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 addresses 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 $S = (Q, \rightarrow, q_{init})$ 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 explicitly 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 $\rho$ 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 \( q \in Q \), \( \text{post}_\rho(q) \) is the set of classes in \( \rho \) immediately reachable from \( q \)
\[
\text{post}_\rho(q) = \{ X \in \rho \mid \exists q' \in X \text{ such that } q \rightarrow q' \}
\]

An equivalence relation, noted \( \sim \), is associated with \( \rho \) as follows:

\[
q_1 \sim q_2 \iff \text{post}_\rho(q_1) = \text{post}_\rho(q_2)
\]

A subset \( X \) of \( Q \) is said to be \text{stable with respect to} \( \rho \) if and only if it is included in some equivalence class of \( \mathcal{L} \). The partition \( \rho \) 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 \text{refinement} of a partition \( \rho \) is a partition \( \rho' \) such that:

\[
\forall X \in \rho', \exists Y \in \rho \text{ such that } X \subseteq Y
\]

The \text{reduction} of a transition system \( S \) with respect to a stable partition \( \rho \) is the transition system \((\rho, \sim, [q_{\text{init}}]_\rho)\), where

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

\[
X \sim Y \iff \exists q \in X, q' \in Y \text{ such that } q \rightarrow q'
\]

With the above terminology, given an initial partition \( \rho \) of \( Q \), we are looking for the reduction of \( S \) with respect to the least stable refinement of \( \rho \).

3 Algorithm

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

- The set \( R \) of \text{reachable classes}, i.e. the classes containing at least one element which has been found reachable from \( q_{\text{init}} \).
- The set \( S \) of \text{stable classes}, i.e. the reachable classes which have been found to belong to \( \mathcal{L} \).

The algorithm is the following:

\[
R = \{[q_{\text{init}}]_\rho \}; S = \phi; \quad (1)
\]

\[
\text{while } R \neq S \text{ do} \quad (2)
\]

\[
\text{choose } X \text{ in } R - S; \quad (3)
\]

\[
\text{let } N = X/\mathcal{L}; \quad (4)
\]

\[
\text{if } N = \{X\} \text{ then} \quad (5)
\]

\[
S := S \cup \{X\}; R := R \cup \{\text{post}_\rho(q) \mid q \in N\}; \quad (6)
\]

\[
\text{else} \quad (7)
\]

\[
R := R - \{X\}; \quad (8)
\]

\[
\text{if } \exists Y \in N \text{ such that } q_{\text{init}} \in Y \text{ then } R := R \cup \{Y\}; \quad (9)
\]

\[
S := S - \{Y \in S \mid X \in \text{post}_\rho(Y)\}; \quad (10)
\]

\[
\rho := (\rho - \{X\}) \cup N; \quad (11)
\]

\[
\text{fi} \quad (12)
\]

\[
\text{od} \quad (13)
\]
Proof: Let $Rea$ be the set of reachable states, that is the least subset $X$ of $Q$ containing $q_{init}$, and such that

$$(g \in X \land g \to q') \implies q' \in X$$

Then,

(i) $X \in S \implies X \in \mathcal{L}$

since a subset $X$ is only put into $S$ if $X = X/\mathcal{L}$ (line 6), and as soon as a refinement of $\rho$ can involve a refinement of $X/\mathcal{L}$, $X$ is extracted from $S$ (line 10).

(ii) $X \in R \implies X \cap Rea \neq \phi$

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 $N = X/\mathcal{L}$, whose elements are stable with respect to the current $\rho$. Let us detail the computation of this partition. Let $\text{pre}$ denote the precondition function $\lambda Y . \{ q \in Q \mid \exists q' \in Y \text{ such that } g \to q' \}$. Then,

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

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 $W_1 = W \cap \text{pre}(Y);$  

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:

```plaintext
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:

\[ \{(a, w, x, y, z) \mid z \lor y = \text{true}\}, \{(a, w, x, y, z) \mid z \lor y = \text{false}\} \]

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

\[ \rho : \quad C_1 = \{z \lor y\} \quad C_2 = \{\neg z \land \neg y\} \]

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

\[ \text{pre}(X) = X[w \lor z/y]\neg y/z][z/w] \downarrow a[a/z] \]

where \(X \downarrow a = \exists a_0 X[a_0/a] = X[\text{false}/a] \lor X[\text{true}/a] \)

So, \(\text{pre}(C_1) = z \lor \neg y \lor a\), \(\text{pre}(C_2) = \neg z \land y \land \neg a\)

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

**Step 1:** The only reachable class is \(C_1\), since \(z\) is initially true. For splitting it, we compute:

\[ C_1 \land \text{pre}(C_1) = (z \lor y) \land (z \lor \neg y \lor a) = z \lor (y \land a) \]

\[ C_1 \land \neg \text{pre}(C_1) = \neg z \land y \land \neg a \]

\[ C_1 \land \text{pre}(C_1) \land \text{pre}(C_2) = \text{false} \]

\[ C_1 \land \neg \text{pre}(C_1) \land \text{pre}(C_2) = C_1 \land \neg \text{pre}(C_1) \]

So, \(C_1\) is split into:

\[ C_{11} = \{z \lor (y \land a)\} \quad C_{12} = \{\neg z \land y \land \neg a\} \]

and only \(C_{11}\) is reachable. We have: \(\text{pre}(C_{11}) = z \lor \neg y \lor a\), \(\text{pre}(C_{12}) = y \land (z \lor a)\)

**Step 2:** For splitting \(C_{11}\), we compute:

\[ C_{12} \land \text{pre}(C_{11}) = (z \lor (y \land a)) \land (\neg y \lor z \lor a) = C_{11} \]

\[ C_{11} \land \text{pre}(C_{11}) \land \text{pre}(C_{12}) = (x \lor (y \land a)) \land (y \land (z \lor a)) = y \land (z \lor a) \]

\[ C_{11} \land \text{pre}(C_{11}) \land \neg \text{pre}(C_{12}) = z \land \neg y \]

\[ C_{11} \land \text{pre}(C_{11}) \land \text{pre}(C_{12}) \land \text{pre}(C_2) = \text{false} \]

\[ C_{11} \land \text{pre}(C_{11}) \land \neg \text{pre}(C_{12}) \land \text{pre}(C_2) = \text{false} \]

So, \(C_{11}\) is split into:

\[ C_{111} = y \land (z \lor a) \quad C_{112} = z \land \neg y \]

and only \(C_{112}\) is reachable. We have: \(\text{pre}(C_{111}) = z \lor a\), \(\text{pre}(C_{112}) = z \land \neg y \land \neg a\)
Step 3: When splitting $C_{12}$, we get only:

$$C_{12} \land \text{pre}(C_{111}) \land \neg\text{pre}(C_{12}) \land \neg\text{pre}(C_{12}) \land \neg\text{pre}(C_{12}) = x \land \neg y = C_{112}$$

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

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

$$C_{111} = C_{111} \land \text{pre}(C_{111}) \land \neg\text{pre}(C_{112}) \land \text{pre}(C_{12}) \land \neg\text{pre}(C_{2})$$

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

Step 5: $C_{12}$ is stable, and leads to $C_{2}$, since:

$$C_{12} = C_{12} \land \neg\text{pre}(C_{111}) \land \neg\text{pre}(C_{112}) \land \neg\text{pre}(C_{12}) \land \text{pre}(C_{2})$$

Step 6: $C_{2}$ is split into:

$$C_{21} = C_{2} \land \text{pre}(C_{111}) = \neg x \land \neg y \land a$$

$$C_{22} = C_{2} \land \text{pre}(C_{112}) = \neg x \land \neg y \land \neg a$$

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

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

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

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

Step 8 and 9: From

$$C_{21} = C_{21} \land \text{pre}(C_{111})$$

$$C_{22} = C_{22} \land \text{pre}(C_{112})$$

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 $\text{pre}$ 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


