# THEORY AND ALGORITHMS FOR STATE MINIMIZATION OF NON-DETERMINISTIC FSM'S

by

Timothy Kam, Tiziano Villa, Robert K. Brayton,
and Alberto L. Sangiovanni-Vincentelli

Memorandum No. UCB/ERL M95/107

19 December 1995

# THEORY AND ALGORITHMS FOR STATE MINIMIZATION OF NON-DETERMINISTIC FSM'S

by

Timothy Kam, Tiziano Villa, Robert K. Brayton,
and Alberto L. Sangiovanni-Vincentelli

Memorandum No. UCB/ERL M95/107

19 December 1995

**ELECTRONICS RESEARCH LABORATORY**

College of Engineering
University of California, Berkeley
94720

# Theory and Algorithms for State Minimization of Non-Deterministic FSM's

Timothy Kam[1] Tiziano Villa[2] Robert K. Brayton[2]
Alberto L. Sangiovanni-Vincentelli[2]

[1]Intel Development Labs
Intel Corporation
Hillsboro, Oregon 97124-6497

[2]Department of EECS
University of California at Berkeley
Berkeley, CA 94720

December 19, 1995

**Abstract**

This paper addresses state minimization problems of different classes of non-deterministic finite state machines (NDFSM's). We describe a fully implicit algorithm for state minimization of pseudo non-deterministic FSM's (PNDFSM's). The results of our implementation are reported and shown to be superior to a previous explicit formulation. We could solve exactly all but one problem of a published benchmark, while an explicit program could complete approximately one half of the examples, and in those cases with longer run times. Then we present a theoretical solution to the problem of exact state minimization of general NDFSM's, based on the proposal of generalized compatibles. This gives an algorithmic frame to explore behaviors contained in a general NDFSM.

## 1 Introduction

Non-determinism is a valuable tool in the specification of behaviors. It can be used both to capture conditions that cannot arise in a certain situation (extending the usage of don't care conditions) and to postpone implementation choices. In a finite state machine (FSM) specification, it captures multiple choices in the behavior each of which could be chosen to be implemented. The most general form of non-determinism in sequential behaviors can be expressed by means of a non-deterministic FSM (NDFSM). The usual goal is to explore different behaviors contained within an FSM specification and choose an optimum one with respect to some cost function, e.g., one with a minimum number of states. In this paper we address the problem of finding a minimum contained behavior within a stand-alone FSM.

In the case of deterministic FSM's (DFSM's), the most efficient state minimization algorithm has complexity $O(n \log n)$, where $n$ is the number of states [6]. An implicit algorithm for computing equivalent states has been presented in [9]. An exact explicit algorithm for state minimization of incompletely specified FSM's (ISFSM's) has been proposed in [4], and implemented as a program in [10]. In [8], an implicit algorithm for exact state minimization of ISFSM's has been described. It is based on new implicit techniques to generate compatibles and to solve a binate table. Recently, a more general class of NDFSM's

(originally proposed by Cerny in [3]) called pseudo NDFSM's (PNDFSM's) has been shown by Watanabe in [11] to be sufficient to capture the flexibility of an FSM at a node in a network of interacting FSM's. Extracting out of an NDFSM a behavior corresponding to a DFSM with a minimum number of states is an important synthesis objective, that generalizes the problem of state minimization of ISFSM's.

In [12, 2] the problem of extracting a minimum state behavior out of a PNDFSM has been studied and it has been shown that an exact solution can be obtained by extending the notion of compatibles and formulating a binate covering problem. The algorithms in [12] use some implicit techniques from [8], and tackle also the more complex problem of selecting a DFSM that can be correctly implemented in an FSM network.

Here we present a two-fold contribution: (1) A theoretical solution to the problem of exact state minimization of general NDFSM's, based on a new notion of generalized compatibles. They are sufficient to explore behaviors contained in a general NDFSM. (2) An implicit algorithm for state minimization of PNDFSM's. The results of our implementation are reported and shown to be superior to the explicit formulation described in [12]. We could solve exactly all the problems of the benchmark used in [11] (except two cases, where minimal solutions not guaranteed to be minimum were found). The explicit program could complete approximately one half of the examples, and in those cases with longer running times.

It is worth to underline that the first step of exact state minimization is the exploration of all possible behaviors contained in a NDFSM. For some classes of NDFSM's this can be achieved by computing compatibles (as classically defined in [4] and then extended in [12, 2]). Each closed collection of compatibles is a contained DFSM and vice versa. In the case of state minimization, one wants a minimum cardinality closed collection of compatibles. But one can replace the requirement of minimum cardinality with any other desired cost function or property (such as an implementable behavior) and obtain a new problem of behavior selection. Therefore the exploration of all contained behaviors is a key technology for future applications in the synthesis of sequential networks and the capability of doing it efficiently as when using the proposed implicit techniques is a winning tool to support synthesis algorithms.

The remainder of the paper is organized as follows. Section 2 reviews implicit representations and manipulations. A taxonomy of different classes of finite state machines is proposed in Section 3 and their state minimization problem is introduced in Section 4. State minimization of PNDFSM's is discussed in Section 5, and a fully implicit algorithm for PNDFSM minimization is presented in Section 6. A theory for state minimization of NDFSM's is proposed in Section 7, while algorithms are discussed in Section 8. Results on minimization of PNDFSM's are reported in Section 9. Conclusions are summarized in Section 10.

## 2 Implicit Representations and Manipulations

We will use the unified implicit framework proposed in [8] [1]. Implicit techniques are based on the idea of operating on discrete sets by their characteristic functions represented by binary decision diagrams (BDD's) [1]. For example, the state transition relation of an FSM is represented by a BDD of its characteristic function.

To perform state minimization, one needs to represent and manipulate efficiently sets of sets of states. With $n$ states, each subset of states is represented in **positional-set** form, using a set of $n$ Boolean variables, $x = x_1 x_2 \ldots x_n$. The presence of a state $s_k$ in the set is denoted by the fact that variable $x_k$ takes the value 1 in the positional-set, whereas $x_k$ takes the value 0 if state $s_k$ is not a member of the set. For example, if $n = 6$, the set with a single state $s_4$ is represented by 000100 while the set of states $s_2 s_3 s_5$ is represented by

[1] $\exists x(\mathcal{F})$ ($\forall x(\mathcal{F})$) denotes the existential (universal) quantification of function $\mathcal{F}$ over variables $x$; $\Rightarrow$ denotes Boolean implication; $\Leftrightarrow$ denotes XNOR; $\neg$ denotes NOT.

011010.

A set of sets of states is represented as a set $S$ of positional-sets by a BDD characteristic function $\chi_S : B^n \rightarrow B$ as: $\chi_S(x) = 1$ if and only if the set of states represented by the positional-set $x$ is in the set $S$. A BDD representing $\chi_S(x)$ will contain minterms, each corresponding to a state set in $S$.

**Lemma 2.1** *Set* **equality, containment** *and* **strict-containment** *between two positional-sets $x$ and $y$ are expressed by: $(x = y) = \prod_{k=1}^{n}(x_k \Leftrightarrow y_k)$; $(x \supseteq y) = \prod_{k=1}^{n}(y_k \Rightarrow x_k)$; and $(x \supset y) = (x \supseteq y) \cdot (x \neq y)$.*

**Lemma 2.2** *Given two sets of positional-sets,* **complementation, union, intersection,** *and* **sharp** *can be performed on them as logical operations $(\neg, +, \cdot, \cdot\neg)$ on their characteristic functions.*

# 3 Taxonomy of Finite State Machines

In this section, we shall first define different classes of finite state machines (FSM's) used in this paper, and their state minimization problems. Then we shall introduce the two common steps of a state minimization algorithm: compatible generation and selection.

**Definition 3.1** *A* **deterministic FSM** *(DFSM) can be defined as a 6-tuple $M = \langle S, I, O, \delta, \lambda, r \rangle$. $S$ represents the finite state space, $I$ represents the finite input space and $O$ represents the finite output space. $\delta$ is the next state function defined as $\delta : I \times S \rightarrow S$ where $n \in S$ is the next state of present state $p \in S$ on input $i \in I$ if and only if $n = \delta(i, p)$. $\lambda$ is the output function defined as $\lambda : I \times S \rightarrow O$ where $o \in O$ is the output of present state $p \in S$ on input $i \in I$ if and only if $o = \lambda(i, p)$. $r \in S$ represents the unique reset state.*

**Definition 3.2** *A* **non-deterministic FSM** *(NDFSM) is defined as a 5-tuple $M = \langle S, I, O, T, R \rangle$ where $S, I, O$ are defined as above. $T$ is the transition relation defined as a characteristic function $T : I \times S \times S \times O \rightarrow B$. On an input $i$, the NDFSM at present state $p$ can transit to a next state $n$ and output $o$ if and only if $T(i, p, n, o) = 1$ (i.e., $(i, p, n, o)$ is a transition). There exists one or more transitions for each combination of present state $p$ and input $i$. $R \subseteq S$ represents the set of reset states.*

The above is the most general definition of an FSM and it contains, as special cases, different well-known classes of FSM's. To capture flexibility/don't-cares in the next state $n$ and/or the output $o$ from a state $p$ under an input $i$, one can specify one or more transitions $(i, p, n, o) \in T$. We assume that the state transition relation $T$ is complete with respect to $i$ and $p$, i.e., there is always at least one transition from each state on each input.

An NDFSM is a PNDFSM such that, for each triple $(i, p, o) \in I \times S \times O$, there is a unique state $n$ satisfying $T(i, p, n, o) = 1$. It is non-deterministic because for a given input and present state there may be more than one output; it is called pseudo non-deterministic because transitions carrying different outputs must go to different next states [2].

**Definition 3.3** *A* **pseudo non-deterministic FSM** *(PNDFSM) is a 6-tuple $M = \langle S, I, O, \delta, \Lambda, R \rangle$. $\delta$ is the next state function defined as $\delta : I \times S \times O \rightarrow S$ where each combination of input, present state and output is mapped to a unique next state. $\Lambda$ is the output relation defined by its characteristic function $\Lambda : I \times S \times O \rightarrow B$ where each combination of input and present state is related to one or more outputs. $R \subseteq S$ represents the set of reset states.*

Since the next state $n$ is unique for a given output $o$, present state $p$ and input $i$, it can be given by a next state function $n = \delta(i, p, o)$. Since the output is non-deterministic in general, it is represented by the relation $\Lambda$.

[2] The underlying finite automaton of a PNDFSM is deterministic.

**Definition 3.4** *Given a finite set of inputs $I$ and a finite set of outputs $O$, a* **trace** *between $I$ and $O$ is a pair of input and output sequences $(\sigma_i, \sigma_o)$ where $\sigma_i \in I^*$, $\sigma_o \in O^*$ and $|\sigma_i| = |\sigma_o|$.*

**Definition 3.5** *A* **trace set** *is a set of traces.*

**Definition 3.6** *An NDFSM $M = \langle S, I, O, T, R \rangle$* **realizes** *a trace set between $I$ and $O$ from state $s_0 \in S$, denoted by $\mathcal{L}(M|_{s_0})$ [3], if for every trace $(\{i_0, i_1, \ldots, i_j\}, \{o_0, o_1, \ldots, o_j\})$ in the trace set, there exists a state sequence $s_1, s_2, \ldots, s_{j+1}$ such that $\forall k : 0 \leq k \leq j, T(i_k, s_k, s_{k+1}, o_k) = 1$.*

The trace set realized by a deterministic FSM with inputs $I$ and outputs $O$ is called a behavior between the inputs $I$ and the outputs $O$. A formal definition follows.

**Definition 3.7** *Given a finite set of inputs $I$ and a finite set of outputs $O$, a* **behavior** *between $I$ and $O$ is a trace set, $\mathcal{B} = \{(\sigma_i, \sigma_o) \mid |\sigma_i| = |\sigma_o|\}$, which satisfies the following conditions:*

1. *Completeness:*
   *For an arbitrary sequence $\sigma_i$ on $I$, there exists a unique pair in $\mathcal{B}$ whose input sequence is equal to $\sigma_i$.*

2. *Regularity:*
   *There exists a DFSM $M = \langle S, I, O, \delta, \lambda, s_0 \rangle$ such that, for each $((i_0, \ldots, i_j), (o_1, \ldots, o_j)) \in \mathcal{B}$, there is a sequence of states $s_1, s_2, \ldots, s_{j+1}$ with the property that $s_{k+1} = \delta(i_k, s_k)$ and $o_k = \lambda(i_k, s_k)$ for every $k : 0 \leq k \leq j$.*

For each state in a DFSM, each input sequence corresponds to exactly one possible output sequence. Given a reset state, a DFSM realizes a unique input-output behavior. But given a behavior, there can be (possibly infinitely) many DFSM's that realize the same behavior. Thus, the mapping between behaviors and DFSM realizations is a one-to-many relation.

Any other kinds of FSM's, on the other hand, can represent a set of behaviors because by different choices of next states and/or outputs, more than one output sequence can be associated with an input sequence. Therefore, while a DFSM represents a single behavior, an NDFSM can be viewed as representing a set of behaviors. Each such behavior within its trace set is called a contained behavior of the NDFSM. Thus an NDFSM expresses handily flexibility in sequential synthesis. The choice of a particular behavior for implementation is based on some cost function such as the number of states.

# 4 State Minimization of FSM's

A specification represents a set of behaviors. The sets associated to different specifications can be compared by means of the notion of **behavioral containment**.

**Definition 4.1** *An NDFSM's $M = \langle S, I, O, T, R \rangle$* **behaviorally contains** *another NDFSM $M' = \langle S', I, O, T', R' \rangle$, denoted by $\mathcal{L}(M) \supseteq \mathcal{L}(M')$, if [4] for every $r' \in R'$, there exists $r \in R$ such that the trace set of $M$ from $r$ contains the trace set of $M'$ from $r'$. i.e.,*

$$\mathcal{L}(M) \supseteq \mathcal{L}(M') \text{ if and only if } \forall r' \in R' \; \exists r \in R \; \mathcal{L}(M|_r) \supseteq \mathcal{L}(M'|_{r'}).$$

[3] If the NDFSM $M$ is viewed as a NFA $A$ which alphabet is $\Sigma = I \times O$, the trace set of $M$ from a state $s_0$ corresponds to the language of $A$ from $s_0$, and both will be denoted by $\mathcal{L}(M|_{s_0})$.

[4] *cf.* classical definition for ISFSM minimization.

A criterion in the choice of a behavior is representability by a state transition graph with a minimum number of states. This gives rise to the problem of state minimization.

**Definition 4.2** *Given an NDFSM* $M = \langle S, I, O, T, R \rangle$, *the* **state minimization problem** *is to find a DFSM* $M' = \langle S', I, O, T', R' \rangle$ *such that*

1. $\mathcal{L}(M') \subseteq \mathcal{L}(M)$, *and*
2. $\forall M''$ *such that* $\mathcal{L}(M'') \subseteq \mathcal{L}(M)$, $|S'| \leq |S''|$. [5]

*Such a case is denoted by* $\mathcal{L}(M') \overset{min}{\subseteq} \mathcal{L}(M)$.

The state minimization problem defined above is very different from the minimization problem of nondeterministic finite automata described in classical automata textbooks [5]. Here we require a minimum state implementation which is behaviorally contained in the specification, while the classical problem require an NDFSM which represents the same set of behaviors as the original NDFSM but has the fewest number of states.

We are going to define next closed covers, since a way to explore all behaviors contained in a PNDFSM is by finding all closed covers in it.

**Definition 4.3** *Given an NDFSM* $M = \langle S, I, O, T, R \rangle$, *a set of state sets,* $\{c_1, c_2, \ldots, c_n\}$, *is a* **cover** *of* $M$ *if*[6] *there exists* $r \in R$ *and* $c_j : 1 \leq j \leq n$ *such that* $r \in c_j$.

**Definition 4.4** *Given an NDFSM* $M = \langle S, I, O, T, R \rangle$, *a set of state sets,* $K = \{c_1, c_2, \ldots, c_n\}$, *is* **closed** *in* $M$ *if for every* $i \in I$ *and* $c_j : 1 \leq j \leq n$, *there exists* $o \in O$ *and* $c_k : 1 \leq k \leq n$ *such that for each* $s \in c_j$, *there exists* $s' \in c_k$ *such that* $T(i, s, s', o) = 1$. *i.e.,*

$$\forall i \in I \; \forall c_j \in K \; \exists o \in O \; \exists c_k \in K \; \forall s \in c_j \; \exists s' \in c_k \; T(i, s, s', o) = 1$$

**Definition 4.5** *A set* $K$ *of state sets is called a* **closed cover** *for* $M = \langle S, I, O, T, R \rangle$ *if*

1. $K$ *is a cover of* $M$ , *and*
2. $K$ *is closed in* $M$.

**Definition 4.6** *Let* $M = \langle S, I, O, T, R \rangle$, *and* $K = \{c_1, c_2, \ldots, c_n\}$ *be a closed cover for* $M$ *where* $c_j \in 2^S$ *for* $1 \leq j \leq n$, *and* $M' = \langle S', I, O, T', R' \rangle$ *where* $S' = \{s_1, s_2, \ldots, s_n\}$.

$K$ **is represented by** $M'$ *if for every* $i \in I$ *and* $j : 1 \leq j \leq n$, *there exists* $k : 1 \leq k \leq n$ *and* $o \in O$ *such that, if* $T'(i, s_j, s_k, o) = 1$ *then* $\forall s \in c_j \; \exists s' \in c_k \; T(i, s, s', o) = 1$.

Note that this definition implies a one-to-one mapping of $K$ onto $S'$; in particular, $c_j \rightarrow s_j$ for $1 \leq j \leq n$. However, many different FSM's can represent a single closed cover.

[5] Given a set $S$, $|S|$ denotes the cardinality of the set.

[6] *cf.* classical definition for ISFSM minimization.

# 5 State Minimization of PNDFSM's

It has been proved in [7] that one can explore all behaviors contained in a PNDFSM by finding all closed covers of the PNDFSM.

**Theorem 5.1** *Let $M$ be a PNDFSM and $M'$ be a DFSM. $\mathcal{L}(M') \subseteq \mathcal{L}(M)$ if and only if there exists a closed cover for $M$ which is represented by $M'$.*

The following theorem, proved in [7], is a companion and an extension of Theorem 5.1. It proves the optimality of exact state minimization algorithms which find minimum closed covers.

**Theorem 5.2** *Let $M$ be a PNDFSM and $M'$ be a DFSM. $\mathcal{L}(M') \overset{min}{\subseteq} \mathcal{L}(M)$ if and only if there exists a minimum closed cover for $M$ which is represented by $M'$.*

By Theorem 5.2, the state minimization problem of PNDFSM's can be reduced to the problem of finding minimum closed covers. From what established so far, a closed cover might contain arbitrary sets of states. Once a minimum closed cover is found, a minimum state DFSM which represents the closed cover can be obtained easily, and this final step is traditionally called *mapping*. A brute force approach to find a minimum closed cover would be to enumerate sets of state sets, and test each one of them to see if it represents a closed cover according to Definition 4.3. Then one would pick a closed cover of minimum cardinality. One improves on this brute force approach by introducing the notion of compatibles.

In the previous brute force, each candidate closed cover is a subset of $2^S$ where $S$ is the state space of the PNDFSM. Actually we do not have to consider all subsets of $S$, but only those that can be elements of a closed cover. The definition of a closed cover requires that it contains subsets only of the following types.

**Definition 5.1** *A set of states is an* **output compatible** *if for every input, there is a corresponding output which can be produced by each state in the set.*

**Lemma 5.1** *Every element of a closed cover* [7] *is an output compatible.*

*Proof:* By Definition 4.4, for each set $c_j : 1 \leq j \leq n$ on every input $i \in I$, there is an output $o \in O$ that can be produced by each state in the set. Therefore $c_j$ is an output compatible. ■

The following definition says that states within a compatible can potentially be considered as a single state.

**Definition 5.2** *A set of states is a* **compatible** *if for each input sequence, there is a corresponding output sequence which can be produced by each state in the compatible.*

The following lemma is proved in [7].

**Lemma 5.2** *Every element of a closed cover is a compatible.*

Because of Lemma 5.2, an exact state minimization algorithm only needs to generate compatibles. The next step of an exact algorithm after compatible generation is to select a subset of compatibles that corresponds to a minimized machine. To satisfy behavioral containment, the selection of compatibles should be such that appropriate covering and closure conditions are met. The covering conditions guarantee that some selected compatible (i.e., some state in the minimized machine) corresponds to a reset state of the original machine. The closure conditions require that for each selected compatible, the compatibles implied by state transitions should also be selected. The state minimization problem reduces to one that selects

[7] This more restrictive lemma is also true: Every element of a closed set is an output compatible.

a minimum closed cover of compatibles. Instead of enumerating and testing all subset of compatibles, the selection is usually solved as a *binate covering problem*, where covering and closure conditions are expressed as binate clauses.

Explicit algorithms for exact state minimization of PNDFSM's have been proposed by Watanabe *et al.* in [12] and by Damiani in [2]. An algorithm for PNDFSM state minimization is more complicated than one for ISFSM state minimization [8] because the definition of compatibles and the closure conditions are more complex. In the rest of the section we present the key elements of an algorithm, with new results on the logical representation of closure conditions. They will be used in Section 6 to introduce the first fully implicit algorithm for exact state minimization of PNDFSM's.

The following example will be used throughout the text for illustrative purposes. **Example** $M_p$ in Figure 1 is a PNDFSM because on input 1, state $D$ either outputs 0 and goes to state $B$, or outputs 1 and goes to state $A$.

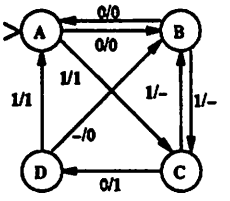


Figure 1: A PNDFSM, $M_p$.

## 5.1 Compatibles

The following theorem, proved in [7], serves as an equivalent, constructive definition of compatibles (*cf.* Definition 5.2). The theorem yields an implicit compatible generation procedure given in Section 6.1.

**Theorem 5.3** *A set c of states is a* **compatible** *if and only if for each input i, there exists an output o such that*

1. *each state in c has a transition under input i and output o, and*
2. *from the set c of states, the set $c'$ of next states under i and o is also a compatible.*

**Example** $A, B, C, D, AB$ are the compatibles of $M_p$ of Figure 1. $AB$ is a compatible because on input 0, it loops back to itself, and on input 1, it goes to $C$ which is also a compatible.

## 5.2 Covering and Closure Conditions

**Definition 5.3** *A set of compatibles* **covers** *the reset state(s) if at least one selected compatible contains a reset state.*

**Example** The covering condition for $M_p$ requires either compatible $A$ or compatible $AB$ to be selected. It can be expressed by the simple clause $(p_A + p_{AB})$ where the positive literal $p_A$ ($p_{AB}$ respectively) is true if and only if compatible $A$ ($AB$ respectively) is selected.

**Definition 5.4** *A set $C$ of compatibles is* **closed** *if for each compatible $c \in C$, for each input i, there exists an output o such that*

1. *each state in c has a transition under input i and output o, and*

2. *from the set c of states, the set d of next states under i and o is contained in a compatible in C .*

The following statement refines Theorem 5.2.

**Corollary 5.1** *The state minimization problem of a PNDFSM reduces to the problem of finding a minimum set of compatibles that covers the reset state(s) and is closed.*

**Definition 5.5** *Assuming a compatible c is selected, the* **closure condition** *for c, denoted by* $closure(c)$*, is a logic formula expressing the requirement that some other compatibles be selected according to Definition 5.4.*

**Example** Consider the closure condition for compatible $D$. On input 0, $D$ transits to $B$. For state $B$ to be in a selected compatible, we must select either compatible $B$ or $AB$. On input 1, $D$ either transits to $A$ with output 1, *or* transits to $B$ with output 0. We must select a compatible which contains either $A$ or $B$, i.e., we must select either compatible $A$ or $B$ or $AB$. Thus, the closure condition for $D$ is the conjunction of disjunctions $(p_B + p_{AB}) \cdot (p_A + p_B + p_{AB})$. Similarly, the closure condition for $A$ is $(p_B + p_{AB}) \cdot (p_C)$, for $B$ is $(p_A + p_{AB}) \cdot (p_C)$, for $C$ is $(p_D) \cdot (p_B + p_{AB})$ and for compatible $AB$ is $(p_C)$.

During binate covering, the covering table should guarantee that for each compatible $c$, either $c$ is not selected (i.e., $\overline{p_c}$ is true), or its closure condition $closure(c)$ is satisfied. In other words, $\overline{p_c} + closure(c)$. This expression can be represented as a conjunction of binate clauses as discussed in more detail later.
**Example** For compatible $D$, the binate covering table should have clauses expressing $\overline{p_D} + closure(D) = \overline{p_D} + (p_B + p_{AB}) \cdot (p_A + p_B + p_{AB}) = (\overline{p_D} + p_B + p_{AB}) \cdot (\overline{p_D} + p_A + p_B + p_{AB})$, which can be simplified to $(\overline{p_D} + p_A + p_B + p_{AB})$.

## 5.3 Prime Compatibles

As in the case of ISFMS's, it is sufficient to consider a subset of compatibles, called prime compatibles.

**Definition 5.6** *A compatible c'* **prime dominates** *a compatible c if for each minimum closed cover containing c, the selection with c replaced by c' also corresponds to a minimum closed cover.*

**Definition 5.7** *A compatible c is a* **prime compatible** *if there does not exist another compatible c' such that c' prime dominates c.*

**Theorem 5.4** *There exists a minimum closed cover made up entirely of prime compatibles.*

The above theorem, proved in [7], justifies that prime compatibles are sufficient to find a minimum solution.

A sufficient condition for prime dominance is given by the following theorem.

**Theorem 5.5** *A compatible c'* **prime dominates** *a compatible c if*

1. *if* $(c \cap R) \neq \emptyset$ *then* $(c' \cap R) \neq \emptyset$*, and*
2. *the closure condition for c implies* [8] *the closure condition for c', and*
3. $c' \supset c$.

[8] Condition A implies condition B if and only if the satisfaction of condition A automatically guarantees the satisfaction of condition B. In other words, A is *not less restrictive* than B.

*Proof:* Assume by contradiction that $c'$ does not prime dominate $c$, i.e., there is a minimum closed cover containing $c$ which is not any more a closed cover when $c$ is replaced by $c'$ (Definition 5.7). We show that at least one of the above three conditions is false.

Consider any set of compatibles $C$ such that $C \cup \{c\}$ is a minimum closed cover [9]. As $C \cup \{c\}$ and $C \cup \{c'\}$ have the same cardinality, in order that $C \cup \{c'\}$ is not a minimum closed cover, either (1) $C \cup \{c'\}$ does not cover the reset state(s) or (2) $C \cup \{c'\}$ is not closed. For case (1), $C \cup \{c\}$ is a cover but $C \cup \{c'\}$ is not a cover if and only if $(c \cap R) \neq \emptyset$ and $(c' \cap R) = \emptyset$, i.e., condition 1 of the above theorem is false. For case (2), $C \cup \{c\}$ is closed but $C \cup \{c'\}$ is not closed if one of the two situations arises: (2a) $C$ satisfies the closure condition for $c$ but not the closure condition for $c'$. This happens if and only if condition 2 is false. (2b) $c$ is needed to satisfy the closure condition for some compatible in $C$, but $c'$ does not satisfy such a condition. This is the case only if $c' \not\supset c$, i.e., condition 3 above is false. ■

The converse of the theorem is not true in general, because condition 3 is a sufficient condition, but not a necessary condition, for case (2b) above.

**Example** Compatible $AB$ prime dominates compatible $B$ because all conditions of Theorem 5.5 are met. In particular, closure condition for $B$ implies closure condition for $AB$ because $[(p_A + p_{AB}) \cdot (p_C)] \Rightarrow [p_C]$. Similarly, $AB$ dominates $A$. As a result, the prime compatibles are $AB, C, D$.

## 5.4 Logical Representation of Closure Conditions

We now construct a set of logical clauses expressing the closure requirement that a next state set $d$ is contained in at least one selected compatible, as stated in Definition 5.4. Since we will generate the set of prime compatibles, we express also with logical clauses part 2 of Theorem 5.5 that refers to the implication between closure conditions.

### 5.4.1 Computation of Closure Conditions

The notion of next state sets $d$ is important for expressing closure conditions and testing prime dominance.

**Definition 5.8** *$d_{c,i,o}$ is the set of next states from compatible $c$ under input $i$ and output $o$.*

Given a triple $(c, i, o)$, the set $d_{c,i,o}$ is unique in a PNDFSM. We associate to each $d_{c,i,o}$ a clause whose positive literals are the prime compatibles that contain $d_{c,i,o}$. This clause will be part of the binate clause representing the closure condition for compatible $c$. For simplicity in notation, we designate by $d_{c,i,o}$ both the set of next states and the clause associated to it. It will be clear from the context which one it is meant.

**Example** The next state set from $D$ on input 0 and output 0, $d_{D,0,0}$ is $B$ and it corresponds to the clause $(p_B + p_{AB})$. $d_{D,1,1}$ is $A$ and it corresponds to the clause $(p_A + p_{AB})$. $d_{D,1,0}$ is $B$ and it corresponds to the clause $(p_B + p_{AB})$.

For a PNDFSM, a set of compatible states $c$ under an input $i$ may go to different sets of next states depending on the choice of output $o$. For at least one choice of $o$, the corresponding next state set $d_{c,i,o}$ must be contained in some selected compatible. This is expressed by the disjunctive clause (or disjunction), $disjunct(c, i)$, defined as:

$$disjunct(c, i) = \exists o \in \text{outputs at } c \text{ under } i,\ d_{c,i,o}.$$

**Example** $disjunct(D, 1)$ represents the clause $(d_{D,1,0} + d_{D,1,1}) = (p_A + p_B + p_{AB})$.

For a PNDFSM, the closure condition for a compatible $c$, denoted by $closure(c)$, has the form of a conjunction of disjunctive clauses. According to Definition 5.4, the conjunction is over all inputs $i$, while

[9] It is possible that no such $C$ exists, and the theorem is trivially true.

the disjunction is over the outputs $o$ such that $\Lambda(i, s, o) = 1$. Given a compatible $c$, the following product of disjunctions must be satisfied (one disjunction per input):

$$closure(c) = \forall i \in \text{inputs}, disjunct(c, i).$$

In summary, the closure condition for compatible $c$ is fulfilled if and only if for each input $i$, there is an output $o$ such that the next state set $d_{c,i,o}$ from compatible $c$ under input $i$ and output $o$ is contained in a selected compatible. In logical terms, the closure condition for $c$ is fulfilled if and only if the following product-of-sums is satisfied:

$$closure(c) = \forall i \in \text{inputs } \exists o \in \text{outputs at } c \text{ under } i, d_{c,i,o} \quad (1)$$

These closure conditions are tested against a certain selection of compatibles.
**Example** Closure condition for compatible $D$ is $closure(D) = (d_{D,0,0} + d_{D,0,1}) \cdot (d_{D,1,0} + d_{D,1,1}) = (p_B + p_{AB}) \cdot (p_A + p_B + p_{AB})$.

### 5.4.2 Implication of Closure Conditions

We construct now the logical clauses expressing the prime dominance condition of Theorem 5.5. Part 1 and 3 are already expressed as simple logic formulas. Notice that the implication between closure conditions mentioned in part 2 of Theorem 5.5 translates exactly to logical implication ($\Rightarrow$) between the clauses of *closure*, i.e., given compatibles $c$ and $c'$, the closure condition for $c$ implies the closure condition for $c'$ if and only if $closure(c) \Rightarrow closure(c')$. It is not convenient to test this implication by first evaluating each closure condition according to Equation 1, because each closure condition is in a product of sums form. We would like to express it in terms of set containment between next state sets, building on top of the related result that will be proved in Theorem 5.8. What follows gives such a useful characterization of the formula $closure(c) \Rightarrow closure(c')$.

Using some fundamental validities of logic, we first prove two useful lemmas for manipulating logical clauses.

**Lemma 5.3** $[\forall x\ F(x)] \Rightarrow [\forall x'\ F'(x')]$ *if and only if* $\forall x'\ \exists x\ [F(x) \Rightarrow F'(x')]$.

*Proof:* $[\forall x\ F(x)] \Rightarrow [\forall x'\ F'(x')]$ iff $\forall x'\ [\forall x\ F(x) \Rightarrow F'(x')]$ iff $\forall x'\ \exists x\ [F(x) \Rightarrow F'(x')]$. ■

**Lemma 5.4** $[\exists x'\ F'(x')] \Rightarrow [\exists x\ F(x)]$ *if and only if* $\forall x'\ \exists x\ [F'(x') \Rightarrow F(x)]$.

*Proof:* $[\exists x'\ F'(x')] \Rightarrow [\exists x\ F(x)]$ iff $\forall x'\ [F'(x') \Rightarrow \exists x\ F(x)]$ iff $\forall x'\ \exists x\ [F'(x') \Rightarrow F(x)]$. ■

**Theorem 5.6** *Given compatibles $c$ and $c'$, $closure(c) \Rightarrow closure(c')$ iff $\forall i' \exists i\ [disjunct(c, i) \Rightarrow disjunct(c', i')]$.*

*Proof:* Substituting $x = i$, $x' = i'$, $F(x) = disjunct(c, i)$, $F'(x') = disjunct(c', i')$ into Lemma 5.3, one gets: $\forall i\ disjunct(c, i) \Rightarrow \forall i'\ disjunct(c', i')$ iff $\forall i'\ \exists i\ [disjunct(c, i) \Rightarrow disjunct(c', i')]$. The former is by definition $closure(c) \Rightarrow closure(c')$. ■

Now that we have expressed implication between closure conditions in terms of implication between disjunctive clauses, the following theorem gives a useful characterization of the formula $disjunct(c', i') \Rightarrow disjunct(c, i)$.

**Theorem 5.7** *Given compatibles $c'$, $c$ and inputs $i'$, $i$,*

$$disjunct(c', i') \Rightarrow disjunct(c, i) \text{ iff } \forall o'\ \exists o\ [d_{c',i',o'} \Rightarrow d_{c,i,o}].$$

*Proof:* Substituting $x = o, x' = o', F(x) = d_{c,i,o}, F'(x) = d_{c',i',o'}$ into Lemma 5.4, one gets: $\exists o'\ d_{c',i',o'} \Rightarrow \exists o\ d_{c,i,o}$ iff $\forall o'\ \exists o\ [d_{c',i',o'} \Rightarrow d_{c,i,o}]$. The former is by definition $disjunct(c', i') \Rightarrow disjunct(c, i)$. ■

The following property is key to evaluating implications between logical clauses by set containment. The latter can be performed implicitly because $d_{c,i,o}$ is represented as a positional-set in Section 6.2.

**Theorem 5.8** *If the set of next states $d_{c',i',o'} \supseteq$ the set of next state set $d_{c,i,o}$, then clause $d_{c',i',o'} \Rightarrow$ clause $d_{c,i,o}$.*

*Proof:* If the set of next states $d_{c',i',o'} \supseteq$ the set of next states $d_{c,i,o}$, then each prime compatible that contains $d_{c',i',o'}$ contains also $d_{c,i,o}$. Since each literal in a clause $d$ is a prime compatible that contains the next state set $d$, it means that the clause $d_{c,i,o}$ has all the literals of the clause $d_{c',i',o'}$ and so each assignment of literals that satisfies the clause $d_{c',i',o'}$ satisfies also the clause $d_{c,i,o}$, i.e., clause $d_{c',i',o'} \Rightarrow$ clause $d_{c,i,o}$. ■

The converse does not hold, as shown by the following counter-example. let $d(c', i', o')$ be $AB$ with the corresponding clause $(p_{ABCE} + p_{ABDE})$, and let $d(c, i, o)$ be $AE$ with the corresponding clause $(p_{ABCE} + p_{ABDE} + p_{AEG})$, then $(p_{ABCE} + p_{ABDE}) \Rightarrow (p_{ABCE} + p_{ABDE} + p_{AEG})$, but $AB \not\supseteq AE$.

By substituting Theorem 5.7 into Theorem 5.6 and using Theorem 5.8, we have expressed the implication between closure conditions of two compatibles (i.e., part 2 of Theorem 5.5) in terms of a logic formula on the next state sets from the two compatibles.

**Theorem 5.9** *If $\forall i'\ \exists i\ \forall o'\ \exists o\ (d_{c',i',o'} \supseteq d_{c,i,o})$, then $closure(c) \Rightarrow closure(c')$.*

*Proof:* By Theorems 5.7, 5.6 and 5.8. ■

### 5.4.3 Simplification of Closure Conditions

The following two theorems simplify the closure conditions. In our implicit algorithm, they are applied before the implication between the conditions is computed.

**Theorem 5.10** *Given a compatible $c$ and inputs $i'$ and $i$, if $disjunct(c, i') \Rightarrow disjunct(c, i)$, then $disjunct(c, i)$ can be omitted from the conjunction $closure(c)$ because of the existence of $disjunct(c, i')$.*

*Proof:* If $disjunct(c, i') \Rightarrow disjunct(c, i)$, the conjunction of $disjunct(c, i')$ and $disjunct(c, i)$ is simply $disjunct(c, i')$. Therefore $disjunct(c, i)$ can be omitted from the conjunction $closure(c)$. ■

**Theorem 5.11** *A set of next states $d$ is not needed to be part of the clause $disjunct(c, i)$, if*

1. *$d$ is a singleton reset state* [10], *or*
2. *$d \subseteq c$, or*
3. *$d \supseteq d'$ if $d'$ is part of $disjunct(c, i)$.*

*Proof:* (1) If the clause $disjunct(c, i)$ contains (the clause corresponding to) $d$ and if $d$ is a singleton reset state, then the covering condition implies $disjunct(c, i)$. (2) If the clause $disjunct(c, i)$ contains (the clause corresponding to) $d$ and if $d \subseteq c$, then (the literal corresponding to) $c$ must be in $disjunct(c, i)$, so that if $c$ is selected then $disjunct(c, i)$ is satisfied. (3) $d \supseteq d'$ by Theorem 5.8 means $d \Rightarrow d'$. The disjunction of $d$ and $d'$ is simply $d'$, so $d$ can be omitted from $disjunct(c, i)$. ■

The order in which the next state sets are pruned in $disjunct(c, i)$ is important, especially if these pruning rules are executed implicitly (in a sense, simultaneously). For proper removal of next state sets, one should find all the $d$'s that satisfy condition 1 or 2 first and remove them from $disjunct(c, i)$. Then on a separate step one removes all the $d$'s containing other $d'$ according to condition 3.

[10] Condition 1 is valid only if a unique reset state is specified.

# 6 Implicit State Minimization Algorithm for PNDFSM's

In this section, we will show how the state minimization algorithm described in Section 5 can be implicitized.

First, we outline the differences between the state minimization algorithm of PNDFSM's and the state minimization algorithm for ISFSM's [8]. For the latter, a set of states is a compatible if and only if each pair of states in it are compatible. This is not true for PNDFSM's and is illustrated by the following counter-example. As a result, the set of compatibles cannot be generated from the set of incompatible pairs as in [8]. In addition, the computations for closure conditions as well as prime dominance are more complicated than those for ISFSM's.

**Example** The following PNDFSM has three states $A$, $B$, and $C$, no input and an output with three values $\{x, y, z\}$. All state pairs are compatibles but the set $ABC$ is not a compatible because they cannot all agree on an output in one transition.

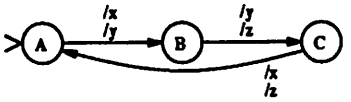


Figure 2: A PNDFSM which doesn't have a compatible ABC.

## 6.1 Implicit Generation of Compatibles

As we cannot generate compatibles from incompatible pairs, we have to start with output compatibles (i.e., state sets) of arbitrary cardinalities. First we compute the transition relation $T^{det}$ between sets of states, using the implicit procedure described in Section 8.2.

Given the transition relation $T(i, s, s', o)$ of a PNDFSM $M = \langle S, I, O, T, R\rangle$, first we compute the relation $T^0(i, c, c', o)$. A 4-tuple $(i, c, c', o)$ is in relation $T^0$ if and only if the set of states $c$ on input $i$ can transit to another set of states $c'$, and produce output $o$: $T^0(i, c, c', o) = \forall s\ \{[Single(s) \cdot (s \subseteq c)] \Rightarrow \exists s'\ [T(i, s, s', o) \cdot (s' \subseteq c')]\} \cdot \forall s'\ \{[Single(s') \cdot (s' \subseteq c')] \Rightarrow \exists s\ [T(i, s, s', o) \cdot (s \subseteq c)]\} \cdot \neg\emptyset(c) \cdot \neg\emptyset(c')$.

**Proposition 6.1** *The set $C$ of compatibles of a PNDFSM can be found by the following fixed point computation:*

- $\tau_0(i, c, c') = \exists o\ T^0(i, c, c', o)$,
- *Initially all subsets of states are compatible:* $C_0(c) = 1$,
- *By Theorem 5.3,* $\tau_{k+1}(i, c, c') = \tau_k(i, c, c') \cdot C_k(c')$, $C_{k+1}(c) = \forall i\ \exists c'\ \tau_{k+1}(i, c, c')$.

*The iteration can terminate when for some $j$, $C_{j+1} = C_j$, and the greatest fixed point has been reached. The set of compatibles is given by $C(c) = C_j(c)$ and the transition relation on the compatibles is $\tau(i, c, c') = \tau_{j+1}(i, c, c') \cdot C_j(c)$.*

## 6.2 Implicit Generation of Prime Compatibles and Closure Conditions

In our implicit framework, we represent each next state set as a positional-set $d$. The fact that a next state set $d$ is part of $disjunct(c, i)$ can be expressed by the transition relation on compatibles, $\tau(i, c, d)$. The following computation will prune away next state sets $d$ that are not necessary according to Theorem 5.11, and the result is represented by the following relation $B$.

**Definition 6.1** *The ITE operator returns function $G_1$ if function $F$ evaluates true, else it returns function $G_2$:*

$$ITE(F, G_1, G_0) = \begin{cases} G_1 & \text{if } F = 1 \\ G_0 & \text{otherwise} \end{cases}$$

*where range(F)={0,1}.*

**Theorem 6.1** *The disjunctive conditions can be computed by the following relation B:*

$$A(c, i, d) = ITE(\exists d\,\{\tau(i, c, d) \cdot [R(d) + (d \subseteq c)]\}, \emptyset(d), \tau(i, c, d))$$

$$B(c, i, d) = Minimal_d(A(c, i, d))$$

*Proof:* The first equation corresponds to conditions 1 and 2 of Theorem 5.11. Given a compatible $c$ and an input $i$, *if* there exists a $d$ which is a next state set from $c$ under $i$ such that $R(d) + (d \subseteq c)$ is true, *then* the $disjunct(c, i)$ is set to the empty set $\emptyset(d)$, *else* we keep the original $d$ in the relation $\tau(i, c, d)$. The second equation tests condition 3 and prunes all the $d$'s that are not minimal (i.e., containing some other $d'$ that is part of $disjunct(c, i)$). ■

In summary $\tau(i, c, d)$ represents the set of disjunctive clauses, while $B(c, i, d)$ represents the pruned set of disjunctive clauses: $d$ is in the relation $B$ with $(c, i)$ if and only if $d$ is part of the disjunctive clause for $c$ under $i$ after pruning.

The following theorem computes the set of disjunctive clauses according to Theorems 5.7 and 5.8, that are used to express the closure conditions. Then the set of prime compatibles is computed according to Theorem 5.5.

**Theorem 6.2** *If* $D(c', i', c, i) = \forall d'\ \{B(c', i', d') \Rightarrow \exists d\ [B(c, i, d) \cdot (d' \supseteq d)]\}$, *then* $disjunct(c', i') \Rightarrow disjunct(c, i)$.

*The set of prime compatibles can be computed by:*

$$PC(c) = C(c) \cdot \not\exists c'\{C(c') \cdot [\exists s\,(R(s) \cdot (s \subseteq c)) \Rightarrow \exists s'\,(R(s') \cdot (s' \subseteq c'))] \cdot \forall i' \exists i\,(D(c, i, c', i')) \cdot (c' \supset c)\}.$$

*Proof:* To evaluate $disjunct(c', i') \Rightarrow disjunct(c, i)$, by Theorems 5.7 and 5.8, it is sufficient to check: $\forall o' \in$ outputs at $c'$ under $i'$ $\exists o \in$ outputs at $c$ under $i$ $[d_{c',i',o'} \supseteq d_{c,i,o}]$.

In other words, we want to check that for all next state sets $d'$ from compatible $c'$ on input $i'$ (on some output $o'$), there exists a next state set $d$ from compatible $c$ on input $i$ (on some output $o$) such that $d'$ contains $d$. This corresponds to the condition

$$\forall d'\ \{B(c', i', d') \Rightarrow \exists d\ [B(c, i, d) \cdot (d' \supseteq d)]\}.$$

Therefore $D$ is a sufficient condition for $disjunct(c', i') \Rightarrow disjunct(c, i)$.

The second equation defines $PC(c)$ as the set of non-dominated primes. The right sub-formula within $\{\}$ expresses the three conditions in Theorem 5.5. A positional-set $c$ has a non-empty intersection with the set of reset states $R$ if and only if there exists a reset state $\exists s\ R(s)$ such that $s \subseteq c$. Condition 1 is satisfied, because $\exists s\ (R(s) \cdot (s \subseteq c)) \Rightarrow \exists s'\ (R(s') \cdot (s' \subseteq c'))$ makes sure that $(c' \cap R) \neq \emptyset$ if $(c \cap R) \neq \emptyset$. By Theorem 5.6, condition 2 of Theorem 5.5 is checked by $\forall i' \exists i\ D(c', i', c, i)$ according to the first part of this theorem. Condition (3) is simply $(c' \supset c)$. ■

The following theorem computes the pruned set of disjunctive clauses according to Theorems 5.10 and 6.2. This set will be used in the next subsection to set up the binate rows of the covering table.

**Theorem 6.3** *The pairs of compatibles c and inputs i involved in non-trivial disjunctive clauses are expressed by the following relation:*

$$E(c,i) = \nexists i' \left[(i \neq i') \cdot D(c,i',c,i)\right] + \nexists i' \left[(i' \prec i) \cdot D(c,i',c,i) \cdot D(c,i,c,i')\right].$$

*And the corresponding pruned set of disjunctive clauses is given by relation:* $I(c,i,d) = B(c,i,d) \cdot PC(c) \cdot E(c,i) \cdot \neg\emptyset(d)$.

*Proof:* Given a compatible $c$, Theorem 5.10 states that if $disjunct(c,i') \Rightarrow disjunct(c,i)$ then $disjunct(c,i)$ can be omitted from $closure(c)$. And $disjunct(c,i') \Rightarrow disjunct(c,i)$ if the two pairs are in relation $D(c,i',c,i)$, according to Theorem 6.2. The first term $\nexists i' \left[(i \neq i') \cdot D(c,i',c,i)\right]$ deletes all pairs $(c,i)$ such that there is an input $i'$ where $(i' \neq i)$ such that $disjunct(c,i') \Rightarrow disjunct(c,i)$. But this would eliminate two many $(c,i)$ pairs because it is possible that $(i' \neq i)$, and moreover $disjunct(c,i') \Rightarrow disjunct(c,i)$ and $disjunct(c,i) \Rightarrow disjunct(c,i')$ are both true. Such pairs are defined by $[D(c,i',c,i) \cdot D(c,i,c,i')]$. In such a case, we must choose and retain exactly one of the two, in particular we add back the pair $(c,i)$ in which $i$ has the smallest binary interpretation, by the last term of the first equation.

For the second equation, the pruned set of disjunctive clauses contains the clauses in $B(c,i,d)$, constrained to have compatibles $c$ that are primes in $PC(c)$, and pairs $(c,i)$ given by relation $E$. Also, triples with empty set $d$ are vacuously true clauses, and thus are pruned away. ■

## 6.3 Implicit Binate Table Covering

Selection of prime compatibles is performed by the implicit binate covering solver in [8]. In particular, we use the binate table solver which assumes each row has at most one 0. To use the solver, one has to specify four BDD's: two characteristic functions $Col$ and $Row$ representing a set of column labels and a set of row labels respectively; and two binary relations 1 and 0, one relating columns and rows that intersect at a 1 in the table, and another relating columns and rows that intersect at a 0.

Similar to the case for ISFSM's, each prime compatible corresponds to a single column labeled $p$ in the covering table. So the set of column labels, $Col(p)$, is given by: $Col(p) = PC(p)$.

Each row can be labeled by a pair $(c,i)$ because each binate clause originates from the closure condition for a compatible $c \in PC$ under an input $i$. And the covering condition for a reset state is expressed by a single unate clause, to which we assign a row label $(c,i) = (\emptyset,\emptyset)$. $c$ is chosen to be the empty set to avoid conflicts with the labels of the binate rows, while the choice of $i = \emptyset$ is arbitrary. The set of row labels, $Row(c,i)$, is given by a binate part and a unate part:

$$Row(c,i) = \exists d\, I(c,i,d) + \emptyset(c) \cdot \emptyset(i).$$

Each binate clause associated with a compatible $c$ and an input $i$ expresses the condition that for at least one output $o$, the next state set must be contained in a selected compatible $d$. The corresponding next state relation is $I(c,i,d)$.

Next, let us consider the table entries relations $1(c,i,p)$ and $0(c,i,p)$. If $(c,i)$ labels a binate row, the expression $\exists d\, [(p \supseteq d) \cdot I(c,i,d)]$ evaluates to true if and only if the table entry is a 1 at the intersection of the row labeled $(c,i)$ and the column labeled $p$, i.e., the row can be satisfied if next state set $d$ is contained in selected compatible $p$. There is an entry 0 at column $p$ if $(p = c)$, i.e., the row can also be satisfied by not selecting a column labeled $c$.

The row labeled by $(\emptyset,\emptyset)$ represents the disjunction of compatibles $p$ each of which contains at least a reset state $R(s)$. On such a row, a table entry is a 1 if and only if $\exists s\, [\emptyset(c) \cdot \emptyset(i) \cdot R(s) \cdot (s \subseteq p)]$.

As a summary, the inference rules for table entries given a row $(c,i)$ and a column $p$ are: $0(c,i,p) \stackrel{\text{def}}{=} (p = c)$,
$1(c,i,p) \stackrel{\text{def}}{=} \exists d\, [(p \supseteq d) \cdot I(c,i,d)] + \exists s\, [\emptyset(c) \cdot \emptyset(i) \cdot R(s) \cdot (s \subseteq p)]$.

# 7 State Minimization of NDFSM's

Is it possible to apply the classical procedure based on computing compatibles to NDFSM's of the most general kind, i.e., that are not PNDFSM's ? The answer is: yes, the notions of compatibles and selection of a minimum subset carry through to NDFSM's; but, no, that procedure is not guaranteed to produce a behavior with a minimum number of states. There may exist DFSM's behaviorally contained in an NDFSM that do not correspond to a closed cover.

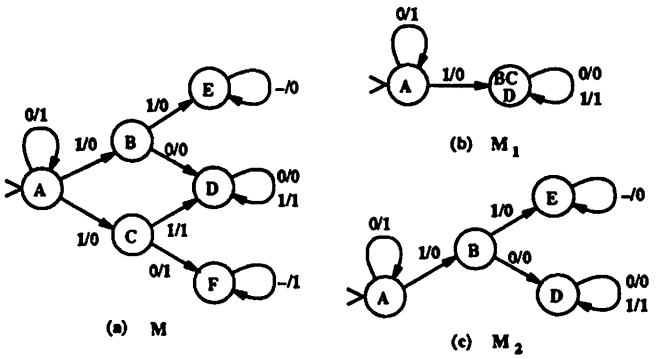


Figure 3: A counter example, a) the NDFSM $M$, b) the minimum state DFSM contained in $M$, c) one DFSM contained in $M$ found using compatibles.

**Example** Given the NDFSM $M$ in Figure 3a, the minimum state DFSM $M_1$ (in Figure 3b) contained in $M$ cannot be found using compatibles alone. According to Definition 5.2, states B and C are not compatible (and any state set containing B and C cannot be a closed set). Any minimized machine $M_2$ obtained by compatible-based algorithms has at least four states (e.g., Figure 3c). Suppose that after exiting non-deterministically from state A into states B and C, one chooses $0/0, 1/1$ as their outgoing transitions, as shown in Figure 3b. In this way B and C can be "merged" together as one state. This merged state is compatible with state D. This merging possibility is not explored by compatibility. The minimum state DFSM $M_1$, whose behavior is contained in the original NDFSM, has only two states.

## 7.1 Generalized Compatibles

We generalize the notion of a compatible (Definition 5.2) which is a set of states, to one that is a set of state sets. Each individual state set contains states that can be merged because they can be non-deterministically reached from a reset state. The state set can be considered a new merged state in a reduced machine which on each input, has a transition copied from a state in the set. If the original FSM does not have non-deterministic transitions, each such state set will be a singleton and each generalized compatible is just a classical compatible.

**Definition 7.1** *A set of state sets is a* **generalized compatible** *if for each input sequence, there is a corresponding output sequence which can be produced by at least one state from each state set in the generalized compatible.*

A generalized compatible can be viewed as a collection of compatible merged states. Each state set within a generalized compatible corresponds to a merged state. The trace set from a merged state is the union of the trace sets from states in the corresponding state set. The trace set associated with a generalized compatible is the intersection of trace sets from all merged states of the generalized compatible.

**Example** {A}, {B}, {C}, {D}, {E}, {F}, {BC}, {BC,D} are generalized compatibles. Some trace sets and corresponding transitions of {BC,D} are: {B,D} $\xrightarrow{0/0}$ {D,D}; {C,D} $\xrightarrow{1/1}$ {D,D}; {B,D} $\xrightarrow{0/0}$ {D,D} $\xrightarrow{0/0}$ {D,D}; {B,D} $\xrightarrow{0/0}$ {D,D} $\xrightarrow{1/1}$ {D,D}; {C,D} $\xrightarrow{1/1}$ {D,D} $\xrightarrow{0/0}$ {D,D}; {C,D} $\xrightarrow{1/1}$ {D,D} $\xrightarrow{1/1}$ {D,D}; ...

thm:PNDFSM-compatible The following theorem, analogous to Theorem 5.3, shows a recursive characterization of generalized compatibles that expresses the compatibility of a set of state sets in terms of the compatibilities of its sets of next state sets.

**Theorem 7.1** *A set $K$ of state sets is a* **generalized compatible** *if and only if, for each input $i$, there exists an output $o$ such that*

1. *for each state set in $K$, its set of transitions under input $i$ and output $o$ is non-empty, and*
2. *from the set $K$ of state sets, the set $K'$ of sets of next states under $i$ and $o$ is also a generalized compatible.*

## 7.2 Generalized Covering and Closure Conditions

The problem of state minimization of NDFSM's can be reduced to one of selecting a minimum subset of generalized compatibles. The selection must satisfy the following covering and closure conditions.

**Definition 7.2** *A set of generalized compatibles* **covers** *the reset state(s) if it contains at least one generalized compatible $c$ such that the set of reset states contains at least one state set in $c$ (i.e., at least one of its state sets is made up entirely of reset states).*

We require that an element in a selected generalized compatible behaves like a reset state but an element is now a set of states to be merged. To make sure that whatever transition we choose for the merged state, it will still correspond to a transition from a reset state, we require that its corresponding state set be made up entirely of reset states.

**Example** Only generalized compatible $\{A\}$ covers the reset state.

The closure condition of a classical compatible requires that the set of next states from the compatible be *contained* in another selected compatible. We now extend this notion of containment to sets of state sets for generalized compatibles.

**Definition 7.3** *A set $K$ of state sets* **contains** *another set $K'$ of state sets if for each state set $S'$ in $K'$, there is state set $S$ in $K$ such that $S'$ contains $S$.*

The trace set of a set of state sets is the intersection of the trace sets of the state sets, and the trace set of a state set is the union of the trace sets from the states in the set. If $K$ contain $K'$ according to Definition 7.3, then the trace set of $K$ contains the trace set of $K'$. For the trace set of $K$ to contain the trace set of $K'$, we require set containment be in the form $\forall S' \in K' \; \exists S \in K$ such that $S' \subseteq S$.

**Definition 7.4** *A set $\mathcal{G}$ of generalized compatibles is* **closed** *if for each generalized compatible $K \in \mathcal{G}$, for each input $i$, there exists an output $o$ such that*

1. *for each state set in $K$, its set of transitions under input $i$ and output $o$ is non-empty, and*
2. *from the set $K$ of state sets, the set $K'$ of sets of next states under $i$ and $o$ is contained in a generalized compatible of $\mathcal{G}$.*

**Example** The set of generalized compatibles $\mathcal{G} = \{\{A\}, \{BC, D\}\}$ is closed. The closure condition for $\{A\}$ requires the generalized compatible $\{A\}$ together with either $\{BC\}$ or $\{BC, D\}$. In the language of clauses it is: $(p_A) \cdot (p_{BC} + p_{BC,D})$. Note that the merged state $\{BC\}$ describes both the choice of going from $\{A\}$ to $\{B\}$ under input/output pair 1/0, than the one of going from $\{A\}$ to $\{C\}$ under input/output pair 1/0. The closure condition for $\{BC, D\}$ is $(p_D + p_{BC,D})$. These closure conditions are satisfied by the selection $\mathcal{G}$.

# 8 Algorithms for State Minimization of NDFSM's

## 8.1 Exact Algorithms for State Minimization

On the basis of the theory outlined in Section 7 one can devise algorithms to compute generalized compatibles. We do not know yet how to design an implicit algorithm along the lines of what done for ISFSM's and PNDFSM's. The main difficulty is to find a compact representation for sets of generalized compatibles. Given that generalized compatibles are already sets of sets of states, there is one more level of set construction complexity than for state minimization of ISFSM's or PNDFSM's. If such a representation exists, Theorem 7.1 gives a constructive definition of generalized compatible, and the covering and closure conditions in Section 7.2 can be solved as a binate covering problem.

As an alternative, suggested in [11], one can convert any NDFSM to a PNDFSM by an implicit determinization step as described below in Section 8.2 and then apply to the PNDFSM our implicit (or any) state minimization algorithm. It goes without saying that subset construction may introduce a blow-up in the number of original states, hurting the efficiency of PNDFSM minimization. It is an open problem whether a better procedure can be devised, or instead this exponential blow-up is intrinsic to the problem of minimizing NDFSM's. It must also be stressed that we do not have yet good sources of general NDFSM's in sequential synthesis, while the work in [11] has shown the pivotal importance of PNDFSM's in the synthesis of interconnected FSM's.

## 8.2 Implicit Subset Construction

Given the transition relation $T(i, s, s', o)$ of an NDFSM $M = \langle S, I, O, T, R \rangle$, first we compute the transition relation $T^{det}(i, c, c', o)$ of the determinized PNDFSM $M^{det}$. A 4-tuple $(i, c, c', o)$ is in relation $T^{det}$ if and only if the set of states $c$ on input $i$ can transit to another set of states $c'$, and simultaneously produce output $o$. $T^{det}(i, c, c', o) = \forall s \{[Single(s) \cdot (s \subseteq c)] \Rightarrow \exists s' [T(i, s, s', o) \cdot (s' \subseteq c')]\} \cdot \forall s' \{[Single(s') \cdot (s' \subseteq c')] \Rightarrow \exists s [T(i, s, s', o) \cdot (s \subseteq c)]\} \cdot \neg\emptyset(c) \cdot \neg\emptyset(c')$.

Given a 4-tuple $(i, c, c', o)$, the first clause on the right side of the above equation requires that for each singleton state $s$ ($Single(s) = 1$) contained in $c$, there is a next state $s'$ according to $T$ which is contained in $c'$. As a result, the next state set of $c$ is a subset of $c'$. With also the second clause, the 4-tuples in the relation will be such that $c'$ is exactly the next state set of $c$ on input $i$ and output $o$. Finally, we eliminate all 4-tuples expressing the fact that the empty state set can transit to the empty set under any input/output combination.

The power of the above computation is that we have effectively determinized the NDFSM $M$ into the PNDFSM $M^{det} = \langle 2^S, I, O, T^{det}, r^{det} \rangle$ where the new reset state is the set of reset states in $M$. Compared with explicit subset construction, no iteration nor state graph traversal is needed.

## 8.3 An Heuristic Algorithm for State Minimization

Exact minimization of NDFSM's requires generation of the generalized compatibles. If we restrict our attention only to the classical compatibles, the algorithm given in Section 5 for exact state minimization of

PNDFSM's will still serve as a heuristic algorithm for NDFSM minimization. This is because the algorithm chooses only from a subset of the generalized compatibles, namely the classical compatibles. For the NDFSM $M$ in Figure 3, such a heuristic algorithm cannot find the 2-state minimum machine $M_1$ but will only find the 4-state machine $M_2$. However, the heuristic algorithm is guaranteed to find at least one solution made up entirely of classical compatibles, because the original NDFSM is such a candidate solution. The hope is that most non-determinism expressed in an NDFSM is pseudo non-deterministic in nature, and as a result, the heuristic can find near optimum solutions most of the time.

# 9 Experimental Results

We have implemented an implicit algorithm for exact state minimization of PNDFSM's in a program called ISM2, a sequel to ISM [8]. Prime compatibles and the binate table are generated according to the algorithm described above; then a minimum cover of the table is found by the implicit binate covering solver described in [8]. We perform and report experiments on the complete set of examples obtained by Watanabe in [11]. Each PNDFSM is an E-machine derived from an arbitrary connection of two completely specified deterministic FSM's, $M_1$ and $M_2$, from the MCNC benchmark. The product machine $M = M_1 \times M_2$ is used as the specification. The E-machine which contains all permissible behaviors at $M_1$ is derived using the procedure in [11]. Our problem is to find a minimum state machine behaviorally contained in the E-machine.

Watanabe's minimizer, PND_REDUCE, does not compute prime compatibles but finds all compatibles instead. In its exact mode, compatible selection is performed by an explicit binate table solver available in the logic synthesis package SIS. In its heuristic mode, it finds instead a Moore machine in the E-machine, by 'expand' and 'reduce' operations [11] on a closed cover of compatibles. As the heuristic looks only to Moore solutions, it is understandable that it will give a worse solution than the minimum contained machine if the latter is not Moore. Also the run times might not be directly comparable. They are reported in the table for completeness.

Table 1 summarizes the results of PNDFSM minimization. For each PNDFSM, we report the number of states in the original PNDFSM, the number of states in a heuristic Moore solution obtained by PND_REDUCE, the number of states in a minimum contained FSM, the size of the binate table for PND_REDUCE and for ISM2, and the overall run time for state minimization for PND_REDUCE (in both heuristic and exact modes) and ISM2. All run times are reported in CPU seconds on a DECstation 5000/260 with 440 Mb of memory. For all experiments, timeout is set at 10000 seconds of CPU time, and spaceout at 440Mb of memory.

Out of the 30 examples, PND_REDUCE in exact mode failed to complete on 8 examples because of timeouts, and failed on 4 examples because of spaceout. It can handle all PNDFSM's with less than 16 states. PND_REDUCE in heuristic mode has a timeout on one example, and spaceout on three. Note that some Moore solutions found heuristically are far from the optimum contained ones, although the heuristic solutions may be close to the minimum Moore solutions. Our program ISM2 can handle more examples than PND_REDUCE and only failed to find an exact solution on 2 examples because of timeouts. In those two cases, ISM2 did succeed in computing the prime compatibles as well as building the binate covering table. Furthermore, for the example *pm41*, it found a solution with 9 states after 4844 seconds (whereas PND_REDUCE heuristic took 4011 seconds to find a 19-state Moore solution). For *pm50*, it found a first solution with 4 states in 150 seconds, and the minimum one with 3 states in 7309 seconds, while optimality was concluded in 49181 seconds after the complete branch-and-bound tree was searched (whereas PND_REDUCE heuristic can only find a Moore solution with 13 states). It is encouraging to note that our implicit exact algorithm has run times at least comparable to the heuristic one in PND_REDUCE, and in some cases, it is much faster.

Note that many exact minimum solutions had only one state, i.e., the solution is pure combinational

logic. Such is a Mealy machine unless the logic is a constant. It is believable that the minimum Moore behavior is much larger in general as indicated possibly by the results of PND_REDUCE.

Note also that each compatible results in a column of the binate table by PND_REDUCE in exact mode whereas ISM2 has one column for each prime compatible. The fact that most examples have very few prime compatibles shows the effectiveness of our prime compatibles computation for PNDFSM minimization. Even in these cases, state minimization may not be trivial because compatible generation and prime dominance may take a long time, e.g., *pm04* and *s3p1*.

# 10 Conclusions

In this paper, we have presented both theoretical and practical contributions to the problem of exploring contained behaviors and selecting one with a minimum number of states for classes of NDFSM's. In particular, we have contributed the following: (1) A fully implicit algorithm for exact state minimization of PNDFSM's. The results of our implementation are reported and shown to be superior to the explicit formulation described in [12]. We could solve exactly all the problems of the benchmark used in [11] (except one case, where a minimal solution not guaranteed to be the minimum was found). The explicit program could complete approximately one half of the examples, and in those cases with longer running times. (2) A theoretical solution to the problem of exact state minimization of general NDFSM's, based on the proposal of generalized compatibles. This gives an algorithmic foundation for exploring behaviors contained in a general NDFSM.

We are currently working on the problem of selecting out of a PNDFSM a minimum DFSM that can be implemented in an interconnection of two FSM's and we will present soon our results.

It is worth to underline that the first step of exact state minimization is the exploration of all possible behaviors contained in a NDFSM. For some classes of NDFSM's this can be achieved by computing compatibles (as classically defined in [4] and then extended in [12, 2]). Each closed collection of compatibles is a contained DFSM and vice versa. In the case of state minimization one wants a closed collection of compatibles of minimum cardinality. But one can replace the requirement of minimum cardinality with any other desired cost function or property (such as an implementable behavior) and obtain a new behavior selection problem. Therefore the exploration of all contained behaviors is a key technology for future applications in the synthesis of sequential networks. Implicit techniques are a winning tool to support sequential synthesis algorithms.

# References


[1] R. Bryant. Graph based algorithm for Boolean function manipulation. In *IEEE Transactions on Computers*, pages C–35(8):667–691, 1986.

[2] M. Damiani. Nondeterministic finite-state machines and sequential don't cares. In *European Conference on Design Automation*, pages 192–198, 1994.

[3] E.Cerny. Verification of I/O trace set inclusion for a class of non-deterministic finite state machines. In *The Proceedings of the International Conference on Computer Design*, pages 526–530, October 1992.

[4] A. Grasselli and F. Luccio. A method for minimizing the number of internal states in incompletely specified sequential networks. *IRE Transactions on Electronic Computers*, EC-14(3):350–359, June 1965.

[5] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley Publishing Company, 1979.

[6] J.E. Hopcroft. n log n algorithm for minimizing states in finite automata. *Tech. Report Stanford Univ. CS 71/190*, 1971.

[7] T. Kam. *State Minimization of Finite State Machines using Implicit Techniques.* PhD thesis, U.C. Berkeley, Electronics Research Laboratory, University of California at Berkeley, May 1995.

[8] T. Kam, T. Villa, R. Brayton, and A. Sangiovanni-Vincentelli. A fully implicit algorithm for exact state minimization. In *The Proceedings of the Design Automation Conference*, pages 684–690, June 1994.

[9] B. Lin and A.R. Newton. Implicit manipulation of equivalence classes using binary decision diagrams. In *The Proceedings of the International Conference on Computer Design*, pages 81–85, September 1991.

[10] J.-K. Rho, G. Hachtel, F. Somenzi, and R. Jacoby. Exact and heuristic algorithms for the minimization of incompletely specified state machines. *IEEE Transactions on Computer-Aided Design*, 13(2):167–177, February 1994.

[11] Y. Watanabe. *Logic Optimization of Interacting Components in Synchronous Digital Systems.* PhD thesis, U.C. Berkeley, Electronics Research Laboratory, University of California at Berkeley, April 1994.

[12] Y. Watanabe and R. K. Brayton. State minimization of pseudo non-deterministic FSM's. In *European Conference on Design Automation*, pages 184–191, 1994.

| PND-FSM | # states orig./heur. /exact | table size (rows x columns) PND_REDUCE exact | ISM2 exact | CPU time (seconds) PND_REDUCE heur. | PND_REDUCE exact | ISM2 exact |
|---|---|---|---|---|---|---|
| L3 | 17 / 2 / 2 | | 10 x 4 | 126.2 | time | 17.4 |
| am9 | 13 / 12 / 1 | | 1 x 1 | 13.7 | time | 2.3 |
| ax4 | 11 / 1 / 1 | 26 x 28 | 1 x 1 | 2 | 0.7 | 0.7 |
| ax7 | 20 / 2 / 2 | 334 x 308 | 20 x 6 | 2.8 | 15.6 | 7.6 |
| bx7 | 23 / 2 / 2 | 254 x 216 | 20 x 6 | 3.2 | 9.9 | 9.0 |
| dami | 5 / 5 / 3 | 21 x 24 | 17 x 10 | 0.1 | 0.1 | 1.3 |
| e4at2 | 14 / 11 / 1 | | 1 x 1 | 5.5 | time | 1.1 |
| e4bp1 | 11 / 1 / 1 | 1064 x 995 | 1 x 1 | 2.4 | 308.1 | 0.8 |
| e4t1 | 6 / 1 / 1 | 103 x 120 | 1 x 1 | 1.1 | 0.7 | 0.3 |
| e69 | 8 / 1 / 1 | 551 x 501 | 1 x 1 | 0.3 | 10.5 | 0.3 |
| e6tm | 21 / 8 / 1 | | 1 x 1 | 26.1 | time | 3.1 |
| ex10 | 13 / 4 / 1 | 23 x 28 | 1 x 1 | 0.6 | 0.5 | 0.6 |
| ex12 | 13 / 1 / 1 | 1451 x 1019 | 1 x 1 | 1.1 | 16149 | 0.8 |
| mc9 | 4 / 1 / 1 | 7 x 11 | 1 x 1 | 0.1 | 0.1 | 0.1 |
| mt51 | 16 / 10 / 1 | | 1 x 1 | 8.9 | time | 3.8 |
| mt52 | 9 / 4 / 1 | 256 x 639 | 1 x 1 | 3.7 | 39.4 | 0.8 |
| pm03 | 15 / 1 / 1 | 1203 x 1019 | 1 x 1 | 1.2 | 1751 | 0.8 |
| pm04 | 79 / / 1 | | 1 x 1 | space | space | 120.6 |
| pm11 | 9 / 1 / 1 | 331 x 395 | 1 x 1 | 43.5 | 29.7 | 1.3 |
| pm12 | 7 / 3 / 1 | 8 x 19 | 1 x 1 | 9.8 | 0.4 | 0.4 |
| pm31 | 22 / / 1 | | 1 x 1 | time | space | 3.6 |
| pm33 | 21 / 13 / 1 | | 1 x 1 | 3327 | time | 6.6 |
| pm41 | 33 / 19 / $\leq$9 | | 12050 x 4774 | 4011 | space | *4844 |
| pm50 | 22 / 13 / 3 | | 1249 x 515 | 32.2 | time | 49181 |
| s3p1 | 38 / / 1 | | 1 x 2 | space | space | 915.8 |
| s3t2 | 36 / / 1 | | 389 x 18 | space | time | 39.9 |
| tm01 | 10 / 1 / 1 | 476 x 767 | 1 x 1 | 1.2 | 40.9 | 0.8 |
| tm02 | 7 / 1 / 1 | 155 x 211 | 1 x 1 | 0.4 | 3.1 | 0.4 |
| tm31 | 9 / 1 / 1 | 125 x 113 | 1 x 1 | 0.3 | 1.1 | 0.3 |
| tm32 | 9 / 3 / 2 | 106 x 143 | 37 x 9 | 0.4 | 1.2 | 2.9 |

Table 1: State minimization of PNDFSM's.