IMPLICIT STATE MINIMIZATION OF NON-DETERMINISTIC FSM's

by

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

Memorandum No. UCB/ERL M95/18

22 March 1995

IMPLICIT STATE MINIMIZATION OF NON-DETERMINISTIC FSM's

by

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

Memorandum No. UCB/ERL M95/18

22 March 1995

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Implicit State Minimization of Non-Deterministic FSM's

Timothy Kam[*] Tiziano Villa[†] Robert Brayton Alberto Sangiovanni-Vincentelli
Department of EECS, University of California at Berkeley, Berkeley, CA 94720

March 7, 1995

Abstract

This paper addresses state minimization problems of different classes of non-deterministic finite state machines (NDFSM's). We present a theoretical solution to the problem of exact state minimization of general NDFSM's, based on the proposal of a notion of generalized compatibles. This gives an algorithmic frame to explore behaviors contained in a general NDFSM. Then 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 running times.

1 Introduction

Implicit techniques are based on the idea of operating on discrete sets by their characteristic functions represented by Binary Decision Diagrams (BDD's) [2]. In many cases of practical interest these sets have a regular structure that translates into small-sized BDD's. BDD's can be manipulated efficiently with the usual Boolean operators. Implicit techniques increase the size of problems that can be solved exactly in logic synthesis and verification.

In [10] an implicit algorithm for exact state minimization of incompletely specified finite state machines (ISFSM's) has been described. It was based on new implicit techniques to generate prime compatibles and to build and solve a binate table. ISFSM's are a subclass of non-deterministic finite state machines (NDFSM's) and recently more classes of NDFSM's have been introduced in sequential logic synthesis as a way to capture flexibility in networks of finite state machines (FSM's). Especially important are pseudo non-deterministic FSM's (PNDFSM's), introduced in [17], that are sufficient to express the flexibility in an arbitrary interconnection of two FSM's. A family of extensions of PNDFSM's called k-PNDFSM's, for k any natural number, has also been proposed in [6, 7]. Extracting out of a 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. We call it in the sequel simply state minimization problem, and it must not be confused with behavior-preserving state minimization, of more interest in theoretical computer science community.

In [19, 4] the problem of extracting a minimum state behavior out of a PNDFSM has been attacked and it has been shown that an exact solution can be obtained by extending the notion of compatibles and formulating a binate table problem. In [19] a contribution has also been made to the problem of selecting a DFSM that can be implemented in an interconnection of two FSM's. Here we present a two-fold contribution:

1. A theoretical solution to the problem of exact state minimization of general NDFSM's, based on the proposal of a notion of generalized compatibles. This gives an algorithmic frame to explore behaviors contained in a general NDFSM.

[*]Research supported by California State MICRO Program, DEC and Intel
[†]Research supported by NSF under contract MIP-8719546 and California State MICRO Program

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 [19]. We could solve exactly all the problems of the benchmark used in [17] (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.

We are currently working also on the problem of selecting out of an NDFSM a minimum DFSM that can be implemented in an interconnection of two FSM's and we will soon present 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 achieved by computing compatibles (as classically defined and then extended in [19, 4]). Each closed collection of compatibles is a contained DFSM and viceversa. 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 behavior selection problem. 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 implicit techniques that we are investigating is a winning tool to support synthesis algorithms.

The remainder of the paper is organized as follows. Our taxonomy of different classes of finite state machines and their state minimization problems is proposed in Section 2. Section 3 reports the current status of these state minimization problems. A new algorithm for state minimization of NDFSM's is proposed in Section 4. Section 5 introduces implicit representations and manipulations. Then a fully implicit algorithm for state minimization of PNDFSM's is presented in Section 6. Considerations on an implicit algorithm for general NDFSM's are discussed in Section 7. Results on minimization of PNDFSM's are reported in Section 8. Conclusions and future work are summarized in Section 9.

2 Definitions

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.

2.1 Taxonomy of Finite State Machines

Definition 2.1 *A* **deterministic FSM** *(DFSM) or* **completely specified FSM** *(CSFSM) 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. δ 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 $n = \delta(i, p)$. λ 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.*

A behavior between the input variables I and the output variables O is the set of pairs of input and output sequences realized by a completely specified deterministic finite state machine with the input I and the output O. A formal definition follows.

Definition 2.2 *Given a finite set of input variables I and a finite set of output variables O, a* **behavior** *between I and O is a set of pairs of input and output sequences, $\mathcal{B} = \{(\sigma_i, \sigma_o) \mid |\sigma_i| = |\sigma_o|\}$, which satisfies the following conditions:*

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

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

For a non-deterministic finite state machine, there might exist more than one valid transition for some state and an input. Therefore, while a DFSM represents a single behavior, a non-deterministic FSM (NDFSM) can be viewed as representing a set of behaviors. Each such behavior is called a contained behavior. Then an NDFSM expresses handily flexibilities in sequential synthesis.

Definition 2.3 *A* **non-deterministic FSM** *(NDFSM) is defined as a 5-tuple $M = \langle S, I, O, T, R \rangle$ where S represents the finite state space, I represents the finite input space and O represents the finite output space. 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. An FSM defines a transition structure that can be described by means of edges. By an edge (i, p, n, o), the FSM transits from state p on input i to state n with output o.

To capture flexibility/choice/don't care/non-determinism in the next state n and/or the output o from a state p at an input i, one can specify one or more transitions $(i, p, n, o) \in T$. As said above, we assume that the state transition relations T is complete with respect to i and p, i.e., there is always at least one transition from each state on each input. This differs from the situation in formal verification where incomplete automata are considered.

Definition 2.4 *A state transition relation T is complete iff $\forall i, p \; \exists n, o \; [T(i, p, n, o)] \equiv 1$.*

We introduce now two useful classes of FSM's: pseudo non-deterministic FSM's and incompletely specified FSM's. An NDFSM is a pseudo non-deterministic FSM (PNDFSM) iff for each triple $(i, p, o) \in I \times S \times O$, there is a unique state n such that $T(i, p, n, o) = 1$. 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 edges carrying different outputs must go to different next states [1].

Definition 2.5 *A* **pseudo non-deterministic FSM** *(PNDFSM) is a 6-tuple $M = \langle S, I, O, \delta, \Lambda, R \rangle$. δ 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. Λ 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, present state and input, 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 Λ.

One can extend the previous definition to get a numerable family of machines as follows. An NDFSM is a k-step pseudo non-deterministic FSM (k-PNDFSM), where $k \in \omega$, iff for any present state the choice of the next state can be uniquely identified by observing input-output sequences of length up to k. An NDFSM is a k-step pseudo non-deterministic FSM (k-PNDFSM), where $k \in \omega$, iff for each tuple $(i, i_2, \ldots, i_k, p, o, o_2, \ldots, o_k) \in I \times I \times \cdots \times I \times S \times O \times O \times \cdots \times O$ there is a unique next state n and there are states $s_2, \cdots, s_k$ such that $T(i, p, n, o) = 1$ and $T(i_2, n, s_2, o_2) = T(i_3, s_2, s_3, o_3) = \cdots = T(i_k, s_{k-1}, s_k, o_k) = 1$.

Definition 2.6 *A* **k-step pseudo non-deterministic FSM** *(k-PNDFSM) is a 6-tuple $M = \langle S, I, O, \delta, \Lambda, R \rangle$. δ is the next state function defined as $\delta : I \times \cdots \times I \times S \times O \times \cdots \times O \rightarrow S$ where each combination of input, present*

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

state, k inputs and outputs gives a unique next state. Λ 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.

By definition, a PNDFSM is an 1-PNDFSM. Cerny [7] has given a polynomial algorithm to convert a k-PNDFSM to a PNDFSM. A k-PNDFSM has a representation smaller or equal to that of an equivalent PNDFSM [2].

An NDFSM is an incompletely specified FSM (ISFSM) iff for each pair $(i, p) \in I \times S$ such that $T(i, p, n, o) = 1$, (1) the machine can transit to a unique next state n or to any next state, and (2) the machine can produce a unique output o or produce any output.

Definition 2.7 *An* **incompletely specified FSM** *(ISFSM) 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. Δ is the next state relation defined as a characteristic function $\Delta : I \times S \times S \rightarrow B$ where each combination of input and present state is related to a single next state or to all states. Λ is the output relation defined as a characteristic function $\Lambda : I \times S \times O \rightarrow B$ where each combination of input and present state is related to a single output or to all outputs. $R \subseteq S$ represents the set of reset states.*

Incomplete specification is used here to express some types of don't cares in the next states and/or outputs. We warn that even though "incompletely specified" is established terminology in the logic synthesis literature, it conflicts with the fact that ISFSM's have a transition relation T that is actually completely specified with respect to present state p and input i, because there is at least one transition for each (i, p) pair in T.

We can restate the definition of a CSFSM in terms of the relation T.

Definition 2.8 *A* **deterministic FSM** *(DFSM) or* **completely specified FSM** *(CSFSM) is an NDFSM where for each pair $(i, p) \in I \times S$, there is a unique next state n and a unique output o such that $T(i, p, n, o) = 1$, i.e., there is a unique transition from (i, p). In addition, R contains a unique reset state.*

Note that an ISFSM and a DFSM are both next-state output uncorrelated because we can represent the next state and output information separately. But a PNDFSM (and k-PNDFSM) is not next-state output uncorrelated as the next state is correlated with the output by $n = \delta(i, p, o)$.

2.2 Taxonomy of State Minimization Problems

Definition 2.9 *Given an NDFSM $M = \langle S, I, O, T, R \rangle$, a state $s_0 \in S$, and an input sequence $\{i_0, i_1, \ldots, i_j\}$, an output sequence $\{o_0, o_1, \ldots, o_j\}$ is* **possible** *from s_0 in M iff $\exists$ a state sequence $\{s_1, s_2, \ldots, s_{j+1}\}$ such that $\forall k : 0 \leq k \leq j, T(i_k, s_k, s_{k+1}, o_k) = 1$.*

For each state in a deterministic FSM, exactly one possible output sequence corresponds to each input sequence. Given an initial state, a deterministic FSM represents a unique input-output behavior. 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. This leads naturally to the notion of behavioral containment between specifications.

Definition 2.10 *Given NDFSM's $M = \langle S, I, O, T, R \rangle$ and $M' = \langle S', I, O, T', R' \rangle$, $M' \subseteq M$ iff $\forall r \in R \ \exists r' \in R' \ \forall$ input sequences $\forall$ output sequences possible from r' in M', the same output sequence is possible from r in M.*

Definition 2.11 *Given an NDFSM M, the* **state minimization problem** *is to find another NDFSM M' such that*

1. $M' \subseteq M$, and

[2] For every STG containing unspecified next-states one can construct an STG where all unspecified next states are replaced by a trap state D, as in [13]. The transitions from D under any input go to D itself and their outputs are unspecified. The new STG describes exactly the same state minimization problem as the old one, but the latter STG represents more behaviors than the former one.

2. *$\forall M''$ s.t. $M'' \subseteq M$, number of states in $M' \leq$ number of states in M''.*

Note that M' is not required to be deterministic. On the contrary, to preserve flexibility for other sequential synthesis tools, one may want choose the M' with maximal non-determinism in specification, out of all state minimal machines.

Definition 2.10 of machine containment and Definition 2.11 of the state minimization problem apply also to other kinds of FSM's.

The state minimization problem defined above is very different from the NDFA minimization problem described in classical automata textbooks. Here we want a minimal state implementation which is contained in the specification, while the classical problem is defined as:

Definition 2.12 *Given an NDFSM M, the* **behavioral-preserving state minimization problem** *is to find an NDFSM M' which represents the same set of behavior as M but has the fewest number of states.*

2.3 Exact State Minimization

Exact algorithms for the state minimization problems (ISFSM's and PNDFSM's) are based on the generation of a collection of state sets called the compatibles.

Definition 2.13 *A set of states is a* **compatible** *iff for each input sequence, there is a corresponding output sequence which can be produced by each state in the compatible.*

This definition says that states within a compatible can potentially be merged together to form a single state in the minimized machine. After the set of compatibles is generated, the second step of an exact state minimization algorithm is to select a subset 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 a **minimum closed cover** of compatibles. The selection is usually solved as a binate covering problem.

The set of compatibles is usually very large. For the purpose of state minimization, it is useful to identify a minimum subset called the prime compatibles such that a minimum closed cover of prime compatibles still yields a minimum contained machine. According to the following definitions, compatibles that are not dominated by other compatibles are called prime compatibles:

Definition 2.14 *A compatible c'* **prime dominates** *a compatible c iff for each compatible selection containing c which corresponds to a minimum machine, the selection with c replaced by c' also corresponds to a minimum machine.*

Definition 2.15 *A compatible is a* **prime compatible** *iff it is not prime dominated by an other compatible.*

The actual computations of compatibles and primeness differ for different types of FSM's. Also the related covering problems vary slightly.

Note that in the previous definitions the state space S, the input space I and the output space O can be generic discrete spaces and so S, I and O can assume symbolic values [5, 16]. A special case is when S, I and O are the cartesian product of copies of the space $B = \{0, 1\}$, i.e., they are binary variables. The fact that the FSM's have symbolic vs. binary encoded input and output variables does not change the formulation of problem, nor the solution based on the computation of compatibles. The theory extends in a straightforward manner to encoded state spaces.

3 Status of State Minimization Problems

The problem of state minimization has already been solved for some classes of FSM's:

- In the case of CSFSM's, it coincides with the problem of behavioral-preserving state minimization. The most efficient algorithm is in [9]. An implicit algorithm is in [12].
- Exact algorithms for the problem of state minimization of ISFSM's have been presented in classical papers [14, 8]. The best computer implementation is in [15]. An implicit formulation of the exact algorithm has been presented in [10].
- Explicit algorithms for exact state minimization of PNDFSM's have been described in [19, 4] [3].

The gist of the contributions of [19, 4] has been to show that "an incremental update" of the classical definition of compatibles as introduced by [14, 8] is sufficient to explore all possible behaviors contained in PNDFSM's and so to extract the minimum one. It is an open problem to characterize the maximal classes of NDFSM's for which analogous notions of compatibles are sufficient to solve exactly state minimization. We will see that in the more general case of NDFSM's we need a definition of generalized compatibles that is a more drastic departure from the previous ones.

Here we propose explicit and implicit algorithms to fill the existing gaps in the literature. In particular we are going to investigate:

- Fully implicit algorithm for exact state minimization of PNDFSM's. Exact algorithms have been proposed by [19, 4]. The algorithms in [19] use some implicit techniques of [10], and handle the more complex problem of state minimization with implementability.
- Explicit and implicit algorithms for exact state minimization of NDFSM's. Currently no such algorithm is known. The only known method to handle NDFSM's is to generate an equivalent PNDFSM by determinization of the underlying automaton. The shortcoming of that approach is that, in the worst case, the state space becomes exponentially larger.

4 State Minimization of NDFSM's

Is it possible to apply the classical procedure based on computing compatibles to NDFSM'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, as shown by the counter example in Figure 1.

Given the NDFSM M as shown in Figure 1a, the minimum state DFSM M_1, as shown in Figure 1b, contained in M, cannot be found using compatibles alone. By Definition 2.13, states B and C are not compatible. As a result, any minimized machine M_2 obtained by compatible-based algorithms will have at least 4 states, one of the two such minimized machines being shown in 1c. However with the non-deterministic transitions into states B and C, we can choose their outgoing transitions in a way as shown in Figure 1b such that B and C are merged together as one state. This merged state is compatible to state D. This merging possibility is not explored by compatibility. The minimum state DFSM, whose behavior is contained in the original NDFSM, has only two states.

4.1 Generalized Compatibles

By the following definition, we generalize the notion of a compatible to one that consists of a set of state sets. Each individual state set contains states that can be merged because they can be non-deterministically reached from an initial state. If the original FSM doesn't have any non-deterministic transition, each such state set will be

[3]Since a k-PNDFSM can be reduced polynomially to a PNDFSM, also the former ones can be treated in this framework.

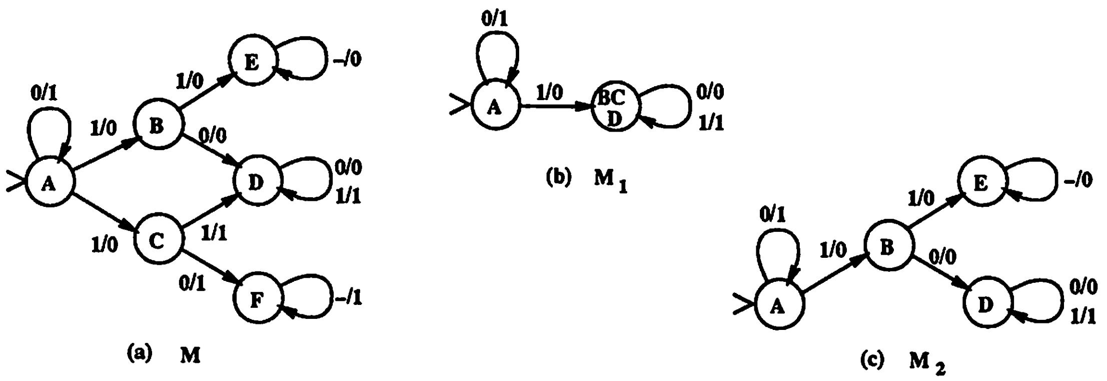

Figure 1: A counter example, a) the NDFSM M, b) the minimum state DFSM contained in M, c) one DFSM contained in M found using compatibles.

a singleton. In such a case, each generalized compatible is a set of singleton-states, corresponding to a classical compatible.

Definition 4.1 *A set of state sets is a* **generalized compatible** *iff 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.*

Example: {A}, {B}, {C}, {D}, {E}, {F}, {BC}, {BC,D} are generalized compatibles. The enumeration of input sequences for {BC,D} are as follows: $\{B,D\} \xrightarrow{0/0} \{D,D\}$; $\{C,D\} \xrightarrow{1/1} \{D,D\}$; $\{B,D\} \xrightarrow{0/0} \{D,D\} \xrightarrow{0/0} \{D,D\}$; $\{B,D\} \xrightarrow{0/0} \{D,D\} \xrightarrow{1/1} \{D,D\}$; $\{C,D\} \xrightarrow{1/1} \{D,D\} \xrightarrow{0/0} \{D,D\}$; $\{C,D\} \xrightarrow{1/1} \{D,D\} \xrightarrow{1/1} \{D,D\}$; ...

Given a NDFSM F with n states, the definition of generalized compatible requires that for all input sequences, there is a common output sequence agreed by at least one state of each state set in the generalized compatible.

For the definition to be constructive, we need a bound on the length of input sequences that must be considered in order to decide compatibility.

Lemma 4.1 *One can decide whether the initial states of two NDFSM's are compatible by examining all input sequences of length at most $O(n.m)$, if n and m are the number of states of the two machines.*

Proof: Given a pair of states and an input sequence, we say that the input sequence runs successfully if a common output sequence can be produced from the two states. A run of an input sequence is the pairs of states that it visits. Suppose that, when starting in both FSM's from their respective reset states, all input sequences of length $\leq n.m$ run successfully. Each such input sequence must come back to pairs of states already visited, because there are only $n.m$ different pairs of states; in the worst-case an input sequence will visit all of them and when reading the $n.m$-th input symbol will return to pairs already seen. Say that one of these of pairs of states is (s_k, s'_k), reached under input sequence $\tilde{i}$. The pair (s_k, s'_k) can also be reached from the initial states by an input sequence that is a prefix of $\tilde{i}$, say $\tilde{i}_k$, of length $l_k < n.m$. By hypothesis we can run successfully any input sequence $\tilde{i}_k i$ that is the concatenation of $\tilde{i}_k$ and one more input symbol i (because $l_k + 1 \leq n.m$). Moreover, the input sequence $\tilde{i}_k i$ will reach other pairs of states already visited. An inductive argument shows that any extension (of arbitrary length) of $\tilde{i}$ can be run successfully. Therefore by inspecting all input sequences of length $\leq n.m$ one can decide if the two reset states are compatibles. If the cardinality of the set of inputs is k, then there are $k^{n.m}$ input sequences of length $\leq n.m$. Since the two machines are non-deterministic, for every input sequence of length $n.m$ there might exist $(n.m)^{n.m}$ distinct output sequences. □

Theorem 4.1 *To decide whether a set of set states of a NDFSM F with n states is a generalized compatible, it is sufficient to verify the definition for all input sequences of length bounded by $O(n^n)$.*

Proof: The core operation to decide generalized compatibility is to find out whether two given states s_1 and s'_1 are compatible, i.e., whether for every input sequence started at them there is an output sequence on which the two states agree. Consider two versions F_{s_1} and $F_{s'_1}$ of the NDFSM F, one with initial state s_1 and the other with initial state s'_1. We want to find whether from the reset states for every input sequence there is a common output behavior. By the previous lemma one can decide whether the initial states of two NDFSM's are compatible by examining all input sequences of length at most $O(n.n)$. If the cardinality of the set of inputs is k, then there are $k^{n.n}$ input sequences of length $\leq n.n$. Since the two machines are non-deterministic, for every input sequence of length $n.n$ there might exist $(n.n)^{n.n}$ distinct output sequences. To decide compatibility of more than 2 states, one can make more copies of the NDFSM and apply the same reasoning. Each new copy brings a factor n in the algorithm. At most one might need to check the compatibility of n states. The point of the exercise is to show that a finite length suffices, even though this naive bound is outrageous in practice. □

It would be hardly practical to compute the generalized compatibles directly from the definition. The following theorem 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 4.2 *A set K of state sets is a* **generalized compatible** *iff 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.*

Note the similarity with the generation of classical compatibles, where we require a set of states to be (1) output compatible, and (2) its next state set to be compatible.

Now, 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:

4.2 Generalized Covering Conditions

Definition 4.2 *A set of generalized compatibles* **covers** *the reset state(s) iff 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).*

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

4.3 Generalized Closure Conditions

Definition 4.3 *A set K of state sets* **contains** *another set K' of state sets iff for each state set S' in K', there is state set S in K such that S' contains S.*

Definition 4.4 *A set of generalized compatibles $\mathcal{G}$ is* **closed** *iff 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 compatible $\mathcal{G} = \{\{A\}, \{BC, D\}\}$ is closed. Closure condition for $\{A\}$ can be represented by the clauses: $(\{A\} \Rightarrow \{A\}) \cdot (\{A\} \Rightarrow \{B\} + \{C\} + \{BC\} + \{BC, D\})$. Closure for $\{BC, D\}$ requires that $\{BC, D\} \Rightarrow (\{D\} + \{F\}) \cdot \{D\}$ and $\{BC, D\} \Rightarrow (\{E\} + \{D\}) \cdot \{D\}$.

4.4 Relationship to Determinization and PNDFSM Minimization

Definition 4.5 *A set of states is a* **mergeable** *iff it corresponds to a state label on the determinized state transition graph.*

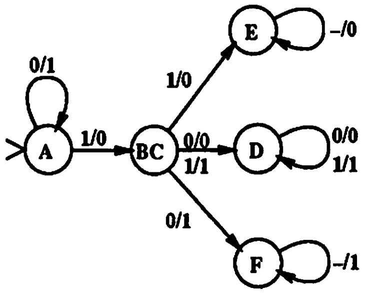

Figure 2: Determinized State Transition Graph of M in Figure 1a

Example: The mergeables of NDFSM M in Figure 1a are A, BC, D, E, F, which are state labels on the determinized state graph shown in Figure 2.

Theorem 4.3 *At least one minimum closed cover consists entirely of generalized compatibles whose state sets are mergeables.*

Example: The minimum closed cover {{A},{BC,D}} is made up of mergeables only.

Therefore to form the covering table and solve the covering problem exactly, it is sufficient to generate only the generalized compatibles made up of mergeables. Therefore if one replaces the words "state sets" with "mergeables" in the previous definitions one obtains a more compact representation of the set of generalized compatibles.

5 Implicit Representations and Manipulations

Algorithms for sequential synthesis have been developed primarily for State Transition Graphs (STG's). STG's have been usually represented in two-level form where state transitions are stored explicitly, one by one. Alternatively, STG's can be represented implicitly with Binary Decision Diagrams (BDD's) [2, 1]. BDD's represent Boolean functions (e.g. characteristic functions of sets and relations) and have been amply reported in the literature [2, 1], to which we refer.[4]

5.1 Positional-set Representation

To perform state minimization, one needs to represent and manipulate efficiently sets of states, or state sets, (such as compatibles) and sets of sets of states (such as sets of compatibles). Our goal is to represent any set of sets of states implicitly as a single BDD, and manipulate such state sets symbolically all at once. Different sets of sets of states can be stored as multiple roots with a single shared BDD.

Suppose a FSM has n states, there are 2^n possible distinct subsets of states. In order to represent collections of them, each subset of states is represented in **positional-set** form, using a set of n Boolean variables, $x = x_1 x_2 \dots x_n$. 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. One Boolean variable is needed for each state because the state can either be present or absent in the set.[5] 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 011010.

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

[5] The representation of primes proposed by Coudert *et al.* [3] needs 3 values per variable to distinguish if the present literal is in positive or negative phase or in both.

A set of sets of states is represented as a set S of positional-sets by a characteristic function $\chi_S : B^n \rightarrow B$ as: $\chi_S(x) = 1$ iff 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.

5.2 Operations on Positional-sets

With our definitions of relations and positional-set notation for representing set of states, useful operators on sets and sets of sets can be derived. We have proposed in [10] a unified notational framework for set manipulation, extending the work by Lin *et al.* in [11]. Here we define some relationships between two sets, between two sets of sets, between a set and a set of sets, etc.

Lemma 5.1 *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 5.2 *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.*

Lemma 5.3 *Given a characteristic function $\chi_A(x)$ representing a set A of positional-sets,* **set union** *defines a positional-set y which represents the union of all state sets in A, and can be computed by:*

$$Set\_Union_x(\chi_A, y) = \prod_{k=1}^{n}(y_k \Leftrightarrow \exists x \, [\chi_A(x) \cdot x_k])$$

For each bit position k, the right-hand expression sets y_k to 1 iff there exists an $x \in \chi_A$ such that its kth bit is a 1. This implies that the positional-set y will contain the k-th element iff there exists a positional-set x in A such that k is a member of x.

Lemma 5.4 *The* **minimal** *of a set F of sets is the set containing sets in F not strictly containing any other set in F, and is given by:*

$$Minimal_x(\chi_F) = \chi_F(x) \cdot \nexists y \, [\chi_F(y) \cdot (y \subset x)]$$

The term $\exists y \, [\chi_F(y) \cdot (y \subset x)]$ is true iff there is a positional-set y in χ_F such that $y \subset x$. In such a case, x cannot be in the minimal set by definition, and is taken away from $\chi_F(x)$.

5.3 k-out-of-n Positional-sets

We define a family of sets of state sets, $Tuple_{n,k}(x)$, which contain all positional-sets $x \subseteq S$ with exactly k states in them. Their BDD's can constructed by the following algorithm, by calling $Tuple(n, k)$:

```
Tuple($i, j$) {
    if ($j < 0$) or ($i < j$)  return 0
    if ($i = 0$) and ($i = j$)  return 1
    return ITE($x_i$, Tuple($i - 1, j - 1$), Tuple($i - 1, j$))
}
```

$Tuple(i, j)$ contains positional-sets of cardinality j with i variables, $x_1, x_2, \ldots, x_i$, which can be grouped into those that include state i and those that do not. The latter group simply corresponds to $Tuple(i-1, j)$, the set of positional-sets of cardinality j with only $x_1, x_2, \ldots, x_{i-1}$ (using one less variable). The former group can be obtained by adding state i to each positional-set in $Tuple(i - 1, j - 1)$, the set of positional-sets of cardinality $j - 1$ with $i - 1$ variables. Therefore $Tuple(i, j)$ can be computed recursively by $ITE(x_i, Tuple(i - 1, j - 1), Tuple(i - 1, j))$. Recursion can stop when a termination condition as shown is met. The BDD size and time complexity of $Tuple(n, k)$ are both $O(nk)$, provided its intermediate results are memoized in a computed table ([1]).

5.4 *cproject* BDD Operator

Definition 5.1 *[12] Given an equivalence relation $E(x,y)$, the cproject operator selects a single element from each equivalent class to represent the class. $\perp(x)$ is a function that maps every element x of an equivalence class to the representative of the equivalence class.*

$$cproject_y(E(x,y)) = \{(x,y)|(x,y) \in E, y = \perp(x)\}$$

5.5 Transition Relation of Determinized PNDFSM and Implicit Subset Construction

Given the transition relation $T(i,s,s',o)$ of an NDFSM $\langle S, I, O, T, R\rangle$, first we compute the transition relation of the determinized PNDFSM, $\tau'(i,c,c',o)$. A 4-tuple (i,c,c',o) is in relation τ' 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. The advantage of using a 1-hot encoding (i.e., positional set notation) to represent states in the original NDFSM, e.g. s, is that a state in the determinized PNDFSM, e.g. c, (corresponding to a set of states in the NDFSM) can be represented by a minterm in the encoded space. τ' can be computed by the following formula:

$$\begin{aligned}
\tau'(i,c,c',o) &= \forall s\ \{[Tuple_1(s) \cdot (s \subseteq c)] \Rightarrow \exists s'\ [T(i,s,s',o) \cdot (s' \subseteq c')]\} \\
&\quad \cdot \forall s'\ \{[Tuple_1(s') \cdot (s' \subseteq c')] \Rightarrow \exists s\ [T(i,s,s',o) \cdot (s \subseteq c)]\} \\
&\quad \cdot \neg Tuple_0(c) \cdot \neg Tuple_0(c') \\
&= Set\_Union_{s' \to c'}\{\exists s\ [(c \supseteq s) \cdot T(i,s,s',o)]\} \cdot \neg Tuple_0(c) \cdot \neg Tuple_0(c')
\end{aligned}$$

Given a 4-tuple (i,c,c',o), the first clause on the right requires that for each singleton state s 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 effectively determinized the NDFSM into the PNDFSM $\langle 2^S, I, O, \tau', r'\rangle$ where the new reset state is the union of reset states in the NDFSM, $r'(c) = Set\_Union_{s \to c} R(s)$. Compared with explicit subset construction, no iteration nor state graph traversal is needed.

The above relation $\tau'(i,c,c',o)$, derived from the transition relation T, is useful in the computation of compatibles and closure conditions in Section 6. τ' contains many 4-tuples, as c can be any output compatible set of states. During compatible generation, τ' will be restricted to τ by forcing c and c' to be compatibles:

$$\tau(i,c,c',o) = \tau'(i,c,c',o) \cdot C(c) \cdot C(c')$$

If some applications other than state minimization need to know the reachable state space $S' \subseteq 2^S$ of the determinized PNDFSM, it can be computed by the following fixed point computation:

- $S'_0(c) = r'(c)$
- $S'_k(c) = S'_k(c) + [c' \to c]\exists c, i, o\,\{S'_k(c) \cdot \tau'(i,c,c',o)\}$

The above iteration can be terminated when for some j, $S'_{j+1} = S'_j$ and the least fixed point is reached. The set of reachable states of the determinized PNDFSM is $S'(c) = S'_j(c)$.

6 State Minimization of PNDFSM's

The algorithm for state minimization of PNDFSM's consists of two steps: compatible generation and binate covering. It is more complicated than the one for ISFSM minimization [10] because the definition of compatibles and the conditions for a closed cover are more complex.

The PNDFSM M_p shown in Figure 3 will be used in this section to illustrate various concepts.

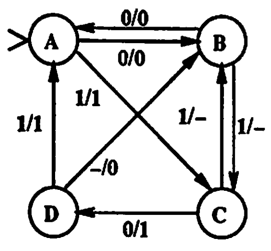

Figure 3: A PNDFSM, M_p

6.1 Compatibles

The following theorem serves as an equivalent, constructive definition of compatibles (*c.f.* Definition 2.13). The theorem yields an implicit compatible generation procedure.

Theorem 6.1 *A set c of states is a* **compatible** *iff 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 compatibles of M_p of Figure 3. 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.

6.2 Covering and Closure Conditions for Compatibles Selection

Definition 6.1 *A set of compatibles* **covers** *the reset state(s) iff at least one selected compatible contains a reset state.*

Example: The covering condition for M_p is the simple clause (A) which requires compatible A be selected.

If no reset state is given, it can be assumed that each state in S has to be covered by some compatible (as it is common in classical papers on the minimization of ISFSM's).

Definition 6.2 *A set of compatibles C is* **closed** *iff 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 selected compatible.*

Example: Consider the closure condition for compatible D. On input 0, D transits to B, and on input 1, it can transit to A (and output 1) *or* transit to B (and output 0). The closure condition for D is the conjunction of disjunctions $(B + AB) \cdot (A + B + AB)$. Similarly, the closure condition for A is $(B + AB) \cdot (C)$, for B is $(A + AB) \cdot (C)$, for C is $(D) \cdot (B + AB)$ and for compatible AB is (C).

Theorem 6.2 *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.*

6.3 Prime Compatibles

For runtime efficiency, we compute the set of prime compatibles which is a subset of the compatibles. Prime dominance has been introduced by Definition 2.14. A sufficient condition for prime dominance is given by the following theorem:

Theorem 6.3 *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 of c implies*[6] *the closure condition of c', and*
3. *$c' \supset c$.*

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 2.14). 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[7]. 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 not closed. In case (1), $C \cup \{c'\}$ is not a cover iff $(c \cap R) \neq \emptyset$ and $(c' \cap R) = \emptyset$, i.e., condition 1 of the above theorem is false. In case (2) $C \cup \{c'\}$ is not closed, if one of the two is true: (2a) C satisfies the closure condition of c but not the closure condition of c'. This happens iff condition 2 is false. (2b) c is needed to satisfy the closure condition of 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 is false.

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

Example: Compatible AB prime dominates compatible B because all conditions of Theorem 6.3 are true. In particular, closure condition of B implies closure condition of AB because $(A + AB) \cdot C \Rightarrow C$. Similarly, AB dominates A. As a result, the prime compatibles are AB, C, D.

6.4 Logical Representation of Closure Conditions

We now build a set of logical clauses expressing the closure requirement that a next state set d is contained by at least one selected compatible, as stated in Definition 6.2. Since we are going to generate the set of prime compatibles, we express also part 3 of Theorem 6.3 that refers to the implication between closure conditions.

The notion of next state sets d is important for expressing closure conditions.

Definition 6.3 *$d_{c,i,o}$ is the set of next states from compatible c under input i and output o.*

Given a triple (c, i, o), such a set is unique in a PNDFSM. We associate to each $d_{c,i,o}$ a clause whose literals are the prime compatibles that contain $d_{c,i,o}$. For simplicity of 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: $d_{D,0,0}$ 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 $(p_B + p_{AB})$.

Theorem 6.4 *Clause $d_{c',i',o'} \Rightarrow$ clause $d_{c,i,o}$ if the set of next states $d_{c',i',o'} \supseteq$ the set of next states $d_{c,i,o}$.*

[6]Condition A implies condition B iff the satisfaction of condition A automatically guarantees the satisfaction of condition B. In other words, A is *not less restrictive* than B.

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

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.

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 clause (or disjunctive clause or disjunction), $disjunct(c,i)$, defined as:

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

For a PNDFSM, the closure condition of a compatible c, denoted by $closure(c)$, has the form of a conjunction of disjunctive clauses. According to Definition 6.2, the conjunction is over all inputs, while the disjunction is over specified outputs. Given a compatible c, the following product of disjunctions must be satisfied (one disjunction per input):

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

In summary, the closure condition for compatible c is fulfilled iff 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, this closure condition of c is fulfilled iff the product-of-sums

$$\forall i \in \text{inputs} \; \exists o \in \text{outputs at } c \text{ under } i, \;\; d_{c,i,o}$$

is satisfied. These closure conditions are tested against a certain selection of compatibles.

We prove first two useful lemmas.

Lemma 6.1 $\forall x' \, \exists x \, [F(x) \Rightarrow F'(x')]$ *iff* $[\forall x \, F(x)] \Rightarrow [\forall x' \, F'(x')]$.

Proof: By using some fundamental validities of logic:

$$\forall x' \, \exists x \, [F(x) \Rightarrow F'(x')]$$

iff

$$\forall x' \, [\forall x \, F(x) \Rightarrow F'(x')]$$

iff

$$\forall x \, F(x) \Rightarrow \forall x' \, F'(x').$$

□

Lemma 6.2 $\forall x' \, \exists x \, [F'(x') \Rightarrow F(x)]$ *iff* $[\exists x' \, F'(x')] \Rightarrow [\exists x \, F(x)]$.

Proof: By using some fundamental validities of logic:

$$\forall x' \, \exists x \, [F'(x') \Rightarrow F(x)]$$

iff

$$\forall x' \, [F'(x') \Rightarrow \exists x \, F(x)]$$

iff

$$\exists x' \, F'(x') \Rightarrow \exists x \, F(x).$$

□

We present now the prime dominance condition of part 3 of Theorem 6.3. Notice that the implication between closure conditions mentioned in Theorem 6.3 translates exactly to logical implication ($\Rightarrow$) between clauses, i.e., given compatibles c and c', the closure condition of c implies the closure condition of c' iff $closure(c) \Rightarrow closure(c')$. What follows gives a useful characterization of the formula $closure(c) \Rightarrow closure(c')$.

Theorem 6.5 *Given compatibles c and c′,*

$$\forall i' \; \exists i \; [disjunct(c,i) \Rightarrow disjunct(c',i')]$$

iff

$$closure(c) \Rightarrow closure(c').$$

Proof: Substituting $x = i, x' = i', F(x) = disjunct(c,i), F'(x') = disjunct(c',i')$ in Lemma 6.1, one gets that $\forall i' \; \exists i \; [disjunct(c,i) \Rightarrow disjunct(c',i')]$ iff $\forall i \; disjunct(c,i) \Rightarrow \forall i' \; disjunct(c',i')$. The latter is by definition $closure(c) \Rightarrow closure(c')$. □

The following theorem gives a useful characterization of the formula $disjunct(c',i') \Rightarrow disjunct(c,i)$.

Theorem 6.6 *Given compatibles c′, c and inputs i′, i,*

$$\forall o' \; \exists o \; [d_{c',i',o'} \Rightarrow d_{c,i,o}]$$

iff

$$disjunct(c',i') \Rightarrow disjunct(c,i).$$

Proof: Substituting $x = o, x' = o', F(x) = d_{c,i,o}, F'(x) = d_{c',i',o'}$ into Lemma 6.2, one gets $\forall o' \; \exists o \; [d_{c',i',o'} \Rightarrow d_{c,i,o}]$ iff $\exists o' \; d_{c',i',o'} \Rightarrow \exists o \; d_{c,i,o}$. The latter is by definition $disjunct(c',i') \Rightarrow disjunct(c,i)$. □

By substituting Theorem 6.6 into Theorem 6.5, we have expressed the implication between closure conditions of two compatibles (i.e., part 3 of Theorem 6.3) in terms of a logic formula on the next state sets from the two compatibles.

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

Theorem 6.7 *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 6.8 *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*[8]*, or*

2. *$d \subseteq c$, or*

3. *$d \supseteq d'$ if d' is part of $disjunct(c,i)$.*

Proof: (1) If d is a reset state, the covering condition would imply this closure condition expressed by d, and thus the latter is not needed and, even more, the closure condition is vacuously true. (2) d expresses the condition that if we choose c, a selected compatible must contain the state set d. If $d \supseteq c$, c is such a compatible, and the closure condition is vacuously true. (3) $d \supseteq d'$ means $d \Rightarrow d'$. The disjunction of d and d' is simply d', so d can be omitted from $disjunct(c,i)$. □

[8]This condition is valid only if a unique reset state is given.

6.5 Implicit Generation of Compatibles

Since the definition of compatibles for a PNDFSM is similar to that for an ISFSM, the compatibles C and prime compatibles PC are generated using implicit techniques similar to the ones proposed in [10]. First we compute the transition relation τ' between sets of states, using the implicit procedure in Section 5.5.

Theorem 6.9 *The set C of compatibles of an NDFSM can be found by the following fixed point computation:*

- $\tau_0(i, c, c') = \exists o \; \tau'(i, c, c', o)$
- *Initially assume all subsets of states to be compatible:* $C_0(c) = 1$,
- *By Theorem 6.1,*
  - $\tau_{k+1}(i, c, c') = \tau_k(i, c, c') \cdot C_k(c')$
  - $C_{k+1}(c) = \forall i \; \exists c' \; \tau_{k+1}(i, c, c')$

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

6.6 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 a relation, e.g. $\tau(i, c, d)$. The following computation will prune away next state sets that are not necessary according to Theorem 6.8, and the result is represented by the following relation B.

Theorem 6.10 *The disjunctive conditions can be computed by the following relation B:*

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

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

Proof: The first equation corresponds to condition (1) and (2) of Theorem 6.8. 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 $R(d) + (c \supseteq d)$, *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, and $B(c, i, d)$ represents the pruned set of disjunctive clauses: d is in the relation B with (c, i) iff d is part of the disjunctive clause for c under i. □

The following theorem computes the set of disjunctive clauses, that are used to express the closure conditions.

Theorem 6.11 *If*

$$D(c', i', c, i) = \forall d' \; \{B(c', i', d') \Rightarrow \exists d \, [B(c, i, d) \cdot (d' \subseteq d)]\},$$

then $disjunct(c', i') \Rightarrow disjunct(c, i)$. The set of prime compatibles can be computed by:

$$PC(c) = C(c) \cdot \nexists c' \{C(c') \cdot [\exists s \, (R(s) \cdot (s \subseteq c)) \Rightarrow \exists s' \, (R(s') \cdot (s' \subseteq c'))] \cdot \forall i' \, \exists i \; D(c, i, c', i') \cdot (c' \supset c)\}$$

Proof: To evaluate $disjunct(c', i') \Rightarrow disjunct(c, i)$, it is sufficient to check $\forall o' \in$ outputs at c' under i' $\exists o \in$ outputs at c under i $[d_{c',i',o'} \supseteq d_{c,i,o}]$, by Theorem 6.6 and 6.4. Quantification over outputs o possible at c under i has the same effect as quantification of d in $B(c, i, d)$. Similarly, $\forall o'$ is the same as $\forall d'$ in $B(c', i', d')$. 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 6.3. For condition (1), $\exists s\ (R(s) \cdot (s \subseteq c)) \Rightarrow \exists s'\ (R(s') \cdot (s' \subseteq c'))$ requires that $(c' \cap R) \neq \emptyset$ if $(c \cap R) \neq \emptyset$. By Theorem 6.5, condition (2) of Theorem 6.3 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. They will be used in the next subsection to set up the binate rows of the covering table.

Theorem 6.12 *Given a compatible c, the inputs i's associated with c which are involved in non-trivial disjunctive clauses are expressed by the following relation E:*

$$E(c,i) = \not\exists i'\ [(i \neq i') \cdot D(c,i',c,i)] + \exists i' cproject_i(D(c,i',c,i) \cdot D(c,i,c,i'))$$

And the corresponding pruned set of disjunctive clauses is given by relation I:

$$I(c,i,d) = B(c,i,d) \cdot PC(c) \cdot E(c,i) \cdot \neg Tuple_0(d)$$

Proof: By Theorem 6.11, given compatible c, $disjunct(c,i') \Rightarrow disjunct(c,i)$ if $D(c,i',c,i)$. The first term $\not\exists i'\ [(i \neq i') \cdot D(c,i',c,i)]$ deletes all pairs (c,i) such that there is an input i',$(i' \neq i)$ 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. A unique (c,i) out of each set of "co-implying" pairs is chosen as representative by the BDD *cproject* operator. And the representative is added back to relation E 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. □

6.7 Implicit Binate Table Covering

Selection of prime compatibles is performed by an implicit binate covering solver presented in [10]. 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 a closure condition of a compatible $c \in PC$ under an input i. And the covering condition of 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 $\emptyset$ 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 $I(c,i,d)$. If (c,i) labels a binate row, the expression $\exists d\ [(p \supseteq d) \cdot I(c,i,d)]$ evaluates to true iff the table entry at the intersection of the row labeled (c,i) and the column labeled p is a 1, i.e., the row can be satisfied if next state set d is contained in a 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 iff $\exists s\, [\emptyset(c) \cdot \emptyset(i) \cdot R(s) \cdot (s \subseteq p)]$.

As a summary, the inference rules for table entries given a row (c, i) and a column p are:

$$1(c,i,p) \overset{\text{def}}{=} \exists d\, [(p \supseteq d) \cdot I(c,i,d)] + \exists s\, [\emptyset(c) \cdot \emptyset(i) \cdot R(s) \cdot (s \subseteq p)]$$

$$0(c,i,p) \overset{\text{def}}{=} (p = c)$$

7 Implicit State Minimization of NDFSM's

7.1 Exact Algorithms

We have not thought yet of an implicitization of the algorithm shown in Section 4. It does not follow as a straightforward extension of the implicit algorithm for PNDFSM's presented in Section 6, since there is one more level of of set construction complexity, given that generalized compatibles are already sets of sets of states.

As an alternative, suggested in [17], one can convert any NDFSM to a PNDFSM by an implicit determinization (via subset construction) step as described in Section 5.5 and then apply to the latter 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; this can hurt the efficiency of the implicit PNDFSM state minimizer whose computations are on a support whose cardinality is linearly proportional to the number of states of the PNDFSM. 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 be also stressed that we do not have yet good sources of general NDFSM's in sequential synthesis, while the work in [17] has shown the pivotal importance of PNDFSM's in the synthesis of interconnected FSM's.

7.2 Heuristic Algorithms

It has been shown that exact minimization of NDFSM's requires computation of the generalized compatibles, instead of the usual compatibles. If we restrict our attention only to the set of compatibles (against generalized compatibles), the algorithm given in Section 6 for exact state minimization of PNDFSM's will still serve as a heuristic algorithm for NDFSM minimization.

8 Experimental Results

We have implemented an implicit algorithm for exact state minimization of PNDFSM's in a program called ISM2. Prime compatibles and the binate table are generated according to the algorithm described above; then a minimum cover of the table is found by our implicit binate covering solver presented in [10]. We perform and report experiments on the complete set of examples obtained by Watanabe in [17]. 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 [18]. Our problem is to find a minimum state machine contained in the E-machine.

Watanabe's minimer, PND_REDUCE, does not compute prime compatibles but finds all compatibles instead. It then solves the binate table using an explicit binate solver available in the logic synthesis package SIS.

Table 1 summarizes the results of PNDFSM state minimization. For each PNDFSM, we report the number of states in the original PNDFSM, the number of states of the solution, the size of the binate table for PND_REDUCE and for ISM2, and the overall run time for state minimization for PND_REDUCE and ISM2. All run times are reported in CPU seconds on a DECstation 5000/260 with 440 Mb of memory. For both programs, timeout is set at 10000 seconds of CPU time, and spaceout at 440Mb of memory.

Out of the 30 examples, PND_REDUCE 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. Our program can handle more examples 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 4844.7 seconds. For pm50, it found a first solution with 4 states in 150.7 seconds, and the minimum one with 3 states in 7309.5 seconds, while optimality was concluded in 49181 seconds after the complete branch-and-bound tree was searched.

Note that each compatible results in a column of the binate table by PND_REDUCE whereas ISM2 has one column for each prime compatible. The fact that most examples have very few prime compatible 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.

9 Conclusions

In this paper we have presented both theoretical and practical contributions to the problem of exploring contained behaviors and selecting one with minimum number of states for classes of NDFSM's. In particular we have contributed:

1. A theoretical solution to the problem of exact state minimization of general NDFSM's, based on the proposal of a notion of generalized compatibles. This gives an algorithmic frame 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 [19]. We could solve exactly all the problems of the benchmark used in [17] (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.

We are currently working also on the problem of selecting out of an NDFSM 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 achieved by computing compatibles (as classically defined and then extended in [19, 4]). Each closed collection of compatibles is a contained DFSM and viceversa. 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 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 as those that we have presented are a winning tool to support synthesis algorithms.

References

[1] K. Brace, R. Rudell, and R. Bryant. Efficient implementation of a BDD package. In *The Proceedings of the Design Automation Conference*, pages 40–45, June 1990.

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

[3] O. Coudert and J.C. Madre. Implicit and incremental computation of prime and essential prime implicants of Boolean functions. In *The Proceedings of the Design Automation Conference*, pages 36–39, June 1992.

[4] M. Damiani. Nondeterministic Finite-State Machines and Sequential Don't Cares. In *European Conference on Design Automation*, pages 192–198, 1994.

[5] M. Davio, J.-P. Deschamps, and A. Thayse. *Discrete and Switching Functions*. Georgi Publishing Co. and McGraw-Hill International Book Company, 1978.

[6] E.Cerny. A compositional transformation for formal verification. In *The Proceedings of the International Conference on Computer Design*, pages 240–244, October 1991.

[7] 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.

[8] 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.

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

[10] 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.

[11] B. Lin, O. Coudert, and J.C. Madre. Symbolic prime generation for multiple-valued functions. In *The Proceedings of the Design Automation Conference*, pages 40–44, June 1992.

[12] 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.

[13] R. Narasimhan. Minimizing incompletely specified sequential switching functions. *IRE Transactions on Electronic Computers*, EC-10:531–532, September 1961.

[14] M. Paull and S. Unger. Minimizing the number of states in incompletely specified state machines. *IRE Transactions on Electronic Computers*, September 1959.

[15] 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.

[16] R. Rudell and A. Sangiovanni-Vincentelli. Multiple-valued minimization for PLA optimization. *IEEE Transactions on Computer-Aided Design*, CAD-6:727–750, September 1987.

[17] Y. Watanabe. Logic optimization of interacting components in synchronous digital systems. *Ph.D. Thesis, Tech. Report No. UCB/ERL M94/32*, April 1994.

[18] Y. Watanabe and R. K. Brayton. The Maximum Set of Permissible Behaviors for FSM networks. In *IEEE International Conference on Computer-Aided Design*, pages 316–320, November 1993.

[19] Y. Watanabe and R. K. Brayton. State Minimization of Pseudo Non-Deterministic FSM's. In *European Conference on Design Automation*, pages 184–191, 1994.

PNDFSM	# states in PNDFSM	# states in sol.	table size (row x col)		CPU time (seconds)	
			PND_REDUCE	ISM2	PND_REDUCE	ISM2
L3	17	2		10 x 4	timeout	17.4
am9	13	1		1 x 1	timeout	2.3
ax4	11	1	26 x 28	1 x 1	0.7	0.7
ax7	20	2	334 x 308	20 x 6	15.6	7.6
bx7	23	2	254 x 216	20 x 6	9.9	9.0
damiani	5	3	21 x 24	17 x 10	0.1	1.3
e4at2	14	1		1 x 1	timeout	1.1
e4bp1	11	1	1064 x 995	1 x 1	308.1	0.8
e4t1	6	1	103 x 120	1 x 1	0.7	0.3
e69	8	1	551 x 501	1 x 1	10.5	0.3
e6tm	21	1		1 x 1	timeout	3.1
ex10	13	1	23 x 28	1 x 1	0.5	0.6
ex12	13	1	1451 x 1019	1 x 1	16149.1	0.8
mc9	4	1	7 x 11	1 x 1	0.1	0.1
mt51	16	1		1 x 1	timeout	3.8
mt52	9	1	256 x 639	1 x 1	39.4	0.8
pm03	15	1	1203 x 1019	1 x 1	1751.1	0.8
pm04	79	1		1 x 1	spaceout	120.6
pm11	9	1	331 x 395	1 x 1	29.7	1.3
pm12	7	1	8 x 19	1 x 1	0.4	0.4
pm31	22	1		1 x 1	spaceout	3.6
pm33	21	1		1 x 1	timeout	6.6
pm41	33	≤ 9		12050 x 4774	spaceout	4844.7*
pm50	22	3		1249 x 515	timeout	49181
s3p1	38	1		1 x 2	spaceout	915.8
s3t2	36	1		389 x 18	timeout	39.9
tm01	10	1	476 x 767	1 x 1	40.9	0.8
tm02	7	1	155 x 211	1 x 1	3.1	0.4
tm31	9	1	125 x 113	1 x 1	1.1	0.3
tm32	9	2	106 x 143	37 x 9	1.2	2.9

* best solution before timeout

Table 1: State Minimization of PNDFSM's