DIMACS Series in Discrete Mathematics
and Theoretical Computer Science
Volume 3, 1991

Minimal Model Generation *

A. Bouajjani , J-C. Fernandez , N. Halbwachs
IMAG/LGI (U.A. CNRS 398)
B.P. 53, 38041 - Grenoble, France

Abstract

This paper adresses the problem of generating a minimal state graph from a program, without building first the whole state graph. The minimality is considered here with respect to bisimulation. A generation algorithm is presented and illustrated.

1 Introduction

Model generation consists of building a state graph from a program, a formula or any comprehensive expression of a transition system. It is used in program verification ("model checking" [6,11]) and compiling (scanner and parser generation [1], control structure synthesis [2,5],...). A crucial problem with model generation is the size of the graph, which can be prohibitive. This size can be large not only because of the intrinsic complexity of the model, but also because the graph contains a lot of states which are in some sense equivalent. Some solutions have been given to this problem, by applying reduction algorithms [8,9,10]. However, these algorithms can only be applied once the graph has been entirely generated. It is often the case that a tremendous amount of time and memory is necessary to generate a graph, which afterward reduces to a very simple one. It even happens that an infinite model reduces to a finite one. So, it would be interesting to reduce the graph during the generation, on one hand to improve the performances of model generation, and on the other hand, to allow finite model generation from infinite systems. This paper presents and illustrates an algorithm performing this task, when the equivalence considered on states is a bisimulation.

After fixing some terminology and notations (section 2), the algorithm is presented (section 3) and illustrated on a simple example (section 4).

2 Definitions and notations

Let $\mathcal{S} = (Q, \rightarrow, q_{init})$ be a transition system, where Q is a set of states, $\rightarrow \subseteq Q \times Q$ is a transition relation, and q_{init} is the initial state. Let $\sim$ be an equivalence relation on Q. Our problem is to explicitely build the quotient of the set of reachable states from q_{init}, by the coarsest bisimulation compatible with $\sim$. Of course, this is only possible if this quotient has finitely many elements. Moreover, the method presented here only works if the quotient of Q by the coarsest bisimulation is finite (notice that Q itself can be infinite). The basic idea is to progressively build a partition of Q, by distinguishing two parts of Q only when their respective elements clearly don't bisimulate each other. Henceforth, we shall consider partitions instead of equivalence relations.

Let ρ be a partition of the set of states Q. The following notations will be used:

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

For any state $q \in Q$, $post_\rho(q)$ is the set of classes in ρ immediately reachable from q

$$post_\rho(q) = \{X \in \rho \mid \exists q' \in X \text{ such that } q \rightarrow q'\}$$

An equivalence relation, noted $\overset{\rho}{\sim}$, is associated with ρ as follows:

$$q_1 \overset{\rho}{\sim} q_2 \iff post_\rho(q_1) = post_\rho(q_2)$$

A subset X of Q is said to be *stable with respect to* ρ if and only if it is included in some equivalence class of $\overset{\rho}{\sim}$. The partition ρ is said to be stable if and only if all of its classes are stable with respect to itself. In other words, a partition is stable if and only if it is the set of classes of a bisimulation.

A *refinement* of a partition ρ is a partition ρ' such that: $\forall X \in \rho', \exists Y \in \rho$ such that $X \subseteq Y$

The *reduction* of a transition system $\mathcal{S}$ with respect to a stable partition ρ is the transition system $(\rho, \rightsquigarrow, [q_{init}]_\rho)$, where

$[q_{init}]_\rho$ is the class of the initial state in ρ

$X \rightsquigarrow Y \iff \exists q \in X, q' \in Y$ such that $q \rightarrow q'$

With the above terminology, given an initial partition ρ of Q, we are looking for the reduction of $\mathcal{S}$ with respect to the least stable refinement of ρ.

3 Algorithm

The algorithm consists of progressively refining the partition ρ. At each step, two subsets of classes will be distinguished:

- The set R of *reachable classes*, i.e. the classes containing at least one element which has been found reachable from q_{init}.
- The set S of *stable classes*, i.e. the reachable classes which have been found to belong to $\overset{\rho}{\sim}$.

The algorithm is the following:

```
R = {[q_init]_ρ}; S = φ;                                   (1)
while R ≠ S do                                             (2)
  choose X in R − S;                                       (3)
  let N = X/~^ρ;                                           (4)
  if N = {X} then                                          (5)
    S := S ∪ {X}; R := R ∪ {post_ρ(q) | q ∈ N};            (6)
  else                                                     (7)
    R := R − {X};                                          (8)
    if ∃Y ∈ N such that q_init ∈ Y then R := R ∪ {Y};       (9)
    S := S − {Y ∈ S | X ∈ post_ρ(Y)};                      (10)
    ρ := (ρ − {X}) ∪ N;                                    (11)
  fi                                                       (12)
od                                                         (13)
```

Proof : Let *Rea* be the set of reachable states, that is the least subset X of Q containing q_{init}, and such that

$$(q \in X \wedge q \rightarrow q') \Longrightarrow q' \in X$$

Then,

(*i*) $X \in S \Longrightarrow X \in \stackrel{\rho}{\sim}$
since a subset X is only put into S if $X = X/\stackrel{\rho}{\sim}$ (line 6), and as soon as a refinement of ρ can involve a refinement of $X/\stackrel{\rho}{\sim}$, X is extracted from S (line 10).

(*ii*) $X \in R \Longrightarrow X \cap Rea \neq \phi$
since a subset X is only put into R if either it contains q_{init} (line 9), or it contains successor states of a stable subset belonging to R (line 6).

(*iii*) When $R = S$, all the reachable classes are in R: If $X \in S$, all the classes directly reachable from X have been put in R (line 6).

(*iv*) So, when $R = S$, R defines a stable partition of *Rea*.

(*v*) The finiteness of the set of classes insures that the algorithm terminates.

Splitting a class : Line 4 of the algorithm splits a reachable class X into a partition $N = X/\stackrel{\rho}{\sim}$, whose elements are stable with respect to the current ρ. Let us detail the computation of this partition. Let *pre* denote the precondition function $\lambda Y.\{q \in Q \mid \exists q' \in Y \text{ such that } q \rightarrow q'\}$. Then,

$$N = \{X \cap \bigcap_{Y \in \rho} Z_Y \mid Z_Y \in \{pre(Y), Q - pre(Y)\}\}$$

Instead of considering such an exponential number of intersections, most of which are generally empty, we propose to compute N as follows:

$N = \{X\}$;
for each $Y \in \rho$ do
  $M := \phi$;
  for each W in N do
    let $W_1 = W \cap pre(Y)$;
    if $W_1 = W$ or $W_1 = \phi$ then $M := M \cup \{W\}$
    else $M := M \cup \{W_1, W - W_1\}$;
  od;
  $N := M$;
od

4 Example

Let us consider the following program, which could be a boolean abstraction of a more realistic program:

```
x := true; y := false; read(a);
loop
  write(x or y);
  z := a; read(a);
  w := x; x := not y; y := w or z;
end;
```

We want to examine all the possible input/output behaviours of this program. So, we consider it as a transition system, whose states are the values of the variables when the output is written. Now, since we are only interested in the output, we may consider as equivalent all the states which produce the same output. So, we start with the initial partition:

$$\{(a,w,x,y,z) \mid x \vee y = true\}, \{(a,w,x,y,z) \mid x \vee y = false\}$$

In the following, classes are represented by their characteristic formulas. The initial partition will be noted:

$$\rho : \quad C_1 = \{x \vee y\} \quad C_2 = \{\neg x \wedge \neg y\}$$

Standard rules of weakest precondition provide the precondition of a class X, with respect to the body of the loop:

$$pre(X) = X[w \vee z/y][\neg y/x][x/w] \downarrow a[a/z]$$

where $X \downarrow a = \exists a_0 X[a_0/a] = X[false/a] \vee X[true/a]$

So, $pre(C_1) = x \vee \neg y \vee a$, $pre(C_2) = \neg x \wedge y \wedge \neg a$

The successive partitions built by the algorithm are illustrated on figure 1.

Step 1: The only reachable class is C_1, since x is initially true. For splitting it, we compute:

$$C_1 \wedge pre(C_1) = (x \vee y) \wedge (x \vee \neg y \vee a) = x \vee (y \wedge a)$$

$$C_1 \wedge \neg pre(C_1) = \neg x \wedge y \wedge \neg a$$

$$C_1 \wedge pre(C_1) \wedge pre(C_2) = false$$

$$C_1 \wedge \neg pre(C_1) \wedge pre(C_2) = C_1 \wedge \neg pre(C_1)$$

So, C_1 is split into:

$$C_{11} = \{x \vee (y \wedge a)\} \quad C_{12} = \{\neg x \wedge y \wedge \neg a\}$$

and only C_{11} is reachable. We have: $pre(C_{11}) = x \vee \neg y \vee a$, $pre(C_{12}) = y \wedge (x \vee a)$

Step 2: For splitting C_{11}, we compute:

$$C_{11} \wedge pre(C_{11}) = (x \vee (y \wedge a)) \wedge (\neg y \vee x \vee a) = C_{11}$$

$$C_{11} \wedge pre(C_{11}) \wedge pre(C_{12}) = (x \vee (y \wedge a)) \wedge (y \wedge (x \vee a)) = y \wedge (x \vee a)$$

$$C_{11} \wedge pre(C_{11}) \wedge \neg pre(C_{12}) = x \wedge \neg y$$

$$C_{11} \wedge pre(C_{11}) \wedge pre(C_{12}) \wedge pre(C_2) = false$$

$$C_{11} \wedge pre(C_{11}) \wedge \neg pre(C_{12}) \wedge pre(C_2) = false$$

So, C_{11} is split into:

$$C_{111} = y \wedge (x \vee a) \quad C_{112} = x \wedge \neg y$$

and only C_{112} is reachable. We have: $pre(C_{111}) = x \vee a$, $pre(C_{112}) = \neg x \wedge \neg y \wedge \neg a$

Step 3: When splitting C_{112}, we get only:

$$C_{112} \wedge pre(C_{111}) \wedge \neg pre(C_{112}) \wedge \neg pre(C_{12}) \wedge \neg pre(C_2) = x \wedge \neg y = C_{112}$$

So C_{112} is stable, and leads to C_{111}. So, C_{111} is reachable.

Step 4: C_{111} is also found stable since:

$$C_{111} = C_{111} \wedge pre(C_{111}) \wedge \neg pre(C_{112}) \wedge pre(C_{12}) \wedge \neg pre(C_2)$$

It leads to itself and to C_{12}, which is found reachable.

Step 5: C_{12} is stable, and leads to C_2, since:

$$C_{12} = C_{12} \wedge \neg pre(C_{111}) \wedge \neg pre(C_{112}) \wedge \neg pre(C_{12}) \wedge pre(C_2)$$

Step 6: C_2 is split into:

$$C_{21} = C_2 \wedge pre(C_{111}) = \neg x \wedge \neg y \wedge a$$

$$C_{22} = C_2 \wedge pre(C_{112}) = \neg x \wedge \neg y \wedge \neg a$$

So C_{12} is removed from stable classes. We have: $pre(C_{21}) = pre(C_{22}) = \neg x \wedge y \wedge \neg a$

Step 7: C_{12} is again found stable, since:

$$C_{12} \wedge pre(C_{21}) \wedge pre(C_{22}) = C_{12}$$

It leads to C_{21} and C_{22}.

Step 8 and 9: From

$$C_{21} = C_{21} \wedge pre(C_{111})$$

$$C_{22} = C_{22} \wedge pre(C_{112})$$

we get that C_{21} and C_{22} are stable, and respectively lead to C_{111} and C_{112}.

We get a graph with 5 vertices (Fig. 2), instead of 16, which would be produced by standard generation (Fig. 3).

5 Conclusion

We have presented an algorithm combining generation and reduction methods. In our opinion, this algorithm is interesting for program verification: a state graph with several thousands (or even infinitely many) states may be reduced to one with a few number of states by considering an equivalence relation.

Of course, one must be capable to compute the function *pre* and intersections of classes, and to decide the inclusion of classes. Such a symbolic computation is achievable in the boolean case, with reasonable average cost [3,7].

Applying our algorithm to program verification appears very close to formal proof (in the Floyd/Hoare sense) or to what is now called "symbolic model checking" [4]. Concerning other applications, the algorithm is being implemented in the new version of the LUSTRE compiler.

We have not presented complexity measures. A comparison with classical reduction methods is difficult, mainly because the complexity of these methods is evaluated as a function of the size of the initial graph, whereas the cost of our method obviously depends on the size of the reduced graph.

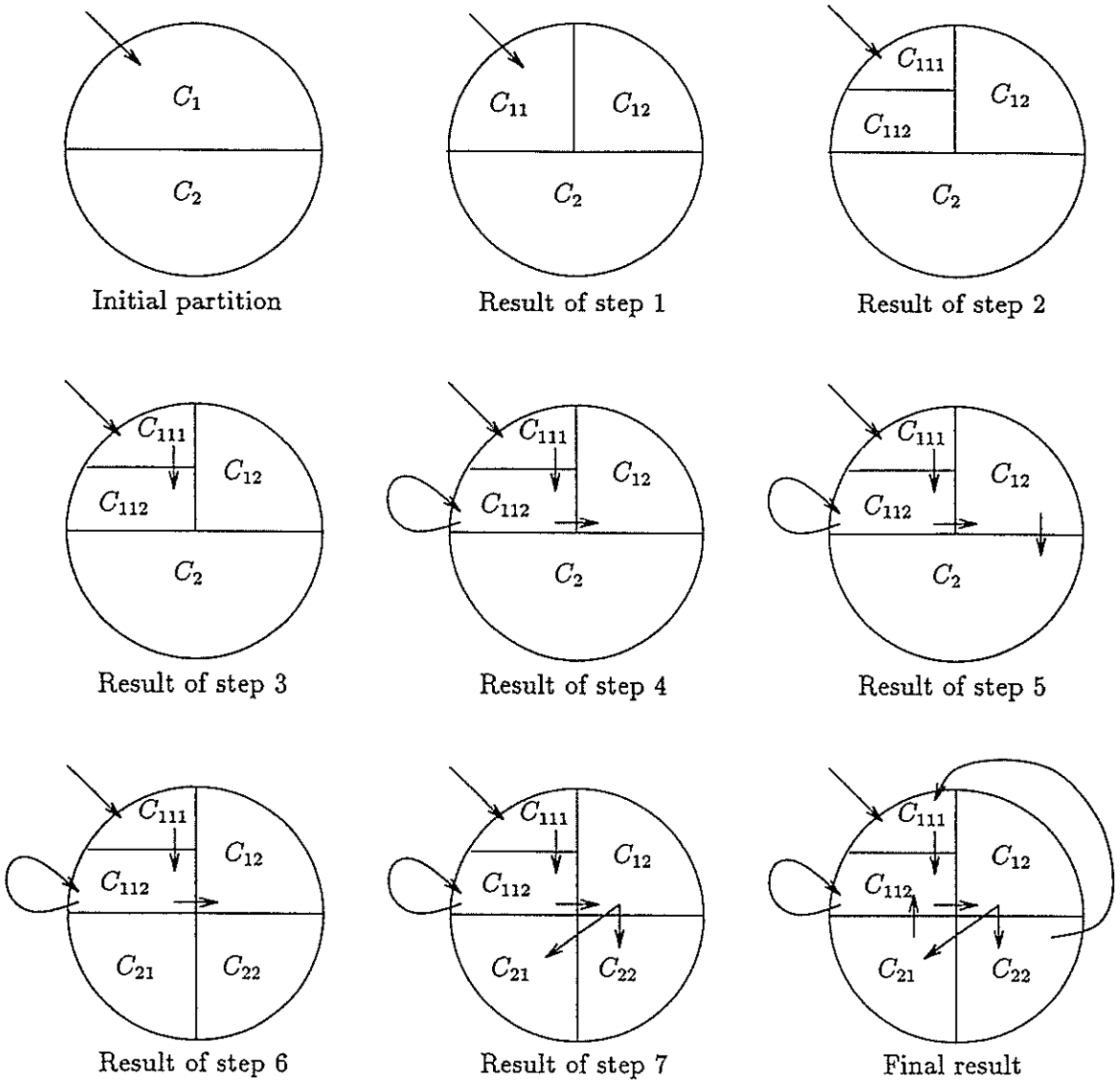

Figure 1: The successive partitions built by the algorithm

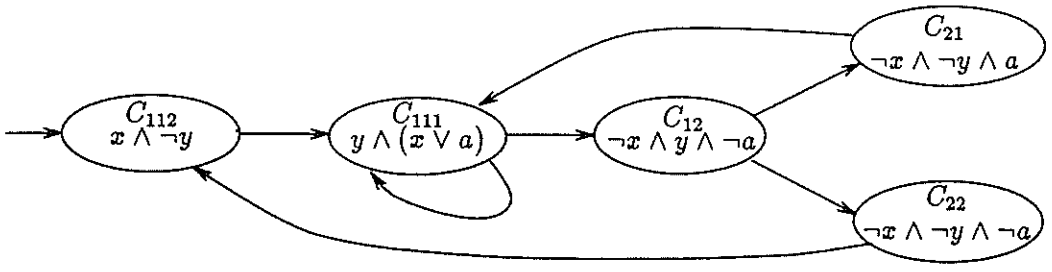

Figure 2: The reduced graph of the example

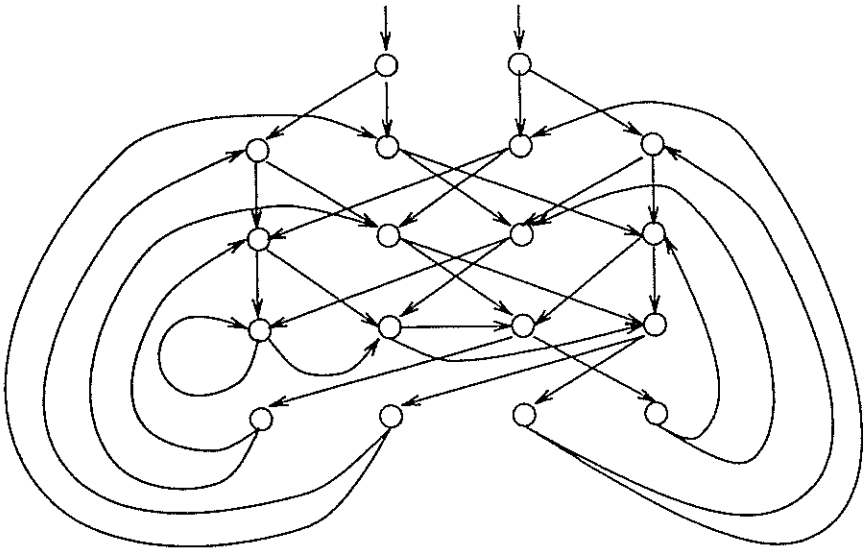

Figure 3: The complete graph of the example

References

[1] A. Aho, R. Sethi, J. Ullman. *Compilers: Principles, Techniques and Tools.* Addison-Wesley, 1986.

[2] G. Berry and G. Gonthier. *The synchronous programming language Esterel, design, semantics, implementation.* Tech. Report 327, INRIA, 1985. To appear in Science of Computer Programming.

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

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

[5] P. Caspi, D. Pilaud, N. Halbwachs, J. Plaice. Lustre: a declarative language for programming synchronous systems. In *14th POPL*, january 1987.

[6] E. Clarke, E.A. Emerson, A.P. Sistla. Automatic verification of finite state concurrent systems using temporal logic. In *10th. Annual Symp. on Principles of Programming Languages*, 1983.

[7] O. Coudert, C. Berthet, J. C. Madre. Verification of synchronous sequential machines based on symbolic execution. In *International Workshop on Automatic Verification Methods for Finite State Systems, LNCS 407*, Springer Verlag, 1989.

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

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

[10] R. Paige and R. Tarjan. Three partition refinement algorithms. *SIAM J. Comput.*, 16(6), 1987.

[11] J.L. Richier, C. Rodriguez, J. Sifakis, J. Voiron. Verification in Xesar of the sliding window protocol. In *17th International Workshop on Protocol Specification Testing and Verification*, 1987.