Copyright © 1996, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

STATE MINIMIZATION OF FSM'S WITH IMPLICIT TECHNIQUES

by

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

Memorandum No. UCB/ERL M96/17

17 April 1996

COVER PAGE

STATE MINIMIZATION OF FSM'S WITH IMPLICIT TECHNIQUES

by

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

Memorandum No. UCB/ERL M96/17

17 April 1996

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

State Minimization of FSM's with Implicit Techniques

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

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

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

February 5, 1996

Abstract

This paper surveys algorithms for exact state minimization of finite state machines. Incompletely specified and pseudo-nondeterministic FSM's are considered. An exact state minimum FSM can be found by computing sets of compatible states and selecting a minimum closed cover of compatibles. An explicit enumeration of compatibles is not always feasible in examples of practical interest. We present techniques based on representation of sets with binary decision diagrams; this enables us to solve exactly larger examples. Sets of compatibles of cardinality up to 2^{1500} were computed and the corresponding binate tables solved.

1 Introduction

Finite state machines (FSM's) are a common formalism to describe sequential systems. Incompletely specified FSM's (ISFSM's) and pseudo-nondeterministic FSM's (PNDFSM's) are very useful classes of FSM's, because they capture collections of input-output behaviors, any of which is a valid implementation of the original specification. FSM's and PNDFSM's arise in the practice of sequential synthesis. The choice of which input-output behavior to implement may be dictated by different criteria. A common one is the minimization of the number of states of the deterministic automaton corresponding to the chosen behavior. The chosen behavior may be required to satisfy other conditions, as implementability of the chosen deterministic FSM within a network of FSM's [19].

It has been shown [14, 8] that in case of ISFSM's and PNDFSM's all contained behaviors can be explored by means of collections of compatibles, called closed sets. To explore closed sets, one must compute maximal compatibles, prime compatibles, class sets of compatibles and other related sets and subsets of sets of states. The number of compatibles can be exponential in the number of states of the original FSM. This may be a problem for computations based on the explicit enumeration of compatibles and their subsets. An alternative is to represent the characteristic functions of these sets by means of binary decision diagrams (BDD's). Then various sets of compatibles can be computed with BDD-based techniques. Moreover, also the binate table that models the selection of compatibles is represented and manipulated

implicitly. The implicit techniques described here can be applied also to other problems of logic synthesis and combinatorial optimization.

The remainder is organized as follows. Section 2 introduces the problem of exact state minimization of FSM's, while representations based on BDD's are described in Section 3. Implicit computations for state minimization are presented, respectively, in Section 4 for ISFSM's, and in Section 5 for PNDFSM's. The restriction to a minimum Moore behavior is treated in Section 6.

2 Definitions

2.1 Finite State Machines

In this section, we shall define different classes of finite state machines (FSM's) used in this paper, and the problem of state minimization of FSM's.

Definition 2.1 *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. 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.*

Definition 2.2 *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 [1].

Definition 2.3 *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 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 Λ.

An NDFSM is an **incompletely specified FSM** (ISFSM) if 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.

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

In the standard literature, no reset state is specified for an ISFSM, and it is assumed that all states can potentially be selected as a reset state for implementation. The same is assumed here, and this is reflected in covering conditions defined later. In addition, an unspecified next state is traditionally not represented in the next state relation Δ. i.e., if the next state is not specified for present state s and input i, there is no state s' such that $\delta(i, s, s') = 1$. This assumption is made in subsequent definitions and computations.

Definition 2.4 *A* **Moore NDFSM** *is 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 non-empty set of next states. Λ is the output relation defined as a characteristic function $\Lambda : S \times O \rightarrow B$ where each present state is related to a non-empty set of outputs. $R \subseteq S$ represents the set of reset states.*

The key fact to notice is that the output is associated with the present state, i.e., all transitions that leave a given present state carry a common output [2]. When the output depends from the present state *and* the input, the FSM is said to be a **Mealy** machine.

Definition 2.5 *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 (σ_i, σ_o) where $\sigma_i \in I^*$, $\sigma_o \in O^*$ and $|\sigma_i| = |\sigma_o|$.*

Definition 2.6 *A* **trace set** *is a set of traces.*

Definition 2.7 *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. 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.

2.2 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**.

[2] This definition is the one given by Moore in [15] and preferred when modeling an hardware system [20]. However, it is common also a "dual" definition where the output is associated with the next state [11, 7]. This second definition enjoys the nice property that it is always possible to convert a Mealy machine into a Moore machine.

[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 $\mathcal{L}(M|_{s_0})$.

Definition 2.8 *An NDFSM $M = \langle S, I, O, T, R\rangle$* **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.,*

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

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 2.9 *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 non-deterministic finite automata described in classical automata textbooks [7]. Here we require a minimum state implementation which is behaviorally contained in the specification, while the classical problem requires an NDFSM which represents the same set of behaviors as the original NDFSM but has the fewest number of states.

Closed covers are a way to explore all behaviors contained in a PNDFSM.

Definition 2.10 *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 2.11 *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 2.12 *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 2.13 *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.

It has been proved in [8] that one can explore all behaviors contained in a PNDFSM by finding all closed covers of the PNDFSM.

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

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

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

Theorem 2.1 *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 [8], is a companion and an extension of Theorem 2.1. It proves the optimality of exact state minimization algorithms which find minimum closed covers.

Theorem 2.2 *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'.*

Closed covers can be found by restricting the attention to compatible sets of states.

Definition 2.14 *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 2.1 *[8] Every element of a closed cover is an output compatible.*

Definition 2.15 *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.*

Lemma 2.2 *[8] Every element of a closed cover is a compatible.*

The following theorem serves as an equivalent, constructive definition of compatibles.

Theorem 2.3 *[8] 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.*

3 Implicit Techniques

3.1 Binary Decision Diagrams

Basics on binary decision diagrams are found in [3, 2].

Definition 3.1 *A* **binary decision diagram** *(BDD) is a rooted, directed acyclic graph. Each nonterminal vertex v is labeled by a Boolean variable $var(v)$. Vertex v has two outgoing arcs, $child_0(v)$ and $child_1(v)$. Each terminal vertex u is labeled 0 or 1.*

Definition 3.2 *A BDD is* **ordered** *if there is a total order $\prec$ over the set of variables such that for every nonterminal vertex v, $var(v) \prec var(child_0(v))$ if $child_0(v)$ is nonterminal, and $var(v) \prec var(child_1(v))$ if $child_1(v)$ is nonterminal.*

Definition 3.3 *A BDD is* **reduced** *if*

1. *it contains no vertex v such that $child_0(v) = child_1(v)$, and*
2. *it does not contain two distinct vertices v and v' such that the subgraphs rooted at v and v' are isomorphic.*

Definition 3.4 *A* **reduced ordered binary decision diagram** *(ROBDD) is a BDD which is both reduced and ordered.*

3.2 Implicit Set Manipulation

In [8] it is presented a full-fledged theory on how to represent and manipulate sets using a BDD-based representation. It extends the notation used in [12]. This theory is especially useful for applications where sets of sets need to be constructed and manipulated.

Given a ground set G of cardinality less or equal to 2^n, any **subset** S can be represented in a Boolean space B^n by a unique Boolean function $\chi_S : B^n \rightarrow B$, which is called its **characteristic function** [4], such that:

$$\chi_S(x) = 1 \text{ if and only if } x \text{ in } S.$$

In other words, a subset is represented in *positional-set* or *positional-cube* notation form [7], using n Boolean variables, $x = x_1 x_2 \ldots x_n$. The presence of an element 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 element s_k is not a member of the set. One Boolean variable is needed for each element because the element can either be present or absent in the set. As an example, for $n = 6$, the set with a single element s_4 is represented by 000100 and the set $s_2 s_3 s_5$ is represented by 011010. The elements s_1, s_4, s_6 which are not present correspond to 0's in the positional-set.

A set of subsets of G can be represented by a Boolean function, whose minterms correspond to the single subsets. In other words, a set of sets is represented as a set S of positional-sets, by a characteristic function $\chi_S : B^n \rightarrow B$ as:

$$\chi_S(x) = 1 \text{ if and only if the set represented by the positional-set } x \text{ is in the set } S \text{ of sets.}$$

Any **relation** $\mathcal{R}$ between a pair of Boolean variables can also be represented by a characteristic function $\mathcal{R} : B^2 \rightarrow B$ as:

$$\mathcal{R}(x, y) = 1 \text{ if and only if } x \text{ is in relation } \mathcal{R} \text{ to } y.$$

$\mathcal{R}$ can be a one-to-many relation over the two sets in B. These definitions can be extended to any relation $\mathcal{R}$ between n Boolean variables, and can be represented by a characteristic function $\mathcal{R} : B^n \rightarrow B$ as:

$$\mathcal{R}(x_1, x_2, \ldots, x_n) = 1 \text{ if and only if the } n\text{-tuple } (x_1, x_2, \ldots, x_n) \text{ is in relation } \mathcal{R}.$$

From an operator Op that acts on two sets of variables $x = x_1 x_2 \ldots x_n$ and $y = y_1 y_2 \ldots y_n$ one obtains a relation $(x \; Op \; y)$ (as a characteristic function) of pairs of positional-sets. The containment and maximal operators are shown next as examples.

Lemma 3.1 *The* **containment relation** *evaluates to true if the set of objects represented by x contains the set of objects represented by y, and can be computed as:*

$$(x \supseteq y) = \prod_{k=1}^{n} y_k \Rightarrow x_k$$

where $x_k \Rightarrow y_k = \neg x_k + y_k$ designates the Boolean implication operation.

Lemma 3.2 *The* **maximal** *of a set χ of subsets is the set containing subsets in χ not strictly contained by any other subset in χ, and can be computed as:*

$$Maximal_x(\chi) = \chi(x) \cdot \nexists y \, [(y \supset x) \cdot \chi(y)].$$

[7] Called also *1-hot encoding*.

The following lemma, proved in [8], shows a quantifier-free computation of the operator *maximal*.

Lemma 3.3 *Given a set of positional-sets $\chi(x)$ and an array of Boolean variables x, the maximal of positional-sets in χ with respect to x can be computed by the recursive BDD operator $Maximal(\chi, 0, x)$:*

```
Maximal(χ, k, x) {
    if (χ = 0) return 0
    if (χ = 1) return ∏_{i=k}^{n} x_i
    M_0 = Maximal(χ_{x̄_k}, k + 1)
    M_1 = Maximal(χ_{x_k}, k + 1)
    return ITE(x_k, M_1, M_0 · ¬M_1)
}
```

3.3 Implicit FSM Representation

A state transition graph (STG) is commonly used as the internal representation of FSM's in sequential synthesis systems, such as SIS [17]. A limitation of STG's is the fact that they are a two-level form of representation where state transitions are stored explicitly, one by one. This may degrade the performance of conventional optimization algorithms on large FSM's.

Assume that the given FSM has n states. To perform state minimization, one must represent and manipulate efficiently sets of states (such as compatibles) and sets of sets of states (such as sets of compatibles). Therefore *1-hot encoding* is used for the states of the FSM [8]. If inputs (outputs, respectively) of the FSM are specified symbolically, they can be represented as a multi-valued symbolic variable i (o, respectively) where each value of i (o, respectively) represents an input (output, respectively) combination. For compactness of representation, we used for these variables a *logarithmic encoding*, i.e. an m-valued variable is represented with $\log_2 m$ Boolean variables. The fact that different multi-valued variables use different encodings is not a problem as long as they are used consistently. However if inputs (outputs, respectively) of the FSM are already given in encoded form, each encoded bit of inputs (outputs, respectively) is represented by a single Boolean variable.

4 State Minimization of ISFSM's

An exact algorithm for state minimization consists of two steps: the generation of various sets of compatibles, and the solution of a binate covering problem. The generation step involves identification of sets of states called compatibles which can potentially be merged into a single state in the minimized machine. Unlike the case of DFSM's, where state equivalence partitions the states, compatibles for ISFSM's may overlap. As a result, the number of compatibles can be exponential in the number of states [16], and the generation of the whole set of compatibles can be a challenging task.

The covering step is to choose a minimum subset of compatibles satisfying covering and closure conditions, i.e., to find a minimum closed cover. The covering conditions require that every state is contained in at least one chosen compatible. The closure conditions guarantee that the states in a chosen compatible are mapped by any input sequence to states contained in a chosen compatible.

[8] An alternative explained in [8] is to represent any set of sets of states (i.e., set of state sets) implicitly as a single 1-hot encoded MDD, and manipulate the state sets symbolically all at once. Different sets of sets of states can be stored as multiple roots with a single shared 1-hot encoded MDD.

4.1 Implicit Generation of Compatibles

In this section, we describe implicit computations to find sets of compatibles required for exact state minimization of ISFSM's.

To generate compatibles, incompatibility relations between pairs of states are derived first from the given output and transition relations of an ISFSM.

Lemma 4.1 *The set of output incompatible pairs, $\mathcal{OICP}(y,z)$, can be computed as:*

$$\mathcal{OICP}(y,z) = Tuple_1(y) \cdot Tuple_1(z) \cdot \exists i \; \nexists o \, [\Lambda(i,y,o) \cdot \Lambda(i,z,o)] \tag{1}$$

In the above and subsequent formulas, we will mix notations between relations and their corresponding characteristic functions. Strictly speaking if we would have used the characteristic function notation, the above formula would have been more clumsy:

$$\begin{aligned} \mathcal{OICP}(y,z) = 1 \quad \text{if and only if} \quad & (Tuple_1(y) = 1) \cdot (Tuple_1(z) = 1) \\ & \cdot \exists i \; \nexists o \, [(\Lambda(i,y,o) = 1) \cdot (\Lambda(i,z,o) = 1)] \end{aligned}$$

Lemma 4.2 *The set of incompatible pairs is the least fixed point of $\mathcal{ICP}$:*

$$\mathcal{ICP}(y,z) = \mathcal{OICP}(y,z) + \exists i,u,v \, [\Delta(i,y,u) \cdot \Delta(i,z,v) \cdot \mathcal{ICP}(u,v)]$$

and can be computed by the following iteration:

$$\begin{aligned} \mathcal{ICP}_0(y,z) &= \mathcal{OICP}(y,z) \\ \mathcal{ICP}_{k+1}(y,z) &= \mathcal{ICP}_k(y,z) + \exists i,u,v \, [\Delta(i,y,u) \cdot \Delta(i,z,v) \cdot \mathcal{ICP}_k(u,v)] \end{aligned} \tag{2}$$

The iteration can terminate when $\mathcal{ICP}_{k+1} = \mathcal{ICP}_k$ and the set of incompatible pairs is $\mathcal{ICP}(y,z) = \mathcal{ICP}_k(y,z)$.

So far we established incompatibility relationships between pairs of states. The following definition introduces sets of states of arbitrary cardinalities.

Lemma 4.3 *The set of incompatibles can be computed as:*

$$\mathcal{IC}(c) = \exists y,z \, [\mathcal{ICP}(y,z) \cdot (c \supseteq y \cup z)] \tag{3}$$

Lemma 4.4 *The set of compatibles, $\mathcal{C}(c)$, can be computed as:*

$$\mathcal{C}(c) = \neg Tuple_0(c) \cdot \neg \mathcal{IC}(c)$$

4.2 Implicit Generation of Prime Compatibles and Closure Conditions

To set up the covering problem, we also need to compute the closure conditions for each compatible. This is done by finding the class set of a compatible, i.e., the set of next states implied by a compatible.

Definition 4.1 *A set of states d_i is an* **implied set** *of a compatible c for input i if d_i is the set of next states from the states in c on input i.*

Lemma 4.5 *The implied set (in singleton form) of a compatible c for input i can be defined by the relation $\mathcal{F}(c,i,n)$ which evaluates to 1 if and only if on input i, n is a next state from state p in compatible c:*

$$\mathcal{F}(c,i,n) = \exists p \, [\mathcal{C}(c) \cdot (c \supseteq p) \cdot \Delta(i,p,n)] \tag{4}$$

Note that the implied next states are represented here as singleton states in $\mathcal{F}(c, i, n)$. All singletons n in relation with a compatible c and an input i can be combined into a single positional-set, for later convenience. This positional-set representation of implied sets associates each compatible c with a set of implied sets d.

Lemma 4.6 *The implied sets d (in positional-set form) of a compatible c for all inputs are computed by the relation $\mathcal{CI}(c, d)$ as:*

$$\mathcal{CI}(c, d) = \exists i \, [\exists n(\mathcal{F}(c, i, n)) \cdot Union_{n \to d}(\mathcal{F}(c, i, n))]$$

Definition 4.2 *An implied set d of a compatible c is in its* **class set** *if*

1. *d has more than one element, and*
2. *$d \not\subseteq c$, and*
3. *$d \not\subseteq d'$ if $d' \in$ class set of c.*

We can ignore any implied set which contains only a single state, because its closure condition is satisfied if the state is covered by some chosen compatible. Also if $d \subseteq c$, the closure condition is satisfied by the choice of c. Finally, if the closure condition corresponding to d' is stronger than that of d, the implied set d is not necessary.

Lemma 4.7 *The class set of a compatible c is defined by the relation $\mathcal{CCS}(c, d)$ which evaluates to 1 if and only if the implied set d is in the class set of compatible c:*

$$\mathcal{CCS}(c, d) = \neg Tuple_1(d) \cdot (c \not\supseteq d) \cdot Maximal_d(\mathcal{CI}(c, d))$$

To solve exactly the covering problem, it is sufficient to consider a subset of compatibles called prime compatibles. As proved in [6], at least one minimum closed cover consists entirely of prime compatibles.

Definition 4.3 . *A compatible c'* **dominates** *a compatible c if*

1. *$c' \supset c$, and*
2. *class set of $c' \subseteq$ class set of c.*

i.e., c' dominates c if c' covers all states covered by c, and the closure conditions of c' are a subset of the closure conditions of c. As a result, compatible c' expresses strictly less stringent conditions than compatible c. Therefore c' is always a better choice for a closed cover than c, thus c can be excluded from further consideration.

Lemma 4.8 *The prime dominance relation is given by:*

$$Dominate(c', c) = (c' \supset c) \cdot Contain_d(\mathcal{CCS}(c, d), \mathcal{CCS}(c', d))$$

Definition 4.4 *A* **prime compatible** *is a compatible not dominated by another compatible.*

Lemma 4.9 *The set of prime compatibles is given by:*

$$\mathcal{PC}(c) = C(c) \cdot \nexists c' \, [C(c') \cdot Dominate(c', c)]$$

4.3 Implicit Generation of the Covering Table

Once the set of (non-essential) prime compatibles is generated, the problem of exact state minimization can be solved as a binate table covering problem. In this section, we shall describe how such a binate table can be generated. To keep with our stated objective, the binate table is also represented implicitly. We describe an implicit representation of the covering table, that adroitly exploits how row and columns were implicitly computed. A description of how the binate table is then solved implicitly can be found in [9].

We do not represent (even implicitly) the elements of the table, but we make use only of a set of row labels and a set of column labels, each represented implicitly as a BDD. They are chosen so that the existence and value of any table entry can be readily inferred by examining its corresponding row and column labels. This choice allows us to define all table manipulations needed by the reduction algorithms in terms of operations on row and column labels and to exploit all the special features of the binate covering problem induced by state minimization (for instance, each row has at most one 0, etc.). A similar technique could be applied to various binate covering problems that arise in logic synthesis, with a suitable encoding of the rows and columns.

Definition 4.5 *A column is labeled by a positional-set p. The set of* **column labels** *C is obtained by prime generation as $C(p) = \mathcal{PC}(p)$.*

Besides distinguishing one row from another, each row label must also contain information regarding the positions of 0 and 1's in the row. Each row label r consists of a pair of positional-sets (c, d). Since there is at most one 0 in the row, the label of the column p intersecting it in a 0 is recorded in the row label by setting its c part to p. If there is no 0 in the row, c is set to the empty set, $Tuple_0(c)$. Because of Definition 4.7 for row labels, the columns intersecting a row labeled $r = (c, d)$ in a 1 are labeled by the prime compatibles p that contain d.

Definition 4.6 *The table entry at the intersection of a row labeled by $r = (c, d) \in R$ and a column labeled by $p \in C$ can be inferred by:*

the table entry is a 0 iff relation $0(r, p) \stackrel{\text{def}}{=} (p = c)$ is true,

the table entry is a 1 iff relation $1(r, p) \stackrel{\text{def}}{=} (p \supseteq d)$ is true.

Definition 4.7 *The set of* **row labels** *R is given by:*

$$R(r) = \mathcal{PC}(c) \cdot CCS(c, d) + Tuple_0(c) \cdot Tuple_1(d)$$

The closure conditions associated with a prime compatible p are that if p is included in a solution, each implied set d in its class set must be contained in at least one chosen prime compatible. A binate clause of the form $(\overline{p} + p_1 + p_2 + \cdots + p_k)$ has to be satisfied for each implied set of p, where p_i is a prime compatible containing the implied set d. The labels for binate rows are given succinctly by $\mathcal{PC}(c) \cdot CCS(c, d)$. There is a row label for each (c, d) pair such that $c \in \mathcal{PC}$ is a prime compatible and d is one of its implied sets in $CCS(c, d)$. This row label consistently represents the binate clause because the 0 entry in the row is given by the column labeled by the prime compatible $p = c$, and the row has 1's in the columns labeled by p_i wherever $(p_i \supseteq d)$.

The covering conditions require that each state be contained by some prime compatible in the solution. For each state $d \in S$, a unate clause has to be satisfied which is of the form $(p_1 + p_2 + \cdots + p_j)$ where the p_i's are the prime compatibles that contain the state d. By specifying the unate row labels to be $Tuple_0(c) \cdot Tuple_1(d)$, we define a row label for each state in $Tuple_1(d)$. Since the row has no 0, its c part must be set to $Tuple_0(c)$. The 1 entries are correctly positioned at the intersection with all columns labeled

by prime compatibles p_i which contain the singleton state d [9]. Since no minimal cover S can contain a compatible contained in another compatible of S [13], one could introduce a new collection of clauses, one for each pair of compatibles p_1 and p_2 such that $p_1 \supset p_2$, each stating that at most one of the two can be chosen $(\overline{p_1} + \overline{p_2})$.

5 State Minimization of PNDFSM's

Explicit algorithms for exact state minimization of PNDFSM's have been proposed by Watanabe *et al.* in [20], by Damiani in [5], and by Kam *et al.* in [10].

An algorithm for PNDFSM state minimization is more complicated than one for ISFSM state minimization [9] because the definition of compatibles and the closure conditions are more complex. By Theorem 2.2, the state minimization problem of PNDFSM's can be reduced to the problem of finding minimum closed covers. Because of Lemma 2.2, an exact state minimization algorithm only needs to generate compatibles. The next step 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 the selection of a minimum closed cover of compatibles.

5.1 Implicit Generation of Compatibles

First, we outline the differences between the state minimization algorithm of PNDFSM's and the state minimization algorithm for ISFSM's [9]. 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, as shown in [10]. As a result, the set of compatibles cannot be generated from the set of incompatible pairs as for ISFSM's. As we cannot generate compatibles from incompatible pairs, we have to start with output compatibles (i.e., state sets) of arbitrary cardinalities.

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 5.1 *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 2.3,* $\tau_{k+1}(i, c, c') = \tau_k(i, c, c') \cdot C_k(c')$, $C_{k+1}(c) = \forall i\, \exists c'\, \tau_{k+1}(i, c, c')$.

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 $\tau(i, c, c') = \tau_{j+1}(i, c, c') \cdot C_j(c)$.

[9] Every closed cover of an ISFSM whose states are all reachable must cover all the states. We can say that the covering clauses express a property of the solution that speeds up the search. Alternatively one could impose that only the reset state is covered and let the search procedure find that a solution covers all states.

The computations for closure conditions and prime dominance are more complicated than in case of ISFSM's. In [10] it is described how to compute the set of primes $PC)c)$ and the set of disjunctive clauses $I(c, i, d)$, where we represent each next state set as a positional-set d.

5.2 Implicit Generation of the Covering Table

To use the implicit binate 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:
$0(c, i, p) \stackrel{\text{def}}{=} (p = c)$,
$1(c, i, p) \stackrel{\text{def}}{=} \exists d\, [(p \supseteq d) \cdot I(c, i, d)] + \exists s\, [\emptyset(c) \cdot \emptyset(i) \cdot R(s) \cdot (s \subseteq p)]$. Notice that these inference rules are more complex than those for the case of ISFSM's. To handle the former rules we upgraded the implicit binate solver used for ISFSM's state minimization and described in [9]. The more general implicit binate solver and the issues involved are described carefully in [8, 18].

6 State Minimization of PNDFSM's for Moore Behavior

It has been shown in [19] that the set of permissible behaviors at a node of a network of FSM's can be represented by a PNDFSM. When choosing a state minimum DFSM among all permissible behaviors, one must ensure that it is possible to implement it in the network without introducing unsafe combinational cycles. A way to enforce this requirement is to restrict the minimum state FSM to be a Moore DFSM.

Watanabe introduced in [20, 19] a more general notion of compatible, as a pair (q, f) where q is a set of compatible states as defined in state minimization of PNDFSM's and f is a function from $B^{|U|}$ to $B^{|V|}$. For Moore behaviors the pair (q, f) becomes a pair (q, v), where q represents a set of compatible states, and v is an output minterm common to outgoing transitions of the states $\in q$. This more complex notion

of compatible, to which we refer as **compatible pair**, allows the formulation of state minimization as a standard binate covering problem. As an alternative formulation, in [8] it is shown how the problem can be modelled without associating output minterms to compatibles [10].

6.1 Implicit Generation of Compatible Pairs

It was observed in [1] that a state that cannot produce a common output for all inputs is not involved in any compatible in any reduced Moore machine. As a result, such states can be deleted from the original machine along with all transitions leading to such pruned states, before the generation of compatibles.

The set of compatibles, i.e., (q, v) pairs, is computed using the following fixed point computation:

$$\begin{aligned} C_0(q,v) &= 1 \\ C_{k+1}(q,v) &= \forall u\, \exists q'\, [(\exists v'\, C_k(q',v')) \cdot T^{det}(u,q,q',v)] \end{aligned}$$

where T^{det} is the transition relation over subsets of states.

First we assume that every pair (q, v), where q is any state set and v is any minterm, is a candidate compatible and so it is in C_0. After the first iteration, $C_1(q, v)$ captures all the state sets q that are output compatibles, i.e., each state in q can produce the same output minterm v for all inputs. During the k-th iteration, (q, v) will be in C_{k+1} if and only if on output v, for every input u, there exists a next state set $q' \in C_k$ from the state set q. The iteration terminates when $C_{k+1} = C_k$ and then the set of compatibles is given by $C(q, v) = C_k(q, v)$.

6.2 Implicit Generation of the Covering Table

We want to construct the set of column labels Col and the set of row labels Row in a format suitable to the implicit specialized binate solver presented in [9]. Let columns be labeled by variables p and rows by pairs of variables (c, d). The numbers of Boolean variables used for p, c and d are the same. At the intersection of row (c, d) and column p, the table entry is a 1 if and only if $p \supseteq d$, and the table entry is a 0 if and only if $p = c$. As no entry can be both 0 and 1 simultaneously, the case $c \supseteq d$ is ruled out.

Each column p in our table is a compatible, i.e., a pair (q, v), where empty states sets q are removed as meaningless:

$$Col(qv) = C(q, v) \cdot \neg\emptyset(q).$$

Each row label consists of two parts c and d. To match the width of p, the row label has a field $c = (q, v)$ and a field $d = (r, w)$, where q and r represent sets of states in positional notation, and v and w represent output minterms. The w variables are set to the empty set $\emptyset(w)$, since they are never used. Row_{unate} represents the unate rows corresponding to the covering conditions of the reset states, i.e., a compatible (q, v) is in relation Row_{unate} with (r, w) if and only if r is a reset state contained in the state set q:

$$Row_{unate}(qv, rw) = \emptyset(q) \cdot \emptyset(v) \cdot reset\_state(r) \cdot \emptyset(w).$$

Row_{binate} represents the binate rows corresponding to the closure conditions. In particular, (q, v) must be a compatible ($C(q, v) = 1$), and r must be a state set in a compatible ($\exists w\ C(r, w)$), and they are in the relation $Row_{binate}(qv, rw)$ only if r is the set of next states of q under output minterm v (computed by $\exists u\, [T^{det}(u, q, r, v)]$):

$$Row_{binate}(qv, rw) = [\exists u\, T^{det}(u, q, r, v)] \cdot C(q, v) \cdot [\exists w\, C(r, w)] \cdot \emptyset(w).$$

Finally we collect all these clauses as the set of Row labels, and make sure that $c \not\supseteq d$:

$$Row(qv, rw) = [Row_{unate}(qv, rw) + Row_{binate}(qv, rw)] \cdot (qv \not\supseteq rw).$$

[10] The state minimization algorithm presented in this section is joint work with Yosinori Watanabe.

7 Conclusions

We have presented algorithms to compute implicitly a minimum state behavior contained in an ISFSM or a PNDFSM, including the case when the minimum machine is restricted to be Moore. Compatibles, maximal compatibles, prime compatibles and implied sets are all represented by the characteristic functions of relations implemented with BDD's. Similarly, the final step of covering a binate table is solved with an implicit solver. The only explicit dependence is on the number of states of the initial problem. Experiments with a variety of benchmark sets show that implicit techniques allow to compute compatible sets of cardinality up to 2^{1500} and to solve the correspondingly large binate tables. Experiments and applicability are discussed in [9, 10, 8]. The techniques described here can be applied to similar problems in logic synthesis and combinatorial optimzation.

References

[1] A. Aziz, F. Balarin, R. K. Brayton, M. D. Di Benedetto, A. Saldanha, and A. L. Sangiovanni-Vincentelli. Supervisory control of finite state machines. In *Proceedings of International Conference on Computer-Aided Verification*, 1995.

[2] 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.

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

[4] E. Cerny. Characteristic functions in multivalued logic systems. *Digital Processes*, vol. 6:167–174, June 1980.

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

[6] 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.

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

[8] 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.

[9] 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.

[10] T. Kam, T. Villa, R. Brayton, and A. Sangiovanni-Vincentelli. Implicit state minimization of non-deterministic fsm's. In *The Proceedings of the International Conference on Computer Design*, October 1995.

[11] Z. Kohavi. *Switching and Finite Automata Theory.* McGraw-Hill Book Company, New York, New York, second edition, 1978.

[12] 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.

[13] F. Luccio. Extending the definition of prime compatibility classes of states in incomplete sequential machine reduction. *IEEE Transactions on Computers*, C-18(6):537–540, June 1969.

[14] R. E. Miller. *Switching theory. Volume I: sequential circuits and machines.* J. Wiley and & Co., N.Y., 1965.

[15] E. Moore. Gedanken-experiments on sequential machines. In C. Shannon and J. McCarthy, editors, *Automata Studies*. Princeton University Press, 1956.

[16] F. Rubin. Worst case bounds for maximal compatible subsets. *IEEE Transactions on Computers*, pages 830–831, August 1975.

[17] E. Sentovich, K. Singh, C. Moon, H. Savoj, R. Brayton, and A. Sangiovanni-Vincentelli. Sequential Circuit Design Using Synthesis and Optimization. In *The Proceedings of the International Conference on Computer Design*, pages 328–333, October 1992.

[18] T. Villa. *Encoding Problems in Logic Synthesis.* PhD thesis, University of California, Berkeley, May 1995.

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

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