January 1993

A State Minimization Algorithm for Communicating State Machines With Arbitrary Data Space

Inhye Kang
University of Pennsylvania

Insup Lee
University of Pennsylvania, lee@cis.upenn.edu

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-93-07.

A State Minimization Algorithm for Communicating State Machines With Arbitrary Data Space

Abstract

A fundamental issue in the automated analysis of communicating systems is the efficient generation of the reachable state space. Since it is not possible to generate all the reachable states of a system with an infinite number of states, we need a way to combine sets of states. In this paper, we describe communicating state machines with data variables, which we use to specify concurrent systems. We then present an algorithm that constructs the minimal reachability graph of a labeled transition system with infinite data values. Our algorithm clusters a set of states that are bisimilar into an equivalent class. We include an example to illustrate our algorithm and identify a set of sufficient conditions that guarantees the termination of the algorithm.

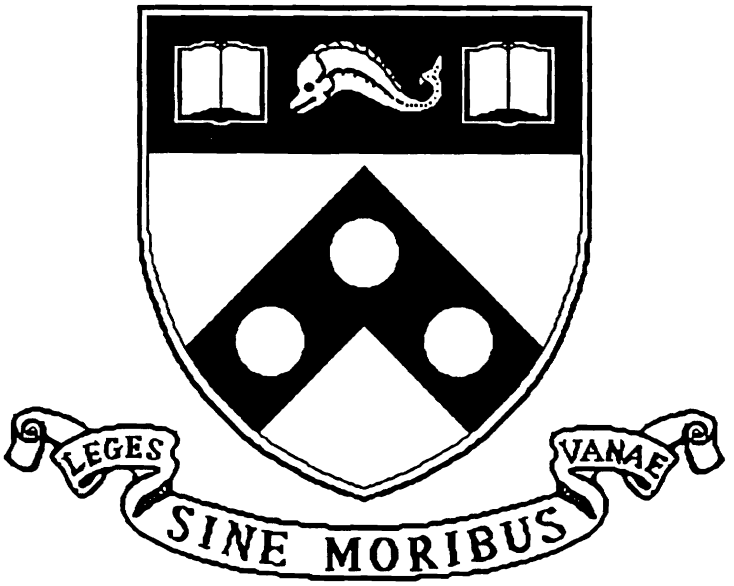
Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-93-07.

A State Minimization Algorithm for Communicating State Machines with Arbitrary Data Space

MS-CIS-93-07
LOGIC & COMPUTATION 55
DISTRIBUTED SYSTEMS LAB 13

Inhye Kang
Insup Lee



University of Pennsylvania
School of Engineering and Applied Science
Computer and Information Science Department
Philadelphia, PA 19104-6389

January 1993

A State Minimization Algorithm for Communicating State Machines with Arbitrary Data Space *

Inhye Kang and Insup Lee
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104-6389

Abstract

A fundamental issue in the automated analysis of communicating systems is the efficient generation of the reachable state space. Since it is not possible to generate all the reachable states of a system with an infinite number of states, we need a way to combine sets of states. In this paper, we describe communicating state machines with data variables, which we use to specify concurrent systems. We then present an algorithm that constructs the minimal reachability graph of a labeled transition system with infinite data values. Our algorithm clusters a set of states that are bisimilar into an equivalent class. We include an example to illustrate our algorithm and identify a set of sufficient conditions that guarantees the termination of the algorithm.

1 Introduction

As distributed computer systems have become widely available, more and more safety-critical applications are implemented as concurrent systems. Because of the importance of their correct operation, interests in analysis techniques for concurrent systems have been increasing. Moreover, due to high complexity inherent in such applications, it is desirable to develop analysis techniques that can be automated. Several automatic analysis techniques for finite systems have been developed and used in practice [CES86, CPS88]. Such techniques are based on *state space exploration*. The major weakness of the state space exploration based approach is that the size of the state space grows exponentially with the number of processes and thus creates the state space explosion problem. Thus, some techniques such as compositional analysis [YY91] and symbolic representation [BCMD90] of systems have been developed to avoid this problem. However, these techniques are limited to finite state systems and still cannot handle systems with an extremely large number of resulting states due to the storage and speed limitations of computers.

Another approach to reduce the number of states that need to be represented is to cluster states that are bisimilar into an equivalent class. This approach is practical in the case that the reachability graph contains a lot of states which are equivalent. Bouajjani *et al.*[BFH90] have proposed an efficient algorithm which constructs the minimal reachability graph of an unlabeled transition system. Alur *et al.*[ACH+92] have extended the algorithm to deal with timed transition systems without data variables. An unlabeled

*This research was supported in part by ONR N00014-89-J-1131 and DARPA/NSF CCR90-14621.

transition system, however, is not suitable in describing concurrent systems since it cannot capture internal actions or communication actions.

The algorithm described in this paper extends the algorithm by Bouajjani *et al.* to a labeled transition system which may have infinitely many states. In this paper, concurrent systems are specified using Communicating State Machines (CSMs). Each CSM has local data variables whose values are from arbitrary domains, and is basically a transition system with transitions that are guarded by enabling conditions over variables. CSMs also support one-to-many synchronous communication with value passing. Although our minimization algorithm does not guarantee termination, we believe it to be powerful enough to handle many interesting communicating systems with an infinite number of reachable states. As a continuing work, we have identified a set of sufficient conditions on the syntax of CSMs that guarantees termination.

The rest of the paper is organized as follows. In Section 2, we overview other methods related with our work. Section 3 defines the syntax and semantics of CSM. Section 4 describes our state minimization algorithm, presents an example, and states a set of properties. Section 5 concludes the discussion.

2 Related Work

Several approaches have been proposed to generate a minimal transition system with respect to the greatest bisimulation. Kanellakis and Smolka [KS90] show that the strong bisimulation equivalence of two labeled transition systems with finite states can be reduced to the relational coarsest partition problem. Their algorithm is based on the state minimization algorithm for deterministic finite automata by Hopcroft [Hop71]. Fernandez [Fer90] provides an algorithm for minimizing the number of states of a labeled transition system with respect to bisimulation equivalence. This algorithm is based on a more efficient algorithm for the relational coarsest partition problem developed by Pagie and Tarjan [PT87]. The main drawback of Fernandez's algorithm is that it explores the whole set of states including unreachable states. Thus, it can be applied only to systems with a finite state space.

For unlabeled transition systems, Bouajjani *et al.* [BFH90, BFHR92] describe an algorithm to find the coarsest partition with respect to an equivalence relation. This equivalence relation identifies two states if they can reach the same class of states.

To handle infinite data values, Jonsson and Parrow [JP89] provide a technique to change a program with infinite states due to data variables into an equivalent finite state program. The approach used in the algorithm is to represent the data values of a variable using a finite number of symbols. This technique, however, is limited to programs in which control statements are data-independent.

3 Communicating State Machines

We use Communicating State Machines (CSMs) for the specification and analysis of concurrent communicating systems. They are state transition systems which support data variables and one-to-many communi-

cation. CSM extends Shaw's communicating state machines [Sha91] with one-to-many communication and composition.

In CSMs, a system consists of a finite set of concurrent components. The components may execute concurrently or communicate with other components. They communicate messages through channels called events. The basic strategy of communication is one-to-many synchronization.

Definition 3.1 *A communicating state machine is a tuple* $M = \langle V, N, n_0, I, \Sigma, T \rangle$, *where*

- $V = \{x_1, x_2, \cdots, x_k\}$ *is a finite set of data variables,*
- N *is a finite set of nodes,*
- n_0 *is the initial node in* N,
- I *is the initialization function over* V,
- Σ *is a finite set of events, and*
- T *is a finite set of transitions.*

Variables. We do not restrict the domains of variables in V. Each variable $x \in V$ has an associated domain $dom(x)$. We denote the data space of M by $D^k = dom(x_1) \times dom(x_2) \times \cdots dom(x_k)$. Let v in D^k represent a list of values of variables. We view v as a function from V to $D = \cup_i dom(x_i)$, where for each x_i in V, $v(x_i)$ is the ith element in v.

Initialization. The initialization function maps each variable to the set of its possible initial values. As an example, the initialization of an integer variable can be given by:

$$I(x) ::= x = i | x \leq i | i \leq x | i \leq x \leq i' \text{ where } i \text{ and } i' \text{ are integers.}$$

We use $D^I = \{v | \forall x \in V. v(x) \in I(x)\}$ to represent the set of the initial values.

Events. CSMs synchronously communicate messages through events. The number and order of messages associated with events is fixed. For each communication event, there can be one sender and multiple receivers. Associated with an event e, there are two kinds of communication operations: $e!(exp_1, \cdots, exp_n)$ for sending messages $exp_1, \cdots, exp_n$ through e, and $e?X$ for receiving messages on a set X of tuples of variables through e. For example, after two CSMs execute $e!(1, 5.5)$ and $e?\{(x, y), (x', y')\}$ simultaneously, the values of x, x', y, y' become 1, 1, 5.5, 5.5, respectively. For an event e, we define $size(e)$ to be the number of messages that are sent or received simultaneously through the event e, and $dom(e, i)$ to be the domain of the ith message of the event e, for $1 \leq i \leq size(e)$.

Transitions. A transition, (n_1, c, α, h, n_2), from node n_1 to node n_2, can be taken if the enabling condition c is true and the set α of communication operations is ready. The values of variables are changed by transformation function h from D^k to D^k, where h can be represented as a set of assignments. As an example, suppose a transition $(n_1, x \leq y, \{e_1!, e_2?\}, \{x := x + y\}, n_2)$ is taken. If the values of x, y are 0, 5 at node n_1, the values of x and y are both 5 after the transition.

Composition. A CSM process can be constructed by composing two CSM processes. We assume two component processes do not share variables, that is, all variables used within each CSM are local. This restriction is reasonable since the communication of CSMs is based on message passing instead of shared variables, and also makes it easier to define composition. Unlike variables, events can be shared among processes and are used to define communication channels.

For a communication operation o and a set α of communication operations, let $event(o)$ and $events(\alpha)$ represent the event in o and the set of events appearing in α, reprectively. For example, $event(e?S)$ is e_1, and $events(\{e_1!(1, 5.5), e_2?\{(x, y)\}, e_3!\})$ is $\{e_1, e_2, e_3\}$. And let $var_e(\alpha, i)$ represent the set of variables associated with the ith message of e such that e is an event for receiving messages in α. If event $e?\{(x, y), (x', y')\}$ is in α, $var_e(\alpha, 2)$ is $\{y, y'\}$. For any operation o, $out(o)$ is true only if o is a receive operation.

Definition 3.2 *Suppose* $M_1 = \langle V_1, N_1, n_{10}, I_1, \Sigma_1, T_1 \rangle$ *and* $M_2 = \langle V_2, N_2, n_{20}, I_2, \Sigma_2, T_2 \rangle$. *The composition of* M_1 *and* M_2 *(*$M_1 || M_2$*) is defined as a tuple* $M = \langle V, N, n_0, I, \Sigma, T \rangle$, *where*

- $V = V_1 \cup V_2$,
- $N = N_1 \times N_2$,
- *The initial node* n_0 *is* (n_{10}, n_{20}),
- *The initialization function* I *is given by:*

$$I(x) = \begin{cases} I_1(x) & \text{if } x \in V_1 \\ I_2(x) & \text{otherwise} \end{cases}$$

- $\Sigma = \Sigma_1 \cup \Sigma_2$.
- T *is defined as follows:*
  for each transition $(n_1, c_1, \alpha_1, h_1, n_1')$ *in* T_1 *and each transition* $(n_2, c_2, \alpha_2, h_2, n_2')$ *in* T_2,
  - *Asynchronous action:*
    $((n_1, n_2), c_1, \alpha_1, h_1, (n_1', n_2)) \in T$ *if* $events(\alpha_1) \cap \Sigma_2 = \emptyset$.
    $((n_1, n_2), c_2, \alpha_2, h_2, (n_1, n_2')) \in T$ *if* $events(\alpha_2) \cap \Sigma_1 = \emptyset$.
  - *Concurrent action:*
    $((n_1, n_2), c_1 \wedge c_2, \alpha_1 \cup \alpha_2, h_1 \circ h_2, (n_1', n_2')) \in T$ *if* $events(\alpha_1) \cap \Sigma_2 = events(\alpha_2) \cap \Sigma_1 = \emptyset$.

– *Synchronization:*

$((n_1, n_2), c_1 \wedge c_2, \alpha, h, (n_1', n_2')) \in T$ *if* $events(\alpha_1) \cap \Sigma_2 = events(\alpha_2) \cap \Sigma_1 \neq \emptyset$, *where*

$$\alpha = \begin{array}{l} \{o \in \alpha_1 \cup \alpha_2 | (event(o) \notin \Sigma_1 \cap \Sigma_2) \vee (event(o) \in \Sigma_1 \cap \Sigma_2 \wedge out(o))\} \cup \\ \{e?(S_1 \cup S_2) | e?S_1 \in \alpha_1 \wedge e?S_2 \in \alpha_2\} \end{array}$$

$$h(v)(x) = \begin{cases} exp_i & \text{if } \exists e, i.((e?S, e!(\overline{exp}) \in \alpha_1 \cup \alpha_2) \wedge (x \in var_e(\alpha_1 \cup \alpha_2, i))) \\ (h_1 \circ h_2)(v)(x) & \text{otherwise.} \end{cases}$$

Execution. In CSMs, a *state* s is of the form (n, v) where n is a node and v is a valuation of variables in D^k. If a CSM executes a transition (n_1, c, α, h, n_2) with the valuation v of variables at node n_1, the set of next valuations is given by the following function f:

$$\begin{array}{rll} f(v, \alpha, h) = \{v' | & (\forall x \in V. & v'(x) \in dom(e, i) \text{ if } x \in var_e(\alpha, i) \text{ for some } e, i \\ & & v'(x) = h(v)(x) \quad \text{otherwise }) \wedge \\ & (\forall x, y \in V. & v'(x) = v'(y) \text{ if } x, y \in var_e(\alpha, i) \text{ for some } e, i)\} \end{array}$$

That is, the value of each variable x is changed by h except when the variable is associated with any receive operation in o. In that case, the value can be one of possible inputs associated with the operation, but the values of variables associated with the same message should be the same.

An execution of a CSM is defined as follows:

Definition 3.3 *An execution of a CSM* $M = \langle V, N, n_0, I, \Sigma, T \rangle$ *is a finite or infinite sequence of the form*

$$s_0 \xrightarrow{c_0, \alpha_0, h_0} s_1 \xrightarrow{c_1, \alpha_1, h_1} s_2 \xrightarrow{c_2, \alpha_2, h_2} s_3 \cdots$$

where $s_i = (n_i, v_i)$ *satisfying*

1. *Initiality:* $v_0(x) \in I(x)$ *for every* $x \in V$.
2. *Succession Constraint: for each* i, *there exists* $(n_i, c_i, \alpha_i, h_i, n_{i+1})$ *in* T *such that* $c_i(v_i)$ *is true and* $v_{i+1}(x) \in f(v_i, \alpha_i, h_i)(x)$.

An Example of a Communication Protocol. To illustrate the expressiveness of CSM, consider the simple communication protocol. Figure 1 shows the communication system as consisting of four components: a receiver, two senders, and a medium.

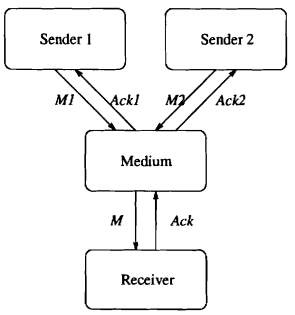


Figure 1: A Communication System

Figure 2 shows the CSM processes of the communication system. The processes are specified as follows: 1) The receiver waits for a message. When a message arrives, it sends an acknowledgement to the source of the message; 2) The sender i for $i = 1, 2$ sends a message, waits an acknowledge, and returns to the initial node; 3) The medium accepts an input (a message or an acknowledgement) and sends an output to an appropriate sender or a receiver. Figure 3 shows the composition of the receiver and the medium.

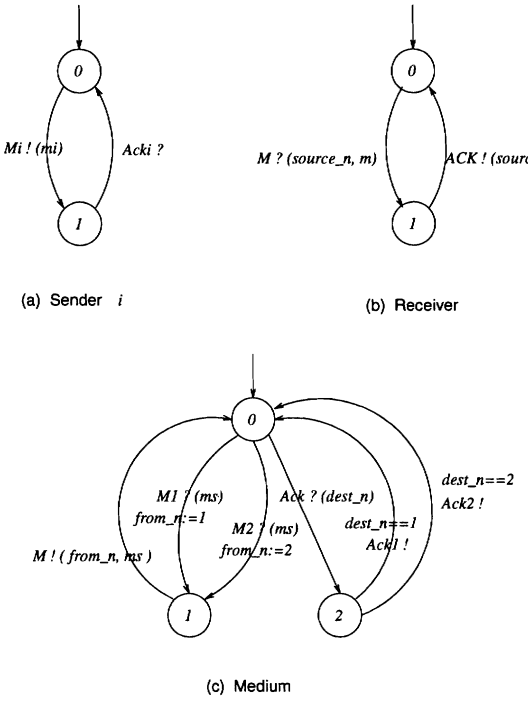


Figure 2: CSMs of A Communication System

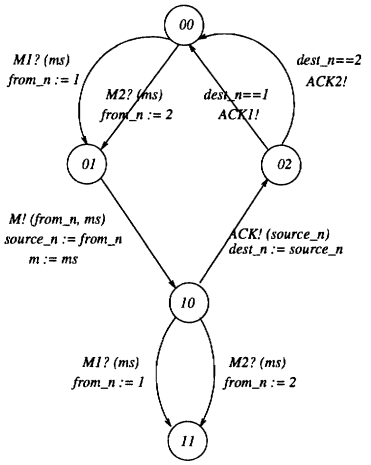


Figure 3: The Composition of A Receiver and A Medium

4 The Minimal Transition System of A CSM

We now describe an algorithm to construct the minimal reachability graph of a CSM with respect to bisimulation. Since our algorithm is based on the algorithm that computes the minimal graph of transition systems given by Bouajjani *et al.* [BFH90], we briefly sketch that algorithm first.

4.1 Notations

A *labeled transition system* is defined as $S = \langle Q, A, \rightarrow, Q_0 \rangle$ where Q is a set of states, A is a set of actions, $\rightarrow \subseteq Q \times A \times Q$ is the transition relation, and Q_0 is a set of the initial states. For convenience, we use a notation :

$$q \xrightarrow{a} q' \text{ for } (q, a, q') \in \rightarrow .$$

The notion of bisimulation as defined by Milner in [Mil89] is used.

Definition 4.1 *Given a labeled transition system $S = \langle Q, A, \rightarrow, Q_0 \rangle$, a binary relation $\rho \subseteq Q \times Q$ is a bisimulation iff*

$$\forall (q_1, q_2) \in \rho. \quad \forall a \in A. \quad \begin{array}{l} \forall r_1.(q_1 \xrightarrow{a} r_1 \Rightarrow \exists r_2.(q_2 \xrightarrow{a} r_2 \wedge (r_1, r_2) \in \rho)) \wedge \\ \forall r_2.(q_2 \xrightarrow{a} r_2 \Rightarrow \exists r_1.(q_1 \xrightarrow{a} r_1 \wedge (r_1, r_2) \in \rho)). \end{array}$$

For $q, q' \in Q$, q and q' are said to be *bisimilar* if $(q, q') \in \rho$ for some bisimulation $\rho \in Q \times Q$.

We represent an equivalence relation $\rho \subseteq Q \times Q$ as a partition $\rho = \{X_i | i \in I\}$ where X_i represents an equivalence class in ρ. A partition ρ' is a *refinement* of a partition ρ denoted $\rho' \sqsubseteq \rho$ iff :

$$\forall X' \in \rho'. \exists X \in \rho. (X' \subseteq X).$$

Let $[q]_\rho$ denote the class in ρ which includes q and let $[Q]_\rho$ denote the set of classes in ρ which include some state in Q, that is, $[Q]_\rho = \{[q]_\rho | q \in Q\}$. The *reduction* of a labeled transition system $S = \langle Q, A, \rightarrow, Q_0 \rangle$, according to a bisimulation ρ of Q is the transition system $S/\rho = \langle \rho, A, \rightarrow_\rho, [Q_0]_\rho \rangle$ where $\rightarrow_\rho = \{([q]_\rho, a, [q']_\rho) | (q, a, q') \in \rightarrow\}$.

Given a CSM $M = \langle V, N, n_0, I, \Sigma, T \rangle$, the corresponding labeled transition system is $S_M = \langle Q, A, \rightarrow, Q_0 \rangle$, where

- $Q = N \times D^k$,
- $A = 2^\Sigma$,
- $\rightarrow = \{((n, v), events(\alpha), (n', v')) | \exists (n, c, \alpha, h, n') \in T. c(v) \wedge v' \in f(v, \alpha, h)\}$,
- $Q_0 = \{(n_0, v_0) | v_0 \in D^I\}$.

We note that $\rightarrow$ is a subset of $Q \times A \times Q$.

4.2 State minimization of an unlabeled transition system

Bouajjani *et al.* [BFH90] provided an algorithm to compute the coarsest partition of the state space in a given system. Since the algorithm plays an important role in our approach, we briefly review the algorithm.

In the algorithm, a system is defined as an unlabeled transition system $S = \langle Q, \rightarrow, q_0 \rangle$ where Q is the set of states, $\rightarrow \subseteq Q \times Q$ is the transition relation, and q_0 is the initial state. For a state q in Q and a subset X of Q, the notation $q \rightsquigarrow X$ is used to denote $q \rightarrow q'$ for some $q' \in X$.

The notion of bisimulation of unlabeled transition systems is slightly different from that of labeled transition systems in Definition 4.1. A binary relation $\rho \subseteq Q \times Q$ is a *bisimulation* iff

$$\forall (q_1, q_2) \in \rho. \quad \forall r_1.(q_1 \rightarrow r_1 \Rightarrow \exists r_2.(q_2 \rightarrow r_2 \wedge (r_1, r_2) \in \rho)) \wedge \\ \forall r_2.(q_2 \rightarrow r_2 \Rightarrow \exists r_1.(q_1 \rightarrow r_1 \wedge (r_1, r_2) \in \rho)).$$

Given a partition ρ of Q, a class $X \in \rho$ is said to be *stable* with respect to ρ iff

$$\forall Y \in \rho.[(\exists x \in X.x \rightsquigarrow Y) \text{ implies } (\forall x \in X.x \rightsquigarrow Y)].$$

A partition ρ is said to be stable iff every class of ρ is *stable* with respect to ρ.

Proposition 4.1 *[BFH90] A partition ρ is stable iff ρ is a bisimulation.*

Therefore, the problem of finding the greatest bisimulation that refines a given partition is reduced to the problem of finding the coarsest stable partition that refines the given partition.

The state minimization algorithm of [BFH90] is as follows: 1) Given the initial partition ρ_0, the set S of stable and reachable classes is initially empty. Let the initial class be a class including the initial state. 2) If S does not include the initial class, then let X be the initial class; otherwise, select a class $X \notin S$ which is immediately reachable from S. If X is stable with respect to any class in the current partition, then insert X in S; otherwise, split X into the largest subclasses in which each subclass is stable with respect to all classes in the current partition. In the latter case, the classes in S from which X is immediately reachable are removed from S. 3) Repeat step 2 until all reachable classes are stable with respect to one another.

In order to use the algorithm it must be possible to define the following three operators for a class X of ρ from the transition systems:

1) $split(X, \rho)$ divides X into the largest subclasses which are all stable with respect to ρ.

2) $pre_\rho(X)$ denotes the set of classes of ρ which contains at least one state from which a state of X is immediately reachable.

3) $post_\rho(X)$ denotes the set of classes of ρ which contains at least one state immediately reachable from a state of X.

By Proposition 4.1, the set S of stable and reachable classes from the algorithm is the greatest bisimulation refining ρ_0.

4.3 Construction of the minimal transition system from a CSM

We now describe an algorithm that finds the coarsest partition of a CSM to construct its minimal state graph with respect to bisimulation. First, we extend the algorithm of Bouajjani *et al.* [BFH90] to handle more

Given an initial partition ρ_0:

$\rho := \rho_0$; $S := \emptyset$;
$R := \{[Q_0]_\rho\}$;
while $R \neq S$ **do**
  choose X in $R - S$;
  $N := split(X, \rho)$;
  if $N = \{X\}$ **then**
    $S := S \cup \{X\}$;
    $R := R \cup post_\rho(X)$;
  else
    $R := R - \{X\}$;
    $R := R \cup \{Y \in N | Y \cap Q_0 \neq \emptyset\}$;
    $S := S - pre_\rho(X)$;
    $\rho = (\rho - \{X\}) \cup N$;

Table 1: State Minimization Algorithm

than one initial states. We then explain how to define an initial partition for a given CSM. After defining the notion of *stability* in CSM, we redefine the functions *split*, pre_ρ and $post_\rho$ that are used in the algorithm of Bouajjani *et al.*

State Minimization with Bisimulation. The state minimization algorithm for CSM, called SMB, is shown in Table 1. Our algorithm is similar to the algorithm by Bouajjani *et al.*, except that we have modified to deal with a system with more than one initial states. This modification is necessary since a CSM can have finitely or infinitely many initial states.

The Initial Partition. Given a CSM $M = \langle V, N, n_0, I, \Sigma, T \rangle$, the initial partition of its whole state space $N \times D^k$ is given as $\rho_0 = \{(n, D^k) | n \in N\}$ for efficiency of implementation. The reason is that it is important to partition the data space since the number of states depends on the data space rather than the number of nodes. Furthermore, it is possible to equate a set of states from the same node since enabling conditions in transitions from a given node are good criteria for dividing the data space. Let Z, called a region, represent a subset of the data space D^k. An equivalence class is of the form $\{(n, v) | v \in Z\}$, which we represent as (n, Z).

Stability. We define a notion of stability for our labeled transition system, which is different from that for the unlabeled transition system used in [BFH90]. Let ρ be a partition of $N \times D^k$. We overload the first argument of the function f (defined in Section 3) to allow a region as well as a valuation. Given a region Z, an action α and a transformation h, the set of possible next valuations $f(Z, \alpha, h)$ is computed by:

$$f(Z, \alpha, h) = \bigcup_{v \in Z} f(v, \alpha, h).$$

Given a valuation v', an action α and a transformation h, we let $f^{-1}(v', \alpha, h)$ be the set of valuations whose next valuations include valuation v' and $f^{-1}(Z', \alpha, h)$ be the set of valuations whose next valuations include a valuation in region Z'. They are defined as follows:

$$f^{-1}(v', \alpha, h) = \begin{cases} \emptyset & \text{if } v'(x) \neq v'(y) \text{ for some } x, y \in V \text{such that } x, y \in var_e(\alpha, i) \text{ for some } e, i) \\ \{v \in D^k | v'(x) = h(v)(x) \text{ if } x \notin var_e(\alpha, i) \text{ for any } e, i\} & \text{otherwise} \end{cases}$$

$$f^{-1}(Z', \alpha, h) = \bigcup_{v' \in Z'} f^{-1}(v', \alpha, h)$$

For $a \in A$ and $X, Y \in \rho$, we define $\Phi_{a,Y}(X)$, which is the set of states in X which can lead to Y through action a, as follows:

$$\Phi_{a,Y}(X) = \bigcup_{(n,c,\alpha,h,n') \in T.a=events(\alpha)} (n, Z \cap c \cap f^{-1}(Z', \alpha, h))$$

where $X = (n, Z)$, $Y = (n', Z')$ for $n, n' \in N$, $Z, Z' \subseteq D^k$.

Definition 4.2 *1. A class X is said to be a-stable with respect to a class Y iff whenever some state in X can lead to Y through a, every other state in X can also lead to Y through a, i.e.,*

$$\Phi_{a,Y}(X) = X \vee \Phi_{a,Y}(X) = \emptyset.$$

2. A class X is said to be stable with respect to a class Y iff for every action $a \in A$, X is a-stable with respect to Y, i.e.,

$$\forall a \in A.(\Phi_{a,Y}(X) = X \vee \Phi_{a,Y}(X) = \emptyset).$$

3. A class X is said to be stable with respect to a partition ρ iff for every class $Y \in \rho$, X is stable with respect to Y, i.e.,

$$\forall Y \in \rho. \forall a \in A.(\Phi_{a,Y}(X) = X \vee \Phi_{a,Y}(X) = \emptyset).$$

4. A partition ρ' is said to be stable with respect to a partition ρ iff for every class $X \in \rho'$, X is stable with respect to ρ, i.e.,

$$\forall X \in \rho'. \forall Y \in \rho. \forall a \in A.(\Phi_{a,Y}(X) = X \vee \Phi_{a,Y}(X) = \emptyset).$$

5. A partition ρ is said to be stable *iff ρ is stable with respect to itself.*

Proposition 4.2 *A partition ρ is stable iff ρ is a bisimulation.*

Proof. Suppose a partition ρ is stable. By definition of stability,

$$\forall X, Y \in \rho. \forall a \in A.(\Phi_{a,Y}(X) = X \vee \Phi_{a,Y}(X) = \emptyset).$$

That is,

$$\begin{aligned} \forall X, Y \in \rho. \forall q_1, q_2 \in X. \quad & \forall a \in A. \\ & \forall r_1 \in Y(q_1 \xrightarrow{a} r_1 \Rightarrow \exists r_2 \in Y. q_2 \xrightarrow{a} r_2) \wedge \\ & \forall r_2 \in Y.(q_2 \xrightarrow{a} r_2 \Rightarrow \exists r_1 \in Y. q_1 \xrightarrow{a} r_1). \end{aligned}$$

This is equivalent to the definition of bisimulation since ρ is a partition. □

Proposition 4.2 is the same as Proposition 4.1 in the previous section. Thus, we can use the algorithm by Bouajjani *et al.* to find the minimal graph of a CSM with our notion of stability.

Three Operators. We define the operators $split(X, \rho)$, $pre_\rho(X)$, and $post_\rho(X)$ that are used in SMB. Let π represent a partial partition or a partition of Q (*i.e.*, $N \times D^k$). We define the operators $split_{a,Y}(\pi)$, $split_Y(\pi)$ and $split_\rho(\pi)$ needed to compute a stable partition.

For each $a \in A$ and $Y \in \rho$, we define the operator $split_{a,Y}(\pi)$ as follows:

$$split_{a,Y}(\pi) = \{\Phi_{a,Y}(X) | X \in \pi\} \cup \{X - \Phi_{a,Y}(X) | X \in \pi\}.$$

That is, $split_{a,Y}(\pi)$ splits each class X in π into at most two subclasses: one for a set of states that can lead to Y through a and another for a set of states that cannot lead to Y through a.

Proposition 4.3 *(properties of $split_{a,Y}(\pi)$)*

1. $split_{a,Y}(\pi) \sqsubseteq \pi$
2. $split_{a_1,Y} \circ split_{a_2,Y} = split_{a_2,Y} \circ split_{a_1,Y}$
3. *Each class in $split_{a,Y}(\pi)$ is a-stable with respect to Y.*

Proof.

1. For each $X' \in split_{a,Y}(\pi)$, there exists $X \in \pi$ such that $\Phi_{a,Y}(X) = X'$ or $X - \Phi_{a,Y}(X) = X'$, that is, $X' \subseteq X$.

2. We first show that $(split_{a_1,Y} \circ split_{a_2,Y})(\pi) \sqsubseteq (split_{a_2,Y} \circ split_{a_1,Y})(\pi)$. By Proposition 4.3, $(split_{a_1,Y} \circ split_{a_2,Y})(\pi) \sqsubseteq \pi$. That is, for every $X' \in (split_{a_1,Y} \circ split_{a_2,Y})(\pi)$, there is $X \in \pi$ such that $X' \subseteq X$. Then X' is one of the following four classes:

   $\{q \in X | (\exists r \in Y. q \xrightarrow{a_1} r) \wedge (\exists r \in Y. q \xrightarrow{a_2} r)\}$,
   $\{q \in X | (\nexists r \in Y. q \xrightarrow{a_1} r) \wedge (\exists r \in Y. q \xrightarrow{a_2} r)\}$,
   $\{q \in X | (\exists r \in Y. q \xrightarrow{a_1} r) \wedge (\nexists r \in Y. q \xrightarrow{a_2} r)\}$,
   $\{q \in X | (\nexists r \in Y. q \xrightarrow{a_1} r) \wedge (\nexists r \in Y. q \xrightarrow{a_2} r)\}$.

   Since $(split_{a_2,Y} \circ split_{a_1,Y})(\pi)$ also partitions X into the above classes, $X' \in (split_{a_2,Y} \circ split_{a_1,Y})(\pi)$. Thus $(split_{a_1,Y} \circ split_{a_2,Y})(\pi) \sqsubseteq (split_{a_2,Y} \circ split_{a_1,Y})(\pi)$.

   Similarly, we can show that $(split_{a_2,Y} \circ split_{a_1,Y})(\pi) \sqsubseteq (split_{a_1,Y} \circ split_{a_2,Y})(\pi)$.

3. For every $X' \in split_{a,Y}(\pi)$, either $X' = \Phi_{a,Y}(X)$ or $X' = X - \Phi_{a,Y}(X)$. Suppose $X' = \Phi_{a,Y}(X)$. For all $q \in X'$, there exists $r \in Y$ such that $q \xrightarrow{a} r$. Thus X' is a-stable with respect to Y. Suppose $X' = X - \Phi_{a,Y}(X)$. For all $q \in X'$, q can not lead to Y, *i.e.*, X' is a-stable with respect to Y. □

Let a_1, a_2,..., a_n be the elements of A. For each $Y \in \rho$, we define the operator $split_Y(\pi)$ as follows:

$$split_Y = split_{a_1,Y} \circ split_{a_2,Y} \circ \cdots \circ split_{a_n,Y}.$$

The operator $split_Y(\pi)$ splits each class X in π into several subclasses with the following property: for every subclass X_i in X, if there is a state q in X_i such that $q \rightsquigarrow Y$ by an action a, then for every q' in X_i, $q' \rightsquigarrow Y$ by an action a.

Proposition 4.4 *(properties of $split_Y(\pi)$)*

1. $split_Y(\pi) \sqsubseteq \pi$
2. $split_X \circ split_Y = split_Y \circ split_X$
3. *Each class in* $split_Y(\pi)$ *is stable with respect to* Y.

Proof. These can be proved using Proposition 4.3. □

Let Y_1, Y_2,..., Y_n be the equivalent classes in ρ. We define the operator $split_\rho(\pi)$ as follows:

$$split_\rho = split_{Y_1} \circ split_{Y_2} \circ \cdots \circ split_{Y_n}.$$

Proposition 4.5 *(properties of $split_\rho(\pi)$)*

1. $split_\rho(\pi) \sqsubseteq \pi$
2. $split_{\rho_1} \circ split_{\rho_2} = split_{\rho_2} \circ split_{\rho_1}$
3. $split_\rho(\pi)$ *is stable with respect to* ρ.

Proof. These can be proved using Proposition 4.4. □

Definition 4.3 *The operator $split(X, \rho)$ is defined as follows:*

$$split(X, \rho) = split_\rho(\{X\}).$$

This definition is valid since $\{X\}$ is a partial partition of Q. We note that every class in $split(X, \rho)$ is stable with respect to ρ from Proposition 4.5(3).

Definition 4.4 *The pre-condition $pre_\rho(X)$ and post-condition $post_\rho(X)$ are defined as follows:*

$$pre_\rho((n, Z)) = \bigcup_{a \in A} \{(n', Z') \in \rho | \exists (n', c, o, h, n) \in T. f(Z' \cap c, o, h) \cap Z \neq \emptyset \wedge a = events(o)\}$$

$$post_\rho((n, Z)) = \bigcup_{a \in A} \{(n', Z') \in \rho | \exists (n, c, o, h, n') \in T. f(Z \cap c, o, h) \cap Z' \neq \emptyset \wedge a = events(o)\}$$

Properties. Now, we can apply the algorithm SMB to CSM with the initial partition ρ_0 and the above three operators. Given a CSM $M = \langle V, N, n_0, I, \Sigma, T \rangle$, let ρ represent the final partition resulted from the algorithm SMB. Let $Acc(\rho) = S$ be the set of all accessible classes from some initial class in $[Q_0]_\rho$ and let $Acc(Q)$ be the set of all accessible states from some initial state in Q_0.

Proposition 4.6 *(Properties of the state minimization algorithm) [BFH90]*

1. *For any* $X \in Acc(\rho)$, $X \cap Acc(Q) \neq \emptyset$.
2. $Acc(Q) \subseteq \cup_{X \in Acc(\rho)} X$.

3. *For $q_1, q_2 \in Acc(Q)$ which are in the same partition, q_1 and q_2 are bisimilar iff q_1 and q_2 are in the same class in $Acc(\rho)$.*

In order to construct the minimal reachability graph of a given CSM, we need to find the greatest bisimulation, called ρ_{GB}, of the reachable states $Acc(Q)$, which also refines the initial partition ρ_0; that is, $\rho_{GB} \sqsubseteq \rho_0$. Then, the minimal reachability graph is S_M/ρ_{GB}. Our algorithm, however, gives the reachability graph $S_M/Acc(\rho)$. Notice that $Acc(\rho)$ may include some states not in $Acc(Q)$, that is, $S_M/Acc(\rho)$ may be different from the minimal reachability graph S_M/ρ_{GB}. Fortunately, they are isomorphic in the following sense: Two transition systems $S = \langle Q, A, \rightarrow, Q_0 \rangle$ and $S' = \langle Q', A, \rightarrow', Q_0' \rangle$ are *isomorphic* iff there exists a bijection f from Q to Q' such that

$$q \in Q_0 \Leftrightarrow f(q) \in Q_0' \quad \text{and} \quad q_1 \xrightarrow{a} q_2 \Leftrightarrow f(q_1) \xrightarrow{a}{}' f(q_2).$$

Theorem 4.1 *S_M/ρ_{GB} and $S_M/Acc(\rho)$ are isomorphic.*

Proof. For each $X \in \rho_{GB}$, $f(X)$ is defined as $[q]_\rho$ such that q is in X. For any two states q_1 and q_2 in $X \in \rho_{GB}$, q_1 and q_2 are bisimilar by the definition of ρ_{GB}. Then $[q_1]_\rho$ is equal to $[q_2]_\rho$ by Proposition 4.6 3. Therefore, f is a function. Suppose $f(X) = f(Y)$. Let q_1 and q_2 be states in X and Y, respectively. Then $[q_1]_\rho = [q_2]_\rho$ which means that q_1 and q_2 are bisimilar by Proposition 4.6 3. Thus $X = Y$, that is, f is one-to-one. And for $X' \in Acc(\rho)$, $[q]_{\rho_{GB}} \in \rho_{GB}$ such that $q \in Acc(Q) \cap X'$ is mapped to X' by f, that is, f is onto. Therefore, f is a bijection. First we prove that X is in $[Q_0]_{\rho_{GB}}$ iff $f(X)$ is in $[Q_0]_\rho$. ($\Rightarrow$) If X in $[Q_0]_{\rho_{GB}}$, then X has at least one initial state q_0 in Q_0. Thus $f(X) = [q_0]_\rho$ is a class in $[Q_0]_\rho$. ($\Leftarrow$) If $f(X)$ is in $[Q_0]_\rho$, then $f(X)$ has at least one initial state q_0 in Q_0. Thus $X = [q_0]_{\rho_{GB}}$ is a class in $[Q_0]_{\rho_{GB}}$. Secondly, We show that $X \xrightarrow{a}_{\rho_{GB}} Y$ iff $f(X) \xrightarrow{a}_\rho f(Y)$. ($\Rightarrow$) If $X \xrightarrow{a}_{\rho_{GB}} Y$ then there exists some $q \xrightarrow{a} q'$ such that $q \in X$ and $q' \in Y$. Thus $[q]_\rho \xrightarrow{a}_\rho [q']_\rho$, that is, $f(X) \xrightarrow{a}_\rho f(Y)$. ($\Leftarrow$) If $f(X) \xrightarrow{a}_\rho f(Y)$ then there exists some $q \in f(X)$ such that $q \in Acc(Q)$, that is, $q \in X$ by Proposition 4.6 1. And there exists some $q' \in f(Y)$ such that $q \xrightarrow{a} q'$. $q' \in Acc(Q)$ since $q \in Acc(Q)$, that is, $q' \in Y$. Thus $X \xrightarrow{a}_{\rho_{GB}} Y$. □

Thus, without loss of generality $S_M/Acc(\rho)$ is considered as the minimal reachability graph with respect to bisimulation that refines the initial partition ρ_0.

Sufficient Conditions. Since the state minimization algorithm does not guarantee termination, we now identify the sets of sufficient conditions on the syntax of CSMs which insure the termination as follows: 1) CSMs with finite data space. 2) CSMs with data-independent controls, that is, no enabling condition. 3) CSMs with the form of assignments $x := i$ where $i \in dom(x)$: Suppose that the numbers of nodes, enabling conditions and assignments of a CSM are j, k and l, respectively. The number of classes in the initial partition is j. The class is divided into at most two subclasses by an enabling condition. The effect of each assignment in split operations can be thought of as increasing the number of enabling conditions to at most $k \times 2^l$. Thus, the number of classes in the coarsest partition is at most $j \times 2^{k \times 2^l}$.

4.4 Example

Figure 4 shows a CSM with two data variables x and y whose domains are reals and whose initial values are any non-negative reals. Suppose $dom(e, 1)$ is the set of reals, that is, this CSM can receive any real value through event e. Here, τ represents an empty operation, *i.e.*, an internal action.

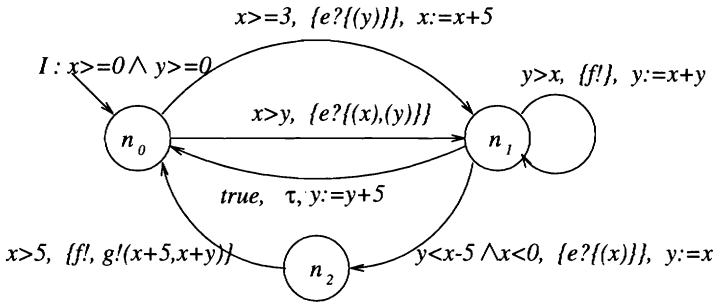


Figure 4: A Communicating State Machine

We present how to construct the minimal reachability graph from the CSM shown on Figure 4. The initial domain D^I is $(n_0, \{v_0 \in R^2 | v_0(x) \geq 0 \wedge v_0(y) \geq 0\})$, and the initial partition ρ is $\{C_0, C_1, C_2\}$, where $C_i = \{(n_i, R^2)\}$.

We start with the only reachable class C_0 since any initial state is in C_0. $split(C_0, \rho) = \{C_{00}, C_{01}\}$, where $C_{00} = (n_0, \{v | v(x) < 3 \wedge v(x) \leq v(y)\}$ and $C_{01} = (n_0, \{v | v(x) \geq 3 \vee v(x) > v(y)\}$. We get a new partition $\rho = \{C_{00}, C_{01}, C_1, C_2\}$.

C_{00} and C_{01} are all initial classes since they include initial states of the CSM, that is $C_{0i} \cap D^I \neq \emptyset$ for $i = 0$ or 1. Considering C_{00} and C_{01}, the two classes are all stable with respect to the current partition since the CSM in any state of C_{00} cannot proceed any more and the CSM in any state of C_{01} can proceed only to C_1 by executing the event e.

Class C_1 is included in the set of reachable classes. Considering $X = C_1$, $split(C_1, \rho) = \{C_{10}, C_{11}, C_{12}, C_{13}, C_{14}\}$, where

$C_{10} = (n_1, \{v | v(y) > v(x) \wedge v(x) \geq 3\})$,

$C_{11} = (n_1, \{v | v(y) > v(x) \wedge v(x) < 3\})$,

$C_{12} = (n_1, \{v | (v(y) \leq v(x) \wedge v(x) \geq 3) \vee (v(y) < v(x) - 5 \wedge v(x) \geq 0)\})$,

$C_{13} = (n_1, \{v | v(y) \leq v(x) \wedge v(y) \geq v(x) - 5 \wedge v(x) < 3\})$, and

$C_{14} = (n_1, \{v | v(y) < v(x) - 5 \wedge v(x) < 0\})$.

Now, the new partition is $\rho = \{C_{00}, C_{01}, C_{10}, C_{11}, C_{12}, C_{13}, C_{14}, C_2\}$.

Let's reconsider C_{01} since C_1 which is not stable is immediately reachable from C_{01}. $split(C_{01}, \rho) = \{C_{010}, C_{011}, C_{012}\}$, where

$C_{010} = (n_0, \{v | v(x) \geq 3 \wedge v(x) \leq v(y)\})$,

$C_{011} = (n_0, \{v | v(x) \geq 3 \wedge v(x) > v(y)\})$,

$C_{012} = (n_0, \{v | v(x) < 3 \wedge v(x) > v(y)\})$.

All of the new classes C_{010}, C_{011}, and C_{012} include some initial states of the CSM.

Since the CSM in every state of C_{010} may go to C_{10} or C_{12}, the CSM in every state of C_{011} may go to C_{10}, C_{12}, or C_{13}, and the CSM in every state of C_{012} may go to C_{12} or C_{13}, we know that these three classes C_{010}, C_{011}, C_{012} are all stable with respect to $\rho = \{C_{00}, C_{010}, C_{011}, C_{012}, C_{10}, C_{11}, C_{12}, C_{13}, C_{14}, C_2\}$. Thus, C_{10}, C_{12}, and C_{13} become reachable classes.

Since C_{10} may go to C_{10} or C_{010}, and C_{13} may go to C_{00}, these two classes C_{10} and C_{13} are stable. Now, C_{12} is divided into three subclasses:

$C_{120} = (n_1, \{v | v(y) \leq v(x) \wedge v(x) \geq 3 \wedge v(y) \geq v(x) - 5\})$,

$C_{121} = (n_1, \{v | v(y) < v(x) - 5 \wedge v(x) \geq 3\})$,

$C_{122} = (n_1, \{v | v(y) < v(x) - 5 \wedge (0 \leq v(x) < 3)\})$.

Let's consider C_{010}, C_{011}, and C_{012} again since C_{12} was immediately reachable from them. All of them are stable since C_{10}, C_{120}, and C_{121} are immediately reachable from every state in C_{010}; C_{10}, C_{120}, C_{121}, and C_{13} are immediately reachable from every state in C_{011}; C_{120} and C_{13} are immediately reachable from every state in C_{012}. Moreover, C_{120} is stable since it can lead to C_{010}, and C_{121} is stable since it can lead to C_{011}.

Therefore, all reachable classes $C_{00}, C_{010}, C_{011}, C_{012}, C_{10}, C_{120}, C_{121}$, and C_{13} from the initial classes $C_{00}, C_{010}, C_{011}, C_{012}$ are stable with respect to the current partition. Figure 5 shows the minimal reachable graph of the CSM in Figure 4.

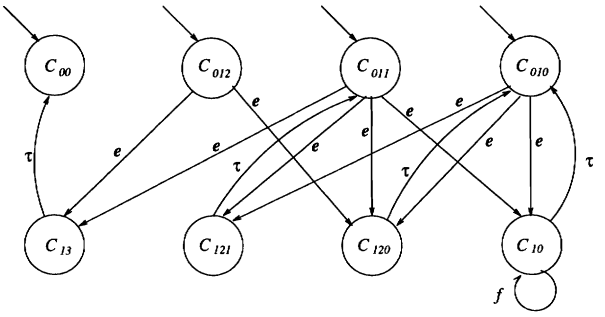


Figure 5: The Minimal Transition System

5 Conclusion

We have presented an algorithm to compute the minimal reachability graph with respect to bisimulation of a system described in CSM. This algorithm extends the algorithm in [BFH90] to a labeled transition system with infinitely many initial states. Our algorithm can deal with systems which have the infinite data domain without restricting systems into data-independent ones as done in [JP89]. We have identified some sufficient conditions on the syntax of CSMs that guarantee termination. We are currently investigating other sets of sufficient conditions.

Acknowledgments. The authors would like to thank Rance Cleaveland for pointing out related research and for clarifying research directions. The paper was improved by the comments made by Hanêne Ben Abdallah and Duncan Clarke.

References

[ACH+92] R. Alur, C. Courcoubetis, N. Halbwachs, D. Dill, and H. Wong-Toi. Minimization of Timed Transition Systems. In W.R. Cleaveland, editor, *Proceedings of International Conference on Concurrency Theory*, Lecture Notes in Computer Science vol. 630. Springer-Verlag, August 1992.

[BCMD90] J. R. Burch, E. M. Clarke, K. L. McMillan, and D. L. Dill. Sequential Circuit Verification using Symbolic Model Checking. In *Proceedings of Design Automation Conference*, 1990.

[BFH90] A. Bouajjani, J.-C. Fernandez, and N. Halbwachs. Minimal Model Generation. In *Proceedings of Workshop on Computer-Aided Verification*, 1990.

[BFHR92] A. Bouajjani, J.-C. Fernandez, N. Halbwachs, and P. Raymond. Minimal State Graph Generation. *Science of computer programming*, 18:247–269, 1992.

[CES86] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic Verification of Finite-state Concurrent Systems using Temporal Logic Specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, April 1986.

[CPS88] R. Cleaveland, J. Parrow, and B. Steffen. The Concurrency Workbench: Operating Instructions. Technical Note 10, Lab. for Foundations of CS, Univ. of Edinburgh, September 1988.

[Fer90] J.-C. Fernandez. An Implementation of an Efficient Algorithm for Bisimulation Equivalence. *Science of Computer Programming*, 13:219–236, 1990.

[Hop71] J.E. Hopcroft. An $n \log n$ Algorithm for Minimizing States in a Finite Automaton. In Z. Kohavi and A. Paz, editors, *Theory of Machines and Computations*, pages 189–196. Academic Press, 1971.

[JP89] B. Jonsson and J. Parrow. Deciding Bisimulation Equivalences for a Class of Non-finite-state Programs. Technical Report SICS/R-89/8908, Swedish Institute of Computer Science, August 1989.

[KS90] P. C. Kanellakis and S. A. Smolka. CCS Expressions, Finite State Processes, and Three Problems of Equivalence. *Information and Computation*, 86:43–68, 1990.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice Hall, 1989.

[PT87] R. Paige and R.E. Tarjan. Three Partition Refinement Algorithms. *SIAM J. Comput.*, 16(6), December 1987.

[Sha91] A. C. Shaw. Communicating Real-Time State Machines. Technical Report 91-08-09, Dept. of Computer Science and Engineering, Univ. of Washington, 1991.

[YY91] W. J. Yeh and M. Young. Compositional Reachability Analysis using Process Algebra. In *Proceedings of Conference on Testing, Analysis and Verification*, August 1991.