ρ be the partition associated to a bisimulation relation. The quotient of the LTS S by ρ is the LTS:

$$S/_R = (\{[p]_\rho \mid p \in Q\}, A, \{([p]_\rho, a, [p']_\rho) \mid (p, a, p') \in T\}, [q_0]_\rho).$$

3 The MMG Algorithm

In this section we recall the algorithm proposed in [BFH90,BFH*92], which allows to compute the quotient of a LTS S with respect to strong bisimulation. First, we give its principle, introducing the main notations. Then, we propose a set of data structures, leading to an efficient implementation, and we show how the original algorithm can be extended to deal with weak bisimulation and branching bisimulation.

3.1 The principle of the algorithm

Given a LTS S, the principle of the MMG algorithm is to refine an initial partition ρ_{init} of the state space of S until a *reachable* and *stable* partition is obtained. More precisely, it can be defined as the computation of the greatest fixpoint of a *split function* on the partition ρ_{init}, distinguishing two subsets of classes at each step of this computation :

- The set π of *reachable classes*, i.e., the classes containing at least one element which has been found reachable so far from q_{init}.
- The set σ of *stable classes*, i.e., the reachable classes which have been found stable with respect to the current partition (assuming a class X is stable with respect to a partition ρ if and only if $\{X\} = split(X, \rho)$).

A step of the algorithm consists in scanning each reachable class of the current partition, checking whether this class is stable or not with respect to this partition. Whenever a reachable class is found unstable, it is split into stable subclasses, and its predecessors are removed from σ, since their stability is questioned (see below the definition of the *split* function). Only subclasses which are obviously reachable are put in π: those which either contain q_{init}, or are directly reachable from a reachable stable class.
It can be shown that, for a suitable *split* function, the resulting partition exactly coincides with the set of equivalence classes of the coarsest (strong) bisimulation on

S containing ρ_{init} ([BFH90]). Thus, when choosing for ρ_{init} the universal partition, this algorithm computes the states of the quotient of S with respect to the strong bisimulation relation $\sim$.
In the rest of the section, we give a more formal description of this algorithm. First, we precise the definitions of the *split* function and of the stability and reachability properties [FM91b]:
Let ρ be a partition of the states of S and $X, Y \in \rho$.

split function : $split(X, \rho) = \bigsqcap_{Y \in \rho} \bigsqcap_{a \in A_\tau} \{X \cap pre^a(Y), X \setminus pre^a(Y)\}$

Stability : $Stable(\rho) = \{X \in \rho \mid split(X, \rho) = \{X\}\}$

$Stable(\rho)$ denotes the set of the stable classes of partition ρ. Note that if a class $X \subseteq pre^a(Y)$ is stable for all classes $Y \in post^a_\rho(X)$ then X is stable. We also extend this notion of stability to transitions between classes : the transition $X \xrightarrow{a} Y$ is said *stable* with respect to ρ if and only if $X \subseteq pre^a(Y)$.

Reachability : $Acc_\rho(X) = [q_{init}] \cup \bigcup_{a \in A_\tau} post^a_\rho(X)$.

Given a partition ρ, the set of reachable classes is the least fixed-point of Acc_ρ in the lattice 2^{2^Q}. However, note that a class belonging to this set can contain unreachable states.

The MMG algorithm computes the greatest fixpoint

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

which can be written in a more algorithmic fashion :

```
begin
    ρ = ρ_init; π = {[init]_ρ}; σ = ∅ ;
    while π ≠ σ do
        choose X in π \ σ ;
        let π' = split(X, ρ);
        if π' = {X} then
            σ := σ ∪ {X}
            π := π ∪ post_ρ(X) ;
        else π := π \ {X} ;
            if ∃Y ∈ π' such that init ∈ Y then
                π := π ∪ {Y} ;
            fi
            σ := σ \ pre_ρ(X);
            ρ := (ρ \ {X}) ∪ π';
        fi
    od
end
```

Note that the statement $\sigma := \sigma \setminus pre_\rho(X)$ can be performed by scanning the stable classes and checking if all the transitions from X to a stable class are stable. This point is detailed in the next section, together with the computation of the *split* function.

3.2 Data structures and implementation issues

We define the LTS $T_r = (\rho_r, A, \{\xrightarrow{a}_r\}_{a \in A_\tau}, [q_{init}]_{\rho_r})$, associated with the current partition ρ_r. T_r represents the quotient of S by ρ_r. It is built as follows : Initially,

$T_{init} = (\rho_{init}, A, \{\xrightarrow{a}_{init}\}_{a\in A_\tau}, [q_{init}]_{\rho_{init}})$ where $\xrightarrow{a}_{init} = \{(X,Y) \mid X,Y \in \rho_{init}\}$. During the refinement process, when a class X is split into subclasses $X_1, .., X_n$, $\rho_{r+1} = \rho_r \setminus \{X\} \cup \{X_1, \cdots, X_n\}$ and the transitions (X,a,Y) (resp. (Z,A,X)) are removed from T_r and replaced by the transitions (X_i,a,Y) (resp (Z,a,X_i)). Note that the decomposition of X in subclasses can be retrieved at any time using a decomposition tree.
The implementation of the *split* function relies on the two following propositions

Proposition 3.1 *When splitting a class X, we only have to consider as splitters the classes Y such that (X,a,Y) belongs to $\{\xrightarrow{a}_r\}$, where $a \in A_\tau$.*

Proof. According to the definition of T_r, for all Y such that $X \cap pre^a(Y) \neq \emptyset$ we have $(X,a,Y) \in \{\xrightarrow{a}_r\}$. Consequently, if we consider a class Y in ρ_r such that $\forall a \in A_\tau$, $(X,a,Y) \notin \{\xrightarrow{a}_r\}$, we can deduce that $\forall a \in A_\tau$, $X \cap pre^a(Y) = \emptyset$. Then $split(X,Y) = \{X\}$ and it is not necessary to try to split X with respect to Y. ■

Proposition 3.2 *When splitting a class X, we only have to consider the unstable transitions $(X,a,Y) \in T_r$.*

Proof. Let $(X,a,Y) \in T_r$ be a stable transition. From the definition of the stability of a transition, $X \subseteq pre^a(Y)$, and then $split(X,Y) = \{X\}$. So it is not necessary to split X with respect to stable transitions. ■

Proposition 3.3 *When the algorithm terminates, T_r is equal to $S/_\sim$.*

It remains to show how this algorithm can be extended respectively to weak bisimulation and branching bisimulation. We recall for each of these relations the definition of the *split* function [FM91b], and we briefly discuss how this function can be implemented.

weak bisimulation :

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

The data structures and propositions given above can be straightly extended to this case by substituting in each definition pre^a by $pre^{\tau^* a}$.

branching bisimulation :

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

where : $\mathcal{F}_a(X,Y) = \mu Z.(X \cap pre^\tau(Z) \cup X \cap pre^a(Y))$ and $\mu Z.f(Z)$ denotes the least fixpoint of f.

Branching bisimulation is a particular case. The definition of $\mathcal{F}_a$ implies some differences in the previous definitions :

1. A transition (X, a, Y) is said *stable* iff $X \subseteq \mathcal{F}_a(X, Y)$.
2. Another difference is that a class X cannot be split with respect to itself if the label considered is τ. So, if there exists a transition (X, τ, X) in T_r, class X must not be split. Moreover, when X is split in $X_1, ..., X_n$, the new transitions (X_i, τ, X_j), $(i, j) \in [1..n] \times [1..n]$ must be inserted in T_r, because a class X_i may be splittable with respect to X_j if $i \neq j$. So we keep in T_r on each class a τ loop during the refinement. When the algorithm terminates, we remove all τ loops, without altering the equivalence.

4 An Implementation Using Binary Decision Diagrams

4.1 The model

Our model is similar in many respects with those of [EFT91] and [BdS92]. It consists of a set of communicating LTSs. $S_i = (Q_i, A_i, \{\xrightarrow{a}_i\}_{a \in A_\tau}, init_i), i = 1..N$ and a composition expression F which expresses the communications between the LTSs. The syntax of the language of composition expressions is extracted from the language LOTOS [ISO87], so all CCS-like programs with parallel composition and hiding operators only can easily be translated.

expression **::=** expression |[label-list]| expression | **hide** label-list **in** expression | **LTS**

label-list **::=** ϵ | **label**, label-list

We will call $S = F(S_1, S_2, ..., S_N)$ the LTS given by this set of communicating LTSs. More details on this model can be found in [Mou92].

Compositional minimization : The use of this model allows an a-priori reduction of the size of the LTSs to be minimized. When the equivalence considered is a congruence, which is the case of the bisimulations mentioned in section 2, it is possible to minimize the LTSs first wrt the congruence, then to apply the MMG algorithm on the composition for the same congruence. This strategy is especially interesting with weaker bisimulations, where some examples needing several hours of computations with the full model can be minimized in a few minutes when the LTSs have been reduced beforehand.

Synchronization set : We define for each action $a \in A_1 \cup A_2 \cup \ldots \cup A_N$ a synchronization set $Synchro(a)$ which will contain the lists of LTSs for which a is a synchronous action and a set Asynchro(a) which will contain the LTS for which a is asynchronous. This set is constructed by analysis of the composition expression F. Each element of $Synchro(a)$ corresponds in fact to a *Labeled Synchronization Vector* as introduced by [AN82] and used in [BdS92] .

Abstraction set : During the analysis of the composition expression F, we collect all the actions used by the operator **hide** in a so-called abstraction set.
Let $L_i, i = [1..n]$ be the sets of actions used by the operator **hide**. We define the abstraction set as follows :

Definition 4.1 $Hide = \bigcup_i Li$

The definition of the set $Hide$ is a restriction of the semantics of the LOTOS **hide** operator, since the hiding of an action in a sub-expression will be considered globally in $Hide$:
Given the expression $S_1|[]|$ (**hide** a **in** $(S_2|[a]|S_3)$), with the current definition of Hide, all occurrences of a in S_1 will be renamed by τ. But this restriction can easily be bypassed by relabeling the action a in the sub expression **hide** a **in** $(S_2|[a]|S_3)$.

4.2 Binary Decision Diagrams

To implement this algorithm, we need to represent state classes, and to perform operations on these classes, like intersection, union and complementation. We also need to represent the function pre^a and to compute $pre^a(X)$ for any state class X. For this purpose, we choose Binary Decision Diagrams.
A Binary Decision Diagram (BDD) [Bry86] is an efficient way to represent and manipulate boolean functions. They are constructed as a decision tree, and it has been shown in [Bry86] that a normal form can then be computed by sharing subtrees, this normal form depending of the ordering of the boolean variables involved. BDDs have already been successfully applied to implement other algorithms related to equivalence checking [BCM*89,EFT91,BdS92].

4.3 Representation of a LTS with BDDs

Given a LTS $S = (Q, A, \{\xrightarrow{a}\}_{a\in A_\tau}, init)$, its representation with BDDs will be given by : $S_{(\mathbf{x},\mathbf{y})} = (A, \{\xrightarrow{a}_{(\mathbf{x},\mathbf{y})}\}_{a\in A_\tau}, init_{\mathbf{x}})$ that is, a set of BDDs representing the transition relation and one BDD for the initial state. $\mathbf{x}$ and $\mathbf{y}$ are the sets of boolean variables needed for the encoding. These sets are called *support sets* in the following. As usual, we will identify a set X with its characteristic function f_X, represented by a BDD. Especially, a LTS $S = (Q, A, \{\xrightarrow{a}\}_{a\in A_\tau}, init)$ will be represented by $S_{(\mathbf{x},\mathbf{y})} = (A, \{\xrightarrow{a}\}_{a\in A_\tau}, init_{\mathbf{x}})$, where each element of the tuple $S_{(\mathbf{x},\mathbf{y})}$ is the characteristic function of the corresponding set in S.

4.4 Representation of the global LTS

Given $S_{i_{(\mathbf{x}_i,\mathbf{y}_i)}} = (A, \{\xrightarrow{a}_i\}_{a\in A_\tau}, init_i)_{0\le i\le n}$ the representation of LTSs S_i with BDDs, and the sets Hide, Synchro and Asynchro associated to a given composition expression F, the representation of the global system

$$S = (Q, A, \{\xrightarrow{a}\}_{a\in A_\tau}, init) = F(S_1, S_2, ..., S_N)$$

is

$$S_{(\mathbf{X},\mathbf{Y})} = (A, \{\xrightarrow{a}\}_{a\in A_\tau}, init)$$

with

$$\mathbf{X} = \bigcup_{i=1..N} \mathbf{x}_i \qquad \mathbf{Y} = \bigcup_{i=1..N} \mathbf{y}_i$$

$$\xrightarrow{a} = (a \notin Hide) \wedge (F_{Synchro(a)} \vee F_{Asynchro(a)})$$

$$\xrightarrow{\tau} = (\bigvee_{a \in Hide} (F_{Synchro(a)} \vee F_{Asynchro(a)}) \vee \bigvee_{i \neq j} ((Stable_i \wedge \xrightarrow{\tau}_j) \vee (\xrightarrow{\tau}_i \wedge Stable_j)))$$

$$F_{Synchro(a)} = \bigvee_{B_i \in Synchro(a)} \bigwedge_{j \in B_i} \xrightarrow{a}_j$$

$$F_{Asynchro(a)} = \bigvee_{i,j \in Asynchro(a), i \neq j} ((Stable_i \wedge \xrightarrow{a}_j) \vee (\xrightarrow{a}_i \wedge Stable_j))$$

$$Stable_i = \bigwedge_{x_j \in \mathbf{X}_i, y_j \in \mathbf{Y}_i} (x_j = y_j)$$

4.5 $pre^a(\boldsymbol{X})$ computation

In order to compute $pre^a(X)$ for $X \in Q$, we need two specialized operators on BDDs.

Definition 4.2 *Let f be a boolean function defined on the support set $\mathbf{y}$, and $\mathbf{x}$ a support set such that $\mathbf{x} \subset \mathbf{y}$. We will call Smooth the existential quantifier defined as :*

$$Smooth_{\mathbf{x}} = Smooth_{x_1} o Smooth_{x_2} o .. o Smooth_{x_n}(f)$$

$$Smooth_{x_i} = f_{x_i} \vee f_{\overline{x_i}}$$

where $f_{x_i} = f(x_1, x_2, .., x_{i-1}, 1, x_{i+1}, .., x_n)$ and $f_{\overline{x_i}} = f(x_1, x_2, .., x_{i-1}, 0, x_{i+1}, .., x_n)$

In practice, the Smooth operator defines a projection of f on the support set $\mathbf{y} \setminus \mathbf{x}$.

Definition 4.3 *Let f be a boolean function defined on the support set $\mathbf{x} = (x_1, .., x_n)$. Let $\mathbf{y} = (y_1, y_2, .., y_n)$ a support set such that $\mathbf{x}$ and $\mathbf{y}$ are disjoints. We will call Shift the substitution operation defined as :*

$$Shift_{\mathbf{x} \to \mathbf{y}}(f) = Smooth_{\mathbf{x}}(\bigwedge_{i=1..n} (x_i \Leftrightarrow y_i) \wedge f)$$

If both support sets have the same relative order, that is $\forall (x_i, x_j) \in \mathbf{x}, x_i < x_j \Rightarrow y_i < y_j$, then this operator will not change the structure of the BDD, thus the resulting BDD will have the same size. Moreover, in that case the implementation of the shift operator consists of one linear traversal of the BDD with variables relabeling. Given these two operators, we can now compute $pre^a(X)$. In the MMG Algorithm, we have to compose the computations of *pre* with the results of previous computations of *pre*. So if the characteristic function of $X \in Q$ is defined on the support set $\mathbf{x}$, we would like the result of $pre^a(X)$ to be defined on the same support set. That gives the following formula for the computation of $pre^a(X)$:

$$pre^a(X) = Smooth_{\mathbf{y}}(\xrightarrow{a}_{(\mathbf{x},\mathbf{y})} \wedge \mathrm{Shift}_{\mathbf{x} \to \mathbf{y}}(X_{\mathbf{x}}))$$

4.6 Ordering of support sets

The ordering of variables in a support set determines the normal form of a BDD. So with two different orderings, the same formula will be represented by two different BDDs. It is important to find a good ordering , as the size of a BDD can be linear to exponential wrt the number of variables used for the encoding
Two levels of ordering are considered: the local ordering, which is internal to a communicating LTS and the global ordering, that describes how we mix local support sets. It is shown in [EFT91] that the interleaved order is better for the construction of *Stable*, giving a BDD with 3*n nodes while the concatenated order gives $O(2^n)$ nodes for this BDD. Since we need to construct *Stable* during the composition, we have chosen the interleaved order. Having fixed the local order, we still have to choose a global ordering. Again, we considered two different orders, as found in [BdS92] : only the concatenated global order has been implemented, with good results. It is argued in [BdS92] that this order is better when the number of classes of the minimal model is small when compared with the size of the complete LTS.

4.7 Discussion

One drawback in this approach is that the algorithm starts from the universal partition which is much bigger than the set of reachable states, so we end up with a lot of computations done just to get rid of these unreachable states. This is partly the case in the MMG Algorithm, where we consider only the reachable classes for splitting. But a reachable class can contain only a few reachable subclasses and a lot of unreachable ones, which have to be removed layer by layer before getting the "right" subclasses. Worse, before an unreachable class is recognized as such, some unneeded computations can be done using this class.
In our model, i.e., communicating LTSs, as long as the LTSs themselves are finite, the reachable states space of the composition is also finite. In that case, our experiments have shown that restricting the initial partition to the reachable states space greatly improve the efficiency of the MMG Algorithm. In practice, such a reachable states space computation can be done efficiently using BDDs.

5 Results

The implementation has been tested on several examples. Three of them are presented here. The first one is the usual benchmark example, namely the Milner's Scheduler, which has the advantage to be made easily bigger by adding new cyclers. The second example is a Reliable Multicast Protocol [SE90], specified in LOTOS within the Hewlett-Packard Laboratories [BM90]. This protocol provides a *multicast* service, with one sender and several receiver. This protocol has been implemented in two versions :

- A "working" version where no process can crash.
- A "crashing" version where the processes have the possibility to crash. The corresponding model happens to be large, with 126 223 states and 428 766 transitions when generated with *Cæsar*. It is reduced to a model with 2995 states and 9228 transitions by minimizing first each LTS of the composition expression.

The last example is a LOTOS specification of the "Transit Node" case-study defined in the RACE project SPECS (Specification Environment for Communicating Software). It describes a routing component in telecommunications network.
For the performances given below, we used a SUN SPARC IPX workstation with 16 megabytes of memory. For these examples, the LTSs and the composition expression have been generated from the corresponding LOTOS program using a parser provided with the *Cæsar-Aldébaran* package.

- Name : Name of the example with :
  - ScXX : Milner's Scheduler with XX cyclers.
  - RelCm : 'Crashing' version of the Reliable Multicast Protocol with minimized LTSs.
  - Transit : Transit Node.
- N : number of states of the non minimized model, if it was generated.
- M : number of transitions of the non minimized model.
  for each bisimulation, we give :
  - n? : number of states of the minimized model.
  - m? : number of transitions of the minimized model.
  - t? : time (in s) for the minimization with reachable states computation.
  - t'? : time (in s) for the minimization without reachable states computation.

  These times are those given by the system, after the execution of *Aldébaran*. They include the LTSs loading time, LTSs composition, reachable states computation (if it is the case) and minimization. A "-" in a cell indicates that the corresponding minimization is not finished after one hour of computation or has been aborted due to lack of memory.

	Full Model		$\sim$				$\sim_w$				$\sim_b$			
Name	N	M	n1	m1	t1	t1'	n2	m2	t2	t2'	n3	m3	t3	t3'
Sc8	3073	13825	N-1	M-1	155	-	8	8	2	280	8	8	2	47
Sc10	15361	84481	N-1	M-1	-	-	10	10	3	1400	10	10	3	170
Sc20	$3.14\ 10^7$	$3.3\ 10^8$	N-1	M-1	-	-	20	16	23	-	16	20	23	-
Sc40	$6.59\ 10^{13}$	$1.35\ 10^{15}$	N-1	M-1	-	-	40	40	102	-	40	40	101	-
Sc80	$1.4\ 10^{26}$	$5.8\ 10^{27}$	N-1	M-1	-	-	80	80	1650	-	80	80	1650	-
RelCm	2995	9228	910	3847	480	-	43	156	125	-	95	358	55	-
Transit	93384	579892	-	-	-	-	-	-	-	-	18	43	360	-

These results show that this implementation is well suited for weaker bisimulations, especially branching bisimulation. It is hardly surprising, since the complexity of the algorithm depends greatly on the size of the minimal model, especially when we compute the reachable state space first. The size of the LTSs have a great influence on the size of the BDDs, and thus on the performances themselves. It is better to have a communicating LTSs system with a lot of small LTSs than a system with a few big ones.
Another problem is the generation of LTS without environment constraints : two LTSs synchronized together can be represented by a small global model. But taken without the constraints due to the synchronization with the other, each automaton can have a big local model, sometimes bigger than the global model.

6 Conclusion

In this paper we have studied one application of the MMG algorithm, which allows the minimization of a LTS during its generation. This algorithm has been adapted for various equivalences (strong, weak or branching bisimulation) and implemented in *Aldébaran* with interesting results. The main problems encountered in our experiments arise from the model itself, a system of communicating automata, where some base automata can be bigger than the full model itself. This disproportion comes mainly from the removal of some constraints due to synchronization, leading to the enumeration of some variables on their domain.
In the same context, we have studied another kind of model where a state is a couple (*control state, memory state)*. Dealing with such model, which can be infinite, requires the use of technics such as *abstract interpretation* proposed by P. & R.Cousot [CC77] and applied in [CH77]. This method allows to approximate a set of states with numerical variables by a polyhedron. All operations needed for the MMG algorithm are defined on polyhedra and computation of an approximation of the reachable states space is possible even in an infinite model, thanks to special operators defined in [CH77].
We are currently experimenting with this kind of model, with interesting results especially for the reachable states space computation. We are now working on the extension of this technic on petri net with values, as used in *Cæsar* [GS90] as intermediate form. Such extension would allow us to do static analysis on LOTOS programs.

Acknowledgements

We would like to thank Nicolas Halbwachs for his fruitful comments on this paper. Thanks are also due to Christophe Ratel for his efficient BDD package used within our implementation.

References

[AN82] A. Arnold and M. Nivat. Comportement de processus. *Les mathématiques de l'informatique*, 1982.

[BCM*89] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and J. Hwang. *Symbolic Model Checking:* 10^{20} *states and beyond.* Technical Report, Carnegie Mellon University, 1989.

[BdS92] A. Bouali and R. de Simone. Symbolic bisimulation minimisation. In *Fourth Workshop on Computer-Aided Verification,Montreal*, june 1992.

[BFH90] A. Bouajjani, J.C. Fernandez, and N. Halbwachs. Minimal model generation. In *Workshop on Computer-aided Verification, Rutgers*, American Mathematical Society, Association for Computing Machinery, june 1990.

[BFH*92] A. Bouajjani, J.C. Fernandez, N. Halbwachs, C. Ratel, and P. Raymond. Minimal state graph generation. *Science of Computer Programming*, 18(3), June 1992.

[BM90] S. Bainbridge and L. Mounier. *Specification and Verification of a Reliable Multicast Protocol.* Software Engineering Department Technical Report HPL-91-63, Hewlett-Packard Laboratories, Bristol, U.K, 1990.

[Bry86] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8), 1986.

[CC77] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *4th POPL*, january 1977.

[CH77] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *5th. Annual Symp. on Principles of Programming Languages*, pages 84–87, 1977.

[EFT91] R. Enders, T. Filkorn, and D. Taubner. Generating bdds for symbolic model checking in ccs. In K. G. Larsen, editor, *Proceedings of the 3rd Workshop on Computer -Aided Verification (Aalborg, Denmark)*, july 1–4 1991.

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

[FGM*92] J.Cl. Fernandez, H. Garavel, L. Mounier, A. Rasse, C. Rodriguez, and J. Sifakis. A tool box for the verification of lotos programs. In *14th International Conference on software Engineering*, 11-15May 1992.

[FM91a] J.-C. Fernandez and L. Mounier. "on the fly" verification of behavioural equivalences and preorders. In *Workshop on Computer-aided Verification, Aalborg University, Denmark*, LNCS 575, Springer Verlag, july 1–4 1991.

[FM91b] J.Cl. Fernandez and L. Mounier. A tool set for deciding behavioural equivalences. In J.F. Groote J.C.M. Baeten, editor, *CONCUR'91, Concurrency theory*, LNCS 527, Springer Verlag, august 26-29 1991.

[GLZ89] J.C. Godskesen, K. Larsen, and M. Zeeberg. Tav, tools for automatic verification. In *Proceedings of the 1st International Workshop on Automatic Verification Methods for Finite State Systems (Grenoble, France)*, Springer Verlag, jun 1989.

[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.

[ISO87] ISO. *LOTOS — A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour*. Draft International Standard 8807, International Organization for Standardization — Information Processing Systems — Open Systems Interconnection, Genève, July 1987.

[KS90] P. Kanellakis and S. Smolka. Ccs expressions, finite state processes and three problems of equivalence. *Information and Computation*, 86(1), May 1990.

[LY92] D. Lee and M. Yanakakis. Online minimization of transition systems. In *ACM STOC 92, Vancouver, B.C.*, 1992.

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

[Mou92] L. Mounier. *Méthodes de Vérification de Spécifications Comportementales : étude et mise en oeuvre*. PhD thesis, Université de Grenoble, 1992.

[PT87] R. Paige and R. Tarjan. Three partition refinement algorithms. *SIAM J. Comput., No. 6*, 16, 1987.

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