Copyright © 1993, 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.

INPUT DON'T CARE SEQUENCES IN FSM NETWORKS

by

Huey-Yih Wang and Robert K. Brayton

Memorandum No. UCB/ERL M93/64

6 August 1993

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Input Don't Care Sequences in FSM Networks *

Huey-Yih Wang Robert K. Brayton
Department of Electrical Engineering and Computer Sciences
University of California at Berkeley
Berkeley, CA 94720

August 6, 1993

Abstract

We present an approach to compute all input don't care sequences for a component in an FSM network with an arbitrary topology. In a cascade FSM network, Kim and Newborn's (K-N) procedure [13] exactly computes all input don't care sequences for the driven machine. However, for a component in a general FSM network the exact computation of input don't care sequences is unsolved. We demonstrate that this problem can be reduced to one for a cascade circuit. This reduction uses the notion of an abstract driving machine. In some cases, the exact computation and exploitation of these sequences may be too expensive. We propose methods to compute subsets of input don't care sequences. We discuss the implementation of these algorithms using implicit methods. We also present approximate methods for managing the complexity of large FSM networks. Finally, we give some preliminary results on small networks.

*This project was supported by DARPA under contract number JFBI90-073 and NSF under contract number EMC-84-19744.

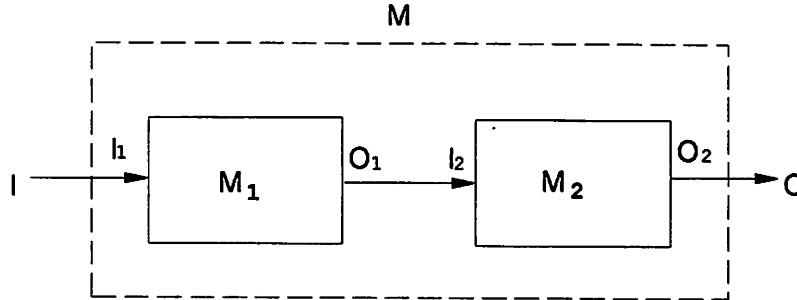

Figure 1: $\mathcal{N}_1$: A cascade circuit of two FSM's

1 Introduction

When one circuit is connected to another, the controllability of its inputs and the observability of its outputs are reduced. This phenomenon has been studied thoroughly in combinational circuits in which limited controllability can be represented by don't cares; on other hand, limited observability requires Boolean relations (observability relations) or symbolic relations [6, 2, 15, 22]. In many cases, better implementations can be obtained by exploiting this information.

Similarly, sequential don't cares play an important role in the optimization of sequential circuits. Several approaches have been proposed; for example, in [16], unreachable or equivalent states are used in the optimization of an isolated sequential circuit. Damiani *et al.* [9] introduced synchronous relations to deal with the logic optimization of sequential circuits with pipelined latches. In this approach a circuit implementation is given as the starting point. On the other hand, a transition relation can be used to represent an isolated finite state machine (FSM); thus, symbolic information, i.e. unencoded machines, can be manipulated.

In sequential circuits, *don't cares sequences* need to be considered. There are two kinds; output and input sequences. In this paper, we deal with the latter. Consider the cascade machine in Figure 1, where M_1 is the driving machine and M_2 the driven machine[1]. Unger [26] observed that M_2, when driven by M_1, may possess more unspecified transitions than as an isolated machine, and proposed a method to approximate and exploit a subset of this information. Recently, Devadas [10] proposed a different but similar procedure. Kim and Newborn [13] proposed an elegant complete solution. However, for a two-way-communication network of FSM's, $\mathcal{N}_2$, as shown in Figure 2, the exact computation is not well known. Rho *et al.* [20] suggested applying Kim and Newborn's (K-N) procedure iteratively between M_1 and M_2 until a fixed point is reached which represents the input don't care sequences for both M_1 and M_2. The upper bound on the number of iterations is unknown.

In this paper, we survey previous related work, and provide an improved understanding of input don't care sequences. We propose a complete solution without iteration for the problem in general FSM networks. Then, we propose methods to compute subsets of these sequences and discuss implementation details using implicit techniques. We consider several special situations which make the computation easy, and give implicit methods to check for these situations. We discuss approximation techniques for managing the complexity in large FSM networks. Finally, we present some preliminary results on small FSM networks.

2 Preliminaries

2.1 Finite Automata

A deterministic finite automaton (DFA), $\mathcal{A}$, is a quintuple $(K, \Sigma, \delta, q_0, F)$ where K is a finite set of states, Σ an alphabet, $q_0 \in K$ the initial state, $F \subseteq K$ the set of final states, and δ the transition function, $\delta : K \times \Sigma \rightarrow K$. A nondeterministic finite automaton (NFA), $\mathcal{A}$, is a quintuple $(K, \Sigma, \delta, q_0, F)$ where δ, the transition relation, is a finite subset of $K \times \Sigma^* \times K$, and Σ^* the set of all strings obtained by concatenating zero or more strings from Σ. An input string is accepted by $\mathcal{A}$ if it ends up in one of final states of $\mathcal{A}$. The language accepted by $\mathcal{A}$, $\mathcal{L}(\mathcal{A})$, is the set of strings it accepts.

[1] In this paper, we only consider synchronous FSM networks with known initial states.

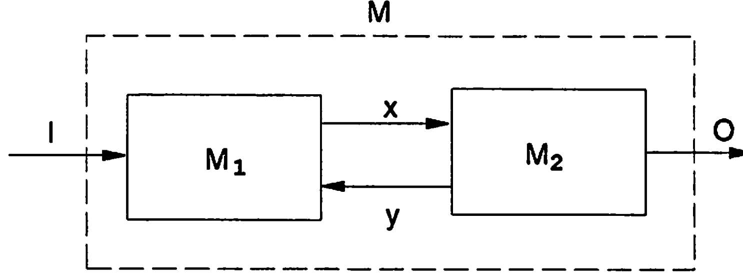

Figure 2: $\mathcal{N}_2$: A two-way-communication network of FSM's.

2.2 Finite State Machines

A finite state machine (FSM), M, is a six-tuple $(I, O, Q, \delta, \lambda, q_0)$, where I is a finite input alphabet, O a finite output alphabet, Q a finite set of states, δ the transition function, λ the output function, and q_0 the initial state. A machine is of *Moore* type if λ does not depend on the inputs, and *Mealy* otherwise. An FSM can be represented by a state transition graph (STG). A machine in which transitions under all input symbols from every state are defined is a *completely specified machine*; in other words, both δ and λ are complete functions. Otherwise, a machine is *incompletely specified.*

A *distinguishing sequence* for two states $q_1, q_2 \in Q$ is a sequence of inputs such that when applied to M, the last input produces different outputs depending whether M started at q_1 or q_2. In a completely specified machine M, two states q_1 and q_2 are *equivalent* if there is no distinguishing sequence. In an incompletely specified machine M, two such states q_1 and q_2 are *compatible.* A sequence S_1 is said to *contain* another S_2 if S_2 appears in S_1.

A *cascade* of FSM's M_1 and M_2, denoted $M_1 \rightarrow M_2$, is shown in Figure 1. M_1 is called the *driving machine*, M_2 the *driven machine.*

2.3 Set Computation and Operators

Let B designate the set $\{0, 1\}$.

Definition 1 *Let E be a set and $S \subseteq E$. The* **characteristic function** *of S is the function $\chi_S : E \rightarrow B$ defined by $\chi_S(x) = 1$ if $x \in S$, and $\chi_S(x) = 0$, otherwise.*

Definition 2 *Let $f : B^n \rightarrow B$ be a Boolean function, and $x = \{x_1, ..., x_k\}$ a subset of the input variables. The* **existential quantification** *(***smoothing***) of f by x, with f_a denoting the* **cofactor** *of f by literal a is defined as :*

$$\begin{aligned} \exists_{x_i} f &= f_{x_i} + f_{\overline{x_i}} \\ \exists_x f &= \exists_{x_1} ... \exists_{x_k} f \, . \end{aligned}$$

Definition 3 *Let $f : B^n \rightarrow B^m$ be a Boolean function, $S_1 \subseteq B^n$ and $S_2 \subseteq B^m$. The* **image** *of S_1 by f is $f(S_1) = \{y \in B^m | y = f(x), x \in S_1\}$. $f(B^n)$ is the* **range** *of f. The* **inverse image** *of S_2 by f is $f^{-1}(S_2) = \{x \in B^n | f(x) = y, y \in S_2\}$.*

Definition 4 *Let $f : B^n \rightarrow B$ be a Boolean function, only depending on a subset of variables $y = \{y_1, ..., y_k\}$. Let $x = \{x_1, ..., x_k\}$ be another subset of variables, describing another subspace of B^n of the same dimension. The* **substitution** *of variables y by variables x in f is the function of x obtained by substituting x_i for y_i in f :*

$$(\theta_{y,x} f)(y) = f(x) \;\; \text{if} \;\; x_i = y_i \;\; \text{for all} \;\; 1 \leq i \leq k.$$

Definition 5 *Let $f : B^n \rightarrow B^m$ be a Boolean function. The relation (characteristic relation) associated with f, $F : B^n \times B^m \rightarrow B$, is defined as $F(x, y) = \{(x, y) \in B^n \times B^m | y = f(x)\}$. Equivalently, in terms of Boolean operations :*

$$F(x, y) = \prod_{1 \leq i \leq m} (y_i \equiv f_i(x)) \, .$$

We can use F to obtain the image by f of $S_1 \subseteq B^n$, by computing the projection on B^m of the set $F \bigcap (S_1 \times B^m)$:

$$f(S_1)(y) = \exists_x (F(x, y) \cdot S_1(x)) .$$

Similarly, the inverse image by f of $S_2 \subseteq B^m$ can be computed as :

$$f^{-1}(S_2)(x) = \exists_y (F(x, y) \cdot S_2(y)) .$$

Reduced ordered binary decision diagrams (BDD's) [3] are well suited to represent the characteristic functions of subsets of a set, and efficient algorithms [1, 3] exist to manipulate them to perform all standard Boolean operations. As a result, the above set operations can be done efficiently.

2.4 Multiple-Valued Functions

Let $X_1, X_2, \cdots X_n$ be multiple-valued variables ranging over sets $P_1, P_2, \cdots, P_n$ respectively, where $P_i = \{0, ..., p_i - 1\}$, and p_i are positive integers. A multiple-valued function f is a mapping

$$f : P_1 \times P_2 \times ... \times P_n \rightarrow B .$$

Let S_i be a subset of P_i, and $X_i^{S_i}$ represent the characteristic function

$$X_i^{S_i} = \begin{cases} 0 & \text{if } X_i \notin S_i . \\ 1 & \text{if } X_i \in S_i . \end{cases}$$

$X_i^{S_i}$ is called a *literal* of the variable X_i. If $|S_i| = 1$, this literal is a minterm of X_i. A *product term* or a *cube* is a Boolean product (AND) of literals. A *sum-of-products* is a Boolean sum (OR) of product terms. An *implicant* of a function f is a product term which does not contain any minterm in the OFF-set ($f^{-1}(0)$) of the function. A *prime implicant* of f is an implicant not contained in any other implicant of f.

Let a symbolic variable s assume values from $S = \{s_0, ..., s_{m-1}\}$. It can be represented by a multiple-valued variable, X, restricted to $P = \{0, ..., m - 1\}$, where each symbolic value of s maps onto a unique integer in P.

We can use multiple-valued decision diagrams (MDD's) [23] to manipulate multiple-valued functions just like BDD's for Boolean functions. Furthermore, similar operations, such as existential, and universal quantification, and substitution, etc., are well defined in the MDD framework [23]. In the sequel, we just use the term BDD to interchangeably refer to characteristic functions of multiple-valued variables.

2.5 Implicit State Reachability Computation

The reachable states can be computed efficiently using implicit state enumeration techniques introduced by Coudert *et al.* [7]. These techniques are widely used in FSM verification [7, 8, 25], and in design verification [4, 24]. This approach is based on representing a set of states by a characteristic function which can be manipulated effectively using BDD's. In the following, we represent a finite state machine implicitly by a characteristic function using BDD's.

Definition 6 *The* **transition relation** *of a finite state machine* $M = (I, O, Q, \delta, \lambda, q_0)$ *is a function* $T : I \times Q \times Q \times O \rightarrow B$ *such that* $T(i, p, n, o) = 1$ *if and only if state* n *can be reached in one state transition from state* p *and produce output* o *when input* i *is applied.*

A predicate transformer is a monotone function operating on the power set of a finite set. The set of states $R(p)$ containing the states reachable from a given set of initial states $I(p)$ can be viewed as the least fixed point containing $I(p)$ of the function :

$$\mathcal{F} : c(p) \mapsto c(p) + \theta_{n,p} \exists_{i,p,o} (T(i, p, n, o) \cdot c(p)) .$$

At a fixed point, $R(p)$ satisfies :

$$R(p) = R(p) + \theta_{n,p} \exists_{i,p,o} (T(i, p, n, o) \cdot R(p)) .$$

The least fixed point of $\mathcal{F}$ can be computed [4] as the limit of the following sequences :

$$R_0(p) = I(p) \quad (1)$$

$$R_{m+1}(p) = R_m(p) + \theta_{n,p} \exists_{i,p,o} (T(i, p, n, o) \cdot R_m(p)) \quad (2)$$

$$R_\infty(p) = R_m(p) \text{ if } R_{m+1}(p) = R_m(p) . \quad (3)$$

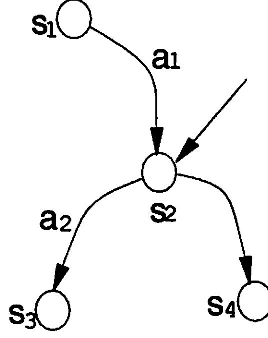

Figure 3: A part of the STG of M_2

2.6 Compatible Projection Operator

The compatible projection operator is defined in [14] and can be manipulated efficiently using BDD's.

Definition 7 *Let $y_1 \prec \cdots \prec y_n$ be an ordering of Boolean variables. The distance between two vertices $\alpha \in B^n$ and $\beta \in B^n$ is defined as [7, 25]*

$$d(\alpha, \beta) = \sum_{i}^{n} |\alpha_i - \beta_i| 2^{n-i} .$$

Using the above distance metric, a total ordering of all the vertices of a Boolean space relative to some reference vertex α can be defined; $order(x) = d(\alpha, x)$.

Definition 8 *Given $\alpha \in B^n$, $C \subseteq B^n$, the* **closest interpretation** *of α in C for a given variable ordering is defined as [14]*

$$\mathcal{P}(\alpha, C) = argmin_{x \in C} d(\alpha, x) .$$

The definition of closest interpretation $\mathcal{P}$, relative to a reference vertex α, is unique for a given variable ordering.

Definition 9 *For a relation, $\mathcal{R} \subseteq B^r \times B^n$, and $\alpha \in B^n$, the closest interpretation of α relative to $\mathcal{R}$ (called* **compatible projection** *in [14]) is :*

$$\perp(\alpha, \mathcal{R}) = \{(x, y) | (x, y) \in \mathcal{R}, y = \mathcal{P}(\alpha, \mathcal{R}_x)\} .$$

Conceptually, the $\perp$ operator selects a unique minterm y for each minterm x defined in the relation $\mathcal{R}$. Thus, $\perp(\alpha, \mathcal{R})$ results in the characteristic function of a function defined on the domain $\exists_y \mathcal{R}(x, y)$; $\perp(\alpha, \mathcal{R}) : \exists_y \mathcal{R}(x, y) \times B^n \rightarrow B$. Also, the $\perp$ operator can be generalized to symbolic relations represented by MDD's.

3 Previous Work

First we discuss input don't care sequences. In this section, we consider the cascade machine $M_1 \rightarrow M_2$ in Figure 1. Part of an STG of M_2 is shown in Figure 3. Consider the transitions from s_1 to s_2 and from s_2 to s_3. When M_2 does not interact with other machines, $(s_1 s_2 s_3)$ is a possible sequence of transitions. However, when M_2 is driven by M_1, this sequence may not happen. Thus, we can regard the input string $(a_1 a_2)$ causing these transitions as a don't care sequence starting at s_1. We cannot determine whether $(s_1 s_2 s_3)$ is a possible sequence by looking at M_2 in isolation, but such information is useful; we may get a smaller number of states for M_2 in the cascade $M_1 \rightarrow M_2$, than when M_2 is in isolation.

3.1 Unger's Procedure

Unger [26] observed that when one FSM is driven by another, there are more *input-incompletely-specified don't cares (unspecified transitions)* than for an isolated machine. We restate a theorem in [10].

Theorem 3.1 *Given a machine M, a set of don't care sequences DC_{seq}, and the set of all distinguishing sequences of two states q_1 and q_2, DIS_{seq}. Suppose that each sequence in DIS_{seq} contains at least one sequence of DC_{seq}. Then q_1 and q_2 are compatible under the DC_{seq}.*

Based on this theorem, one could create a procedure to produce all distinguishing sequences for every pair of states in machine M_2 and then check for the containment condition. Any pair of states satisfying this check is compatible and can be merged. Although it provides some theoretical clarification, this approach is potentially very time consuming, since each pair of states may have many distinguishing sequences that have to be found for every pair.

A more efficient approach was proposed in [26, 10] which employs *explicit state splitting* to represent more of the incompletely specified information. This approach generates M_2' with states $| Q_2' | \geq | Q_2 |$. M_2' has more transitions that are not specified than those of M_2. However, the number of states of a minimum state machine of M_2' will not exceed that of a minimum state machine of M_2.

3.2 K-N Procedure

Kim and Newborn [13] proposed an elegant approach which solves the problem of computing input don't care sequences for a driven machine in a cascade. The procedure is :

1. Construct an NFA $\mathcal{A}'$ to accept the language produced by machine M_1. This can be achieved by removing the input part in the STG of M_1, and assigning every state of M_1 as a final state. For a state s, if there are output symbols not emitted from it, a transition is inserted from s to the *dead state* d with those symbols. The dead state d is the only nonaccepting state. Thus $\mathcal{A}'$ is completely specified but nondeterministic.

2. Convert $\mathcal{A}'$ to a minimized completely specified DFA $\mathcal{A}$. This can be done by using the subset construction [18] and state minimization for DFA [12]. Note that efficient state minimization for completely specified machines can be used, since the subset construction produces a completely specified machine.

3. A modified machine M_2' is constructed as follows : construct $M_2 \times \mathcal{A}$ and delete any transition to a state that contains the dead state d in its subset. M_2' is deterministic but possibly incompletely specified.

The key idea is that sequences not produced by M_1 are the input don't care sequences for M_2, and these are converted into unspecified transitions of a modified machine M_2'. The K-N procedure indeed captures *all* input don't care sequences for M_2. In the next subsection, this is discussed in more detail. It can be seen that the state splitting method of Devadas is a subset of this, but Devadas' procedure is explicit and length limited, whereas the K-N procedure is implicit and not length limited.

3.3 Input Incompletely Specified Don't Cares vs. Input Don't Care Sequences

Consider an incompletely specified FSM P. The *input-incompletely-specified don't cares (unspecified transitions)* for P may be interpreted as characterizing that a given input symbol never occurs when this machine is in a particular state. Thus any input sequence is *forbidden* that drives P to exhibit unspecified behavior. The complement of forbidden input sequences is the set of *allowable* input sequences. Let $\mathcal{L}(P^i)$ denote the set of all allowable input sequences. Since $\mathcal{L}(P^i)$ is a regular language, we can construct an automaton $\mathcal{A}$ to accept it as follows. For each transition in the STG of P, remove the output part. Each state is designated as a final state. For a state s with unspecified input symbols, create a transition edge from s to the *dead state* d with those unspecified symbols. The dead state d is the only nonaccepting state. Consequently, any input sequence not accepted by $\mathcal{A}$ causes P to produce unspecified behavior. Thus from an incompletely specified machine P we can construct an automaton $\mathcal{A}$ which accepts all allowable input sequences to P.

From the above discussion, any incompletely specified FSM corresponds to a regular language. Conversely, the basic idea behind the K-N procedure is that the sequences produced by M_1 is a regular language which can be implicitly expressed with input-incompletely-specified don't cares. This conversion is done in step 3 of the K-N procedure. From this discussion, we conclude that

- Given any incompletely specified machine P, we can construct an automaton which accepts the regular set $\mathcal{L}$ of input sequences for which P is always specified.

- Given any regular language $\mathcal{L}$, we can modify any machine P to an incompletely specified one P' whose set of input sequences for which P' is specified is $\mathcal{L}$.

Thus incompletely specified machines and regular input languages are equivalent.

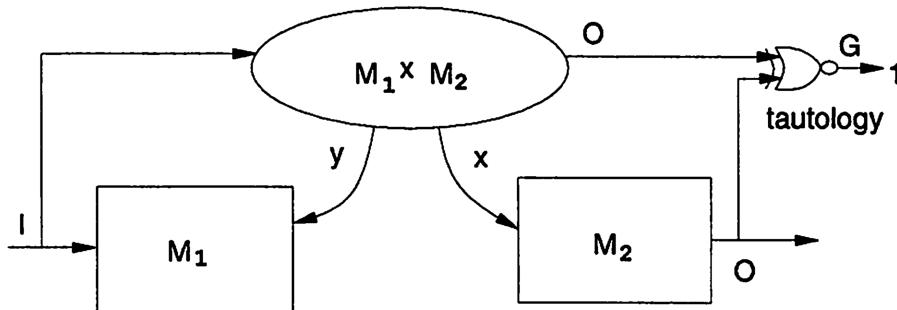

Figure 4: $\mathcal{N}_2{}'$: An equivalent one-way-communication FSM network to $\mathcal{N}_2$.

3.4 Transformation from NFA to DFA

Unfortunately, the worst case complexity for the transformation from an NFA to a DFA (i.e. from $\mathcal{A}'$ to $\mathcal{A}$) is exponential in the number of states [18]. Furthermore, even if $\mathcal{A}$ can be built in a reasonable time, the resultant product machine M_2' may have a large number of states before state minimization is performed.

In [21], a sophisticated heuristic was proposed to reduce the number of states of the DFA $\mathcal{A}$ by sacrificing some don't care information. This simplification process was called *summarizing the DFA*. However, to apply this, one may need to have $\mathcal{A}$ first. So the subset construction to transform $\mathcal{A}'$ to $\mathcal{A}$ is still required. Later in this paper, we propose a simplification process applied during the transformation from $\mathcal{A}'$ to $\mathcal{A}$, which controls the state explosion.

4 FSM Networks with Arbitrary Topologies

In this section, we demonstrate that the problem of computing and exploiting input don't care sequences for a component in an FSM networks with an arbitrary topology can be reduced to one for a cascade circuit.

4.1 Exact Computation

Intuitively, computation of input don't care sequences for a component in an FSM network of arbitrary topology is much more complicated than for a cascade circuit. Nevertheless, it is not theoretically harder. In this subsection, we demonstrate that the K-N procedure can be applied directly to an arbitrary FSM network topology.

Consider a two-way-communication FSM network $\mathcal{N}_2$, $M_1 \leftrightarrow M_2$, as shown in Figure 2. We want to compute input don't care sequences for M_1. The suggestion made in [20] is as follows.

> Let automaton $\mathcal{A}_i$ be defined on the same alphabet X. The DFA's produced for a given machine in the loop by repeated application of the K-N procedure form a sequence $(\mathcal{A}_i)$ of automata such that $\mathcal{L}(\mathcal{A}_i) \subseteq \mathcal{L}(\mathcal{A}_{i+1})$. Each iteration through the loop corresponds to an application of a recurrence relation of the form $\mathcal{A}_{i+1} = f(\mathcal{A}_i)$, with $\mathcal{L}(\mathcal{A}_0) = \emptyset$. If we proceed until we get the least fixed point, we have the whole set of input don't care sequences for these two machines. The fixed point is guaranteed to exist.

The upper bound on the number of iterations required to attain the fixed point is not known and may be large. At each iteration, an equivalence checking of two automata is required. Moreover, the number of states of $\mathcal{A}_i$ at some iteration i may grow too large.

Now consider another FSM network $\mathcal{N}_2{}'$ as shown in Figure 4. We show that this circuit produces the same I/O sequences as those in network $\mathcal{N}_2$. Since x, y in network $\mathcal{N}_2{}'$ are produced by the composite machine $M_1 \times M_2$, the sequences happening in x and y should be the same as those in network $\mathcal{N}_2$. As a result, x and y in network $\mathcal{N}_2{}'$ will drive M_1 and M_2 in network $\mathcal{N}_2{}'$ to exhibit the same behavior as in network $\mathcal{N}_2$. Consequently, we will have the same I/O sequences as in network $\mathcal{N}_2$, and the output G of $\mathcal{N}_2{}'$ is a tautology. By this construction, we transform a two-way-communication network to an equivalent one-way-communication network. Here, the equivalent driving machine to M_1 and M_2 is $M_1 \times M_2$. We can apply the same procedures we used in a cascade circuit directly to compute input don't care sequences for M_1 and M_2 simultaneously, since M_1 and M_2 can be regarded as being driven independently by $M_1 \times M_2$, instead of communicating with each other. By such a construction, we do not suffer the iterative executions of the K-N procedure as suggested in [20].

When we want to compute input don't care sequences for a component, say M_2, in an FSM network of an arbitrary topology, we can always lump the other components together, and call the resultant machine M_1. Then, it is either a one-way-communication model in Figure 1 or a two-way-communication model in Figure 2. Consequently, we are able to perform the computation of input don't care sequences for a component in an FSM network with an arbitrary topology.

Based on the above explanation, we introduce the notion of an **abstract driving machine** in the computation of input don't care sequences in an FSM network. For example, the abstract driving machine to M_2 in Figure 1 is M_1, while the abstract driving machine to M_2 in Figure 2 is $M_1 \times M_2$. The abstract driving machine for a component in an FSM network is the composite machine of all components in this network, i.e. the network itself. However, if a component M_2 is in a one-way communication with other components as in Figure 1, its abstract driving machine will reduce to M_1. Then steps 1 and 2 of the K-N procedure can be used to compute the exact input don't care sequences.

An abstract driving machine itself may be a nondeterministic FSM[2]; however, this does not affect the computation of input don't care sequences in the K-N procedure. Consequently, we may start with a network of machines some of which are nondeterministic (e.g. the environment may be one of the machines). The K-N procedure works anyway.

4.2 Exploitation

In this subsection, we demonstrate that input don't care sequences for a component in a general FSM network can be exploited in the same way as in a cascade circuit. This is not discussed in [13, 10, 21].

Consider the two-way-communication circuit $M_1 \leftrightarrow M_2$ in Figure 2. Let $\mathcal{L}_1$ be possible sequences over alphabet X produced by M_1 in isolation, i.e. when input y is unrestricted, and $\mathcal{L}_2$ possible sequences over alphabet Y produced by M_2 in isolation, i.e. when input x is unrestricted. Let $\mathcal{L}_x$, $\mathcal{L}_y$, $\mathcal{L}_{xy}$ be possible sequences over alphabets X, Y, and $X \times Y$ in $M_1 \leftrightarrow M_2$, respectively. $\overline{\mathcal{L}_x}$ and $\overline{\mathcal{L}_y}$ are input don't care sequences for M_2 and M_1, respectively. Let $M_2{'}$ be the modified machine of M_2 with $\overline{\mathcal{L}_x}$ as unspecified transitions.

Let a state-minimized machine of $M_2{'}$ be $M_2{''}$. The specified behavior of $M_2{'}$ is preserved in $M_2{''}$, i.e. the behavior of $M_2{''}$ restricted to input sequences $\mathcal{L}_x$ is the same as that of M_2. For unspecified input symbols of a state s in $M_2{'}$, state minimization procedures can exploit them by assigning transitions from s to any next states and with any output symbols. Therefore, the set of output sequences produced by $M_2{''}$ in isolation, $\mathcal{L}_2{''}$, may be different from $\mathcal{L}_2$. Let $\mathcal{L}_x{''}$, $\mathcal{L}_y{''}$, $\mathcal{L}_{xy}{''}$ be possible sequences over alphabets X, Y, and $X \times Y$ in $M_1 \leftrightarrow M_2{''}$, respectively. If $\mathcal{L}_x = \mathcal{L}_x{''}$, then $\mathcal{L}_y = \mathcal{L}_y{''}$ and $\mathcal{L}_{xy} = \mathcal{L}_{xy}{''}$. This is because the behavior of M_2 restricted to input sequences $\mathcal{L}_x$ is preserved after performing state minimization on $M_2{'}$. Thus, $\mathcal{L}_x = \mathcal{L}_x{''}$ implies that $M_1 \leftrightarrow M_2{''}$ has the same input/output behavior as that of $M_1 \leftrightarrow M_2$. If $\mathcal{L}_x \neq \mathcal{L}_x{''}$, the input/output behavior of the whole circuit may not be preserved. This is because $\overline{\mathcal{L}_x}$ is assumed to be impossible sequences in order to be exploited by state minimization procedures.

Lemma 4.1 $\mathcal{L}_x = \mathcal{L}_x{''}$.

Proof Our assumption is that $M_1 \leftrightarrow M_2$ is synchronous and there is no direct information feedback loop through x and y. The initial transition of the composite system T can be excited by the initial state of the whole circuit and external input i. If $M_1 \leftrightarrow M_2$ is a closed system, i.e. there is no external input i, the first transition is excited by the initial state. If y is a Moore-type output of M_2, y is still a Moore-type output of $M_2{''}$ with respect to input sequences $\mathcal{L}_x$.

Case 1 : One of x and y is of Moore type.

Let x_i and y_i be symbols of the alphabets X and Y, respectively. Let $(x_1 \cdots x_n)$ be a sequence in $\mathcal{L}_x$, and the corresponding output sequence of M_2 be $(y_1 \cdots y_n)$. First, we prove $\mathcal{L}_x \subseteq \mathcal{L}_x{''}$. x_1 is the initial output value if x is a Moore-type output. If y is a Moore-type output, x_1 is produced after a transition starting from the inital state of $M_1 \leftrightarrow M_2{''}$, which is the same as that of $M_1 \leftrightarrow M_2$. Therefore, initially symbol x_1 should be able to be produced by M_1. Symbol x_1 must drive $M_2{''}$ to produce y_1. This is because $M_2{''}$ restricted to input sequences $\mathcal{L}_x$ has the same behavior as that of M_2. Symbol y_1 should be able to drive M_1 to produce symbol x_2, since M_1 is kept unchanged. Input sequence $(x_1 x_2)$ must drive $M_2{''}$ to produce $(y_1 y_2)$ since $(x_1 x_2)$ is a sequence in $\mathcal{L}_x$. By applying the same rationale repeatedly, we can conclude that $(x_1, \ldots, x_n)$ is a sequence in $\mathcal{L}_x{''}$. Next, we prove $\overline{\mathcal{L}_x} \subseteq \overline{\mathcal{L}_x{''}}$. Without loss of generality, consider that $(x_1 \cdots x_n)$ is a sequence in $\mathcal{L}_x$, but $(x_1 \cdots x_n x_{n+1})$ is a sequence in $\overline{\mathcal{L}_x}$. This means that $(x_1 \cdots x_n x_{n+1})X^*$ are sequences in $\overline{\mathcal{L}_x}$. For input sequence $(x_1 \cdots x_n)$, the corresponding output sequence produced by $M_2{''}$ is $(y_1 \cdots y_n)$. Symbol y_n is the input to M_1 for next transition. Suppose on the contrary that $(x_1 \cdots x_n x_{n+1})$ is not a sequence in $\overline{\mathcal{L}_x{''}}$. For input sequence $(y_1 \cdots y_n)$, x_{n+1} should be able to be produced by M_1 as the last output symbol. This implies that $(x_1 \cdots x_n x_{n+1})$ should be a sequence in $\mathcal{L}_x$. This is a contradiction. Thus,

[2] In this paper, a nondeterministic FSM refers to a collection of permissible FSM's.

$(x_1 \cdots x_n x_{n+1})X^*$ are sequences in $\overline{\mathcal{L}_x''}$. Therefore, we have $\overline{\mathcal{L}_x} \subseteq \overline{\mathcal{L}_x''}$.

Case 2 : Both of x and y are of Moore type.

Let x_1 and y_1 be the values in x and y at $t = 0^+$, respectively. Let $(x_1 \cdots x_n)$ be a sequence in $\mathcal{L}_x$, and the corresponding output sequence of M_2 be $(y_2 \cdots y_n y_{n+1})$. First, we prove $\mathcal{L}_x \subseteq \mathcal{L}_x''$. Symbol x_1 must drive M_2'' to produce y_2. This is because M_2'' restricted to input sequences $\mathcal{L}_x$ has the same behavior as that of M_2. Symbol y_1 must be able to drive M_1 to produce x_2, since M_1 is unchanged. Input sequence $(x_1 x_2)$ must drive M_2'' to produce $(y_2 y_3)$ since $(x_1 x_2)$ is a sequence in $\mathcal{L}_x$. By applying the same rationale repeatedly, we can conclude that $(x_1, \ldots, x_n)$ is a sequence in $\mathcal{L}_x''$. Next, we prove $\overline{\mathcal{L}_x} \subseteq \overline{\mathcal{L}_x''}$. Without loss of generality, consider that $(x_1 \cdots x_n)$ is a sequence in $\mathcal{L}_x$, but $(x_1 \cdots x_n x_{n+1})$ is a sequence in $\overline{\mathcal{L}_x}$. This means that $(x_1 \cdots x_n x_{n+1})X^*$ are sequences in $\overline{\mathcal{L}_x}$. For input sequence $(x_1 \cdots x_n)$, the corresponding output sequence produced by M_2'' is $(y_2 \cdots y_n y_{n+1})$. Symbol y_n is the input to M_1 for the current transition. Suppose on the contrary that $(x_1 \cdots x_n x_{n+1})$ is not a sequence in $\overline{\mathcal{L}_x''}$. For input sequence $(y_1 \cdots y_n)$, x_{n+1} should be able to be produced by M_1 as the last output symbol. This implies that $(x_1 \cdots x_n x_{n+1})$ should be a sequence in $\mathcal{L}_x$. This is a contradiction. Thus, $(x_1 \cdots x_n x_{n+1})X^*$ are sequences in $\overline{\mathcal{L}_x''}$. Therefore, we conclude $\overline{\mathcal{L}_x} \subseteq \overline{\mathcal{L}_x''}$. ∎

Thus, $\mathcal{L}_{xy}$ is preserved and so is the input/output behavior of the whole circuit. Furthermore, input don't care sequences for M_1, $\overline{\mathcal{L}_y}$, need not to be recomputed after exploiting $\overline{\mathcal{L}_x}$ as input don't care sequences for M_2.

Theorem 4.2 *Input don't care sequences for a component in a general FSM network can be exploited using state minimization procedures for incompletely specified FSM's.*

Proof Directly from Lemma 4.1. ∎

Moreover, if $y = o$ in Figure 2, M_2' expresses the full flexibility for implementing M_2 as in the cascade circuit $M_1 \rightarrow M_2$ in Figure 1.

5 Implicit Computation

In this section, we focus on the implementation of the K-N procedure using implicit techniques. BDD's can represent and manipulate characteristic functions efficiently, and are thus suitable for implicit enumeration, e.g. implicit state reachability computations.

5.1 Implicit Representation of Finite Automata

The first step of the K-N procedure is to generate an NFA $\mathcal{A}'$ to accept output sequences of M_1. Then $\mathcal{A}'$ is converted into a DFA $\mathcal{A}$. $\mathcal{A}'$ and $\mathcal{A}$ have the following properties — every state except the *dead state* is a final state, the input string in each transition is of length one, and there are no ϵ-transitions. Note that we do not need to explicitly add the dead state in the transition relation, since it is implicit from all unspecified transitions. We do not need to specify the set of final states, since every state is a final state. As a consequence, we can represent the transition relations of $\mathcal{A}$ and $\mathcal{A}'$ in the same way as FSM's. The transition relation for $\mathcal{A}'$ can be implicitly computed as :

$$T_{\mathcal{A}'}(p_1, n_1, o_1) = \exists_{i_1} T_1(i_1, p_1, n_1, o_1) \, . \tag{4}$$

5.2 Implicitly Checking for Nondeterminism

When $\mathcal{A}'$ is a DFA, the subset construction is not needed. To detect this property is easy using BDD's. Let $T(p, n, i)$ be the transition relation of a finite automaton. We compute $T'(p, n, i)$ as follows:

$$T'(p, n, i) \; = \; \perp(\alpha, (T(p, n, i)) \tag{5}$$

where α is a reference next state vertex. For each pair (p, i) defined in $T(p, n, i)$, $\perp$ assigns a unique n. However, if $T'(p, n, i)$ equals $T(p, n, i)$, then nothing was changed, implying that T already had only one such candidate. Hence, $T(p, n, i)$ is deterministic; otherwise nondeterministic.

5.3 Implicitly Checking for Complete Specification

If M_1 and M_2 are completely specified FSM's, we may want to perform state minimization on them first. There exist efficient algorithms based on BDD's performing state minimization for completely specified machines [16]. Let $T(i, p, n, o)$ be the transition relation of an FSM M. An FSM M is completely specified if and only if λ and δ are complete functions. The check for λ is if each (i, p, n) defined in T has a unique o such that $(i, p, n, o) \in T$. We use the $\perp$ operator for this. Thus,

$$\perp(\alpha_o, T(i,p,n,o)) \;=\; T(i,p,n,o) \tag{6}$$

where α_o is a minterm in the O space. This says that T is output deterministic. Equation (5) could check if T is next state deterministic. T is input-completely-specified if and only if

$$\forall_p \exists_{o,n} T(i,p,n,o) \equiv 1\;. \tag{7}$$

This says that for all i and p, there exists a next state and output. Therefore, machine M is completely specified if and only if (5), (6) and (7) hold.

5.4 Implicitly Checking for Input Don't Care Sequences

If there is no dead state in $\mathcal{A}'$, there are no input don't care sequences from the driving machine, and vice versa. This is equivalent to checking if there are no unspecified transitions in $T_{\mathcal{A}'}(i, p, n)$. Therefore, $\mathcal{A}'$ does not generate any input don't care sequences if and only if

$$\forall_p \exists_n T_{\mathcal{A}'}(i,p,n) \equiv 1\;. \tag{8}$$

In Section 6, we present methods to derive a DFA $\mathcal{A}$ from $\mathcal{A}'$ and discuss how to compute subsets of input don't care sequences.

5.5 Implicit Construction of the Modified Machine

In the following theorem, we demonstrate that the transition relation of M_2', the modified machine, can be implicitly computed using BDD's.

Theorem 5.1 *Let the transition relations of $\mathcal{A}$ and M_2 be $T_{\mathcal{A}}(p_1, n_1, i_2)$, and $T_2(i_2, p_2, n_2, o_2)$, respectively. In $T_{\mathcal{A}}$, the dead state does not appear since we use incompletely specification of $\mathcal{A}$ to implicitly specify transitions to dead state. The transition relation of the incompletely specified modified machine $M_2{}'$, T_2', can be implicitly computed as follows :*

$$T_2'(i_2, p, n, o_2) \;=\; T_{\mathcal{A}}(p_1, n_1, i_2) \cdot T_2(i_2, p_2, n_2, o_2)$$

where $p = (p_1, p_2)$, $n = (n_1, n_2)$.

Proof The states of M_2' are represented by 2-tuples (s, t), where s and t are the states of $\mathcal{A}$ and M_2, respectively. Consider two states of M_2', (s_i, t_k) and (s_j, t_l). If there is a transition between these two states, then there exists an input symbol $a \in I_2$ such that it is on the transition edge from s_i to s_j in $\mathcal{A}$, and a transition edge from t_k to t_l taking a as its input in M_2. Since $\mathcal{A}$ is deterministic, a does not appear on the other transition edge emitted from s_i in $\mathcal{A}$. Furthermore, input symbols not specified on a state s of $\mathcal{A}$ implicitly transit from s to the dead state, and they do not contribute any transitions in M_2'. Therefore, they are implicitly converted to unspecified transitions. The above construction is exactly the same as step 3 of the K-N procedure. ∎

Afterwards, the implicit state reachability computation can be used to remove unreachable states in the transition relation of M_2', T_2', and then apply any state minimization program to minimize M_2', e.g. STAMINA [11].

6 Computing Subsets of Input Don't Care Sequences

Consider the cascade machine $M_1 \rightarrow M_2$ in Figure 1. Note that M_1 may be the *abstract driving machine* for M_2. Let output sequences produced by M_1 be $\mathcal{L}(M_1^o)$, a regular language over alphabet I_2. For computing and exploiting only a subset of input don't care sequences, any language $\mathcal{L}'$ such that

$$\mathcal{L}(M_1^o) \subseteq \mathcal{L}' \subseteq I_2^* \tag{9}$$

gives rise to a feasible subset $\overline{\mathcal{L}'}$ of input don't care sequences.

However, any superset of $\mathcal{L}(M_1^o)$ may not be a regular language, and hence not be accepted by any finite automaton. Moreover, in order to apply step 3 of the K-N procedure to construct M_2' which captures $\overline{\mathcal{L}'}$ as input don't care sequences, a DFA $\mathcal{A}$ accepting $\mathcal{L}'$ must be constructed first.

We present methods for finding a DFA accepting an $\mathcal{L}'$ that satisfies (9). In Section 6.1, we propose a bounded subset construction to perform approximation during step 2 of the K-N procedure. Two implementations of this bounded subset construction are disussed in Section 6.2 and 6.4, respectively. An simplification process which performs approximation on the NFA $\mathcal{A}'$ before the subset construction is described in Section 6.3.

6.1 Bounded Subset Construction

The classical algorithm to transform an NFA to a DFA is the subset construction [18]. In the worst case, the resultant state space is exponential in size. However, the exact DFA for accepting $\mathcal{L}(M_1^o)$ is not necessarily to be constructed if only a subset of don't care sequences is required, since only a DFA such that it accepts $\mathcal{L}'$ in (9) is needed. The NFA $\mathcal{A}'$ in step 1 of the K-N procedure does not have ϵ-transitions, and in each transition the input string is of length one. The subset construction[18, 5] is stated below.

Definition 10 **(Subset Construction)** *Given a nondeterministic finite automaton, $A = (K, \Sigma, \delta, q_0, F)$, the corresponding deterministic finite automaton, $A^d = (K^d, \Sigma, \delta^d, q_0^d, F^d)$, is defined as follows :*

$$K^d = 2^K, \quad q_0^d = \{q_0\}, \quad F^d = \{S \in K^d | S \cap F \neq \emptyset\}$$

and δ^d is the state transition function : $K^d \times \Sigma \rightarrow K^d$, defined by

$$\delta^d(S, a) = \bigcup_{q \in S} \delta(q, a).$$

Thus any final state in A^d is any subset of states of A containing at least one final state of A. In the application where A has only one nonaccepting state d which transits to itself for all input symbols, the only nonaccepting state of A^d is $\{d\}$, the dead state of A^d. During the subset construction, there is no difference between a subset S containing d and a subset $S - \{d\}$. The dead state $\{d\}$ of A^d can be implicitly specified by the empty set. Therefore, the dead state d can be excluded from the state space of A during computation.

A bound on the number of states created during the subset construction can be set to prevent space explosion.

Procedure : Bounded Subset Construction

1. Given a bound on the number of states, $N \geq 1$, assign the initial state $S_0 \leftarrow \{q_0\}$, a queue Q initialized as $Q \leftarrow \{S_0\}$, and a set R initialized as $R \leftarrow \{S_0\}$.
2. If (Q is empty) then return A_r^d, else $S_j \leftarrow$ first element of Q, and remove it from Q.
3. For each ($a \in \Sigma$)

   Let $S_k = \delta^d(S_j, a)$.

   Case 1 : If ($S_k \in R$), then add a transition (S_j, a, S_k) in A_r^d.

   Case 2 : If ($S_k \notin R$ and $|R| < N - 1$), then add S_k into R, append S_k in the end of Q, and add a transition (S_j, a, S_k) in A_r^d.

Case 3 : If ($S_k \notin R$ and $|R| = N - 1$), then add a transition (S_j, a, S_N) in A_r^d, where S_N transits to itself for all $a \in \Sigma$.

4. Go to step 2.

Theorem 6.1 *Let the NFA in step 1 of the K-N procedure be $\mathcal{A}'$, and its corresponding DFA be A^d by the subset construction. Then, the DFA A_r^d generated by the above procedure has the following property :*

$$\mathcal{L}(A^d) \subseteq \mathcal{L}(A_r^d)\,. \tag{10}$$

Therefore, A_r^d satisfies (9).

Proof If we exit with $(| R |< N - 1)$, the above procedure is exactly the subset construction. Thus, (10) is satisfied. Before we terminate the above procedure, we are just performing the subset construction. This corresponds to cases 1 and 2 in step 3. If we reach $N - 1$ before completion, we have two cases. If $S_k \in R$ (case 1), this procedure adds a transition (S_j, a, S_k) to A_r^d. In this case, this procedure still preforms the subset construction. If $S_k \notin R$ (case 3), this procedures adds a transition (S_j, a, S_N), where S_N transits to itself for all input symbols. S_N accepts all input sequences starting at S_N. Moreover, $\mathcal{A}'$ has the following property : every state except the dead state d is a final state. Therefore, every new state of A^d generated in the above procedure is a final state except the dead state $\{d\}$. With this construction, it guarantees that A_r^d accepts more input sequences than A^d. Thus, when this procedure terminates, the resultant automaton A_r^d accepts more input sequences than A^d. Therefore, it satisfies (10), and hence A_r^d satisfies (9). ■

We may need to perform state minimization on A_r^d, since it may contain many equivalent states. With the above procedure, the number of states generated in the subset construction is limited. The heuristic, *summarizing the DFA*, proposed in [21] is more sophisticated to approximate input don't care sequences, but one may need to start from a DFA $\mathcal{A}$ to perform approximation, i.e. after the subset construction, while we can avoid this complexity.

Based on the same rationale in the proof of Theorem 6.1, the NFA $\mathcal{A}'$ can be approximated before performing subset construction. We call this *filtering the NFA* and explain it in Section 6.3. In the following, we discuss the implementation of the bounded subset construction using implicit techniques.

6.2 Implicit Bounded Subset Construction

The bounded subset construction needs to check whether a new state of A_r^d is generated for an input symbol. When the number of input symbols of an NFA is large, explicit enumeration becomes inefficient. Here, we present an implicit enumeration algorithm. First, the subset construction using implicit enumeration is given.

Let the state space of $\mathcal{A}'$ be K (excluding the dead state d), and $| K |= k$. This can be represented by a k-valued variable s. However, in order to represent all possible 2^k subsets of K, each state s_i in K is associated with a Boolean variable x_i, a one-hot encoding representation. For example, suppose $k = 4$ and $K = \{s_0, s_1, s_2, s_3\}$. The corresponding one-hot encoding of state s_0 is $(x_0, x_1, x_2, x_3) = 1000$. Similarly, $\{s_0, s_1\}$ is 1100. Note that 0000 corresponds to the empty subset, i.e. the dead state $\{d\}$ of A^d. Let $X = (x_0, \ldots, x_{k-1})$, where $x_0 \cdots x_{k-1}$ are Boolean variables. Each minterm in the space of X corresponds to a subset of K. This encoding scheme can be represented using its characteristic function $\mathcal{E}_1(s, X)$,

$$\mathcal{E}_1(s, X) = \prod_{0 \le j \le k-1} (\overline{x_i} \cdot (s \neq s_j) + x_i \cdot (s = s_j))\,. \tag{11}$$

Let $T_{\mathcal{A}'}(i, p, n)$ be the transition relation and q_0 the initial state of $\mathcal{A}'$. Let $T_{A^d}(i, X_p, X_n)$ be the transition relation of A^d in terms of the one-hot encoding.

Theorem 6.2 **(Implicit Subset Construction)** *$T_{A^d}(i, X_p, X_n)$ is the limit of the following sequence, where $R_m(X_p)$ corresponds to the one-hot encoding of the subsets of K generated up to the m-th iteration, and $(T_{A^d})_m$ is all incoming transitions to these states, but given in the one-hot encoding.*

$$\begin{aligned}
R_0(X_p) &= \{\mathcal{E}_1(p, X_p)\}_{(p=q_0)} \\
(T_{A^d})_m &= \exists p, n\{\mathcal{E}_1(p, X_p) \cdot R_{m-1}(X_p) \cdot T_{\mathcal{A}'}(i, p, n) \cdot \mathcal{E}_1(n, X_n)\} \\
R_m(X_p) &= \theta_{X_n, X_p} \exists_{X_p, i}((T_{A^d})_m) + R_{m-1}(X_p) \\
(T_{A^d})_\infty &= (T_{A^d})_m, \quad R_\infty(X_p) = R_m(X_p) \text{ if } (T_{A^d})_{m+1} = (T_{A^d})_m\,.
\end{aligned}$$

Proof Consider $(T_{A^d})_m$. At the m-th iteration, $\mathcal{E}_1(p, X_p)$ maps $R_{m-1}(X_p)$, the generated subsets of K in terms of the one-hot encoding, into the state space K. Then $T_{\mathcal{A}'}(i, p, n)$ computes the next state image which is mapped by $\mathcal{E}_1(n, X_n)$ to express this next state image using the one-hot encoding. The smoothing of p and n requires that there exists p, n for X_p to be mapped into X_n. This gives $(T_{A^d})_m$. The new subset of K generated in the m-th iteration is added to $R_{m-1}(X_p)$ to give $R_m(X_p)$. When the fixed point is reached, the subset construction is completed. ∎

Each minterm of $R(X_p)$ corresponds to a subset of K. Let the number of minterms of $R(X_p)$ be k'(excluding the dead state $\{d\}$, i.e. minterm $00\ldots0$). Similar to equation (10), we can use a characteristic function $\mathcal{E}_2(s', X)$ to associate each minterm of $R(X_p)$ to a unique value of a k'-valued variable s'. Thus, the transition relation of A^d can be reexpressed as follows :

$$T'_{A^d}(i, p', n') = \exists_{X_p, X_n}(\mathcal{E}_2(p', X_p) \cdot T_{A^d}(i, X_p, X_n) \cdot \mathcal{E}_2(n', X_n)) . \tag{12}$$

To perform the bounded subset construction, we count the number of minterms of $R(X_p)$ at each iteration. Let $count(R_m(X_p))$ denote such an operation[3]. It equals the number of states of A^d generated by the subset construction up to the m-th iteration. Let the bound be N. We construct A^d_r as follows. When $count(R_m(X_p)) \leq N < count(R_{m+1}(X_p))$, we pick an arbitrary minterm h in $\overline{R_m(X_p)}$. Next assign h to transit to itself under all input combinations. For those transitions going to $\overline{R_m(X_p)}$ at the $(m+1)$-th iteration, reassign them to go to h. By theorem 6.1, the resultant A^d_r is guaranteed to satisfy (9). The transition relation of A^d_r, $T_{A^d_r}(i, X_p, X_n)$, is thus computed as follows :[4]

$$T_{A^d_r}(i, X_p, X_n) = \{\exists_{X_n}[(T_{A^d})_{m+1} \cdot \overline{R_m(X_n)}]\} \cdot h(X_n) + (T_{A^d})_{m+1} \cdot R_m(X_n) + h(X_p) \cdot h(X_n) .$$

Similarly, we can use (12) to reexpress $T_{A^d_r}(i, X_p, X_n)$ into $T'_{A^d_r}(i, p', n')$.

6.3 Implicit Filtering the NFA

If the state space K of the NFA $\mathcal{A}'$, $|K|$ is large, constructing BDD's for the implicit bounded subset construction may be inefficient, since the number of one-hot-encoding variables is large. To remedy this, an approximation can be performed by constructing an NFA $\mathcal{A}''$ with a bound on the number of states such that $\mathcal{L}(\mathcal{A}') \subseteq \mathcal{L}(\mathcal{A}'')$. This simplification process is called *filtering the NFA.*

Suppose the bound on the number of states in $\mathcal{A}''$ is N'. Let the $R_m(p)$ be the reachable states up to the m-th iteration when we perform implicit reachability computation on $T_{\mathcal{A}'}(i, p, n)$ using equations (1), (2), (3). Similar to the implicit bounded subset construction, when $count(R_m(p)) \leq N' < count(R_{m+1}(p))$, we pick an arbitrary state s_j in $\overline{R_m(p)}$. Next assign s_j to transit to itself for all i (self loop). For those transitions from $R_m(p)$ going to $\overline{R_m(p)}$ at the $(m+1)$-th iteration, reassign them to go to state s_j. Based on the same rationale in the proof of Theorem 6.1, this construction guarantees $\mathcal{L}(\mathcal{A}') \subseteq \mathcal{L}(\mathcal{A}'')$. The transition relation of $\mathcal{A}''$, $T_{\mathcal{A}''}(i, p, n)$ is thus computed as follows :

$$T_{\mathcal{A}''}(i, p, n) = [\exists_n(R_m(p) \cdot T_{\mathcal{A}'}(i, p, n) \cdot \overline{R_m(n)})] \cdot s_j(n) + s_j(p) \cdot s_j(n) + R_m(p) \cdot T_{\mathcal{A}'}(i, p, n) \cdot R_m(n) .$$

6.4 Partially Implicit Bounded Subset Construction

Using *the implicit bounded subset construction* and *filtering the NFA* together, we may handle a large state space and many input symbols. The efficiency of this method may deteriorate when the number of one-hot-encoding variables increases. In comparison, for a large state space and few input symbols, the bounded subset construction using explicit enumeration may be more efficient. In this subsection, we present an implementation using implicit techniques partially.

Let $T_{\mathcal{A}'}(i, p, n)$ be the transition relation of $\mathcal{A}'$. Consider step 3 of the bounded subset construction. The subset generated from $S_j(p)$ under an input symbol a, $S_k(p)$, can be computed as follows :

$$S_k(p) = \theta_{n,p}\exists_p\{(T_{\mathcal{A}'}(i, p, n) \cdot S_j(p))_{(i=a)}\} . \tag{13}$$

Thus, the subsets generated from $S_j(p)$ can be computed using equation (13) for all input symbols. This can be expedited by first computing the next state image of $T_{\mathcal{A}'}$ under $S_j(p)$.

$$\begin{aligned} G(i, p) &= \theta_{n,p}\exists_p(T_{\mathcal{A}'}(i, p, n) \cdot S_j(p)) . \\ L(i) &= \exists_p G(i, p) . \end{aligned}$$

[3]Counting the number of minterms in $R(X_p)$ can be done linearly in the number of BDD nodes.

[4]Here, $h(X_p)$ represents the characteristic function of the single state h chosen as the last state.

$i_1 i_2$	$p_1 p_2$
00	$S_3 = \{ 10 \}$
01	$S_1 = \{ 01, 10 \}$
10	$S_3 = \{ 10 \}$
11	$S_2 = \{ 00, 10 \}$

Table 1: An example of $G(i, p)$.

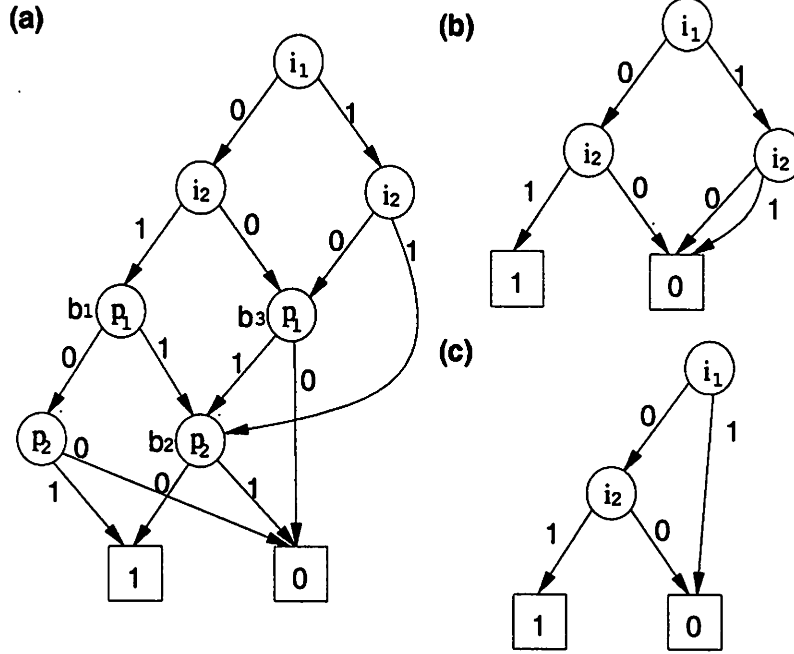

Figure 5: (a) The BDD of $G(i, p)$ with variable ordering $i_1 \prec i_2 \prec p_1 \prec p_2$. (b) Unreduced BDD of $f_1(i_1, i_2)$. (c) Reduced BDD of $f_1(i_1, i_2)$.

$\overline{L(i)}$ are those input symbols transiting to the dead state $\{d\}$ from S_j. The subsets generated from $S_j(p)$ can be computed by only cofactoring $G(i, p)$ with respect to each minterm in $L(i)$. However, this is still not efficient when there are many input minterms in $L(i)$.

Here, we present a method to implicitly compute all subsets generated from $S_j(p)$. First, we use an example to illustrate. $G(i, p)$ is shown in Table 1 where $i = (i_1, i_2)$ and $p = (p_1, p_2)$. The subsets of $G(i, p)$ are $S_1 = \{01, 10\}, S_2 = \{00, 10\}$, and $S_3 = \{10\}$. Assign the BDD variable ordering as $i \prec p$. The BDD of $G(i, p)$ is shown in Figure 5(a). Nodes b_1, b_2, and b_3 in the BDD of $G(i, p)$ correspond to S_1, S_2, S_3, respectively. In fact, they are those nodes with variable p_1 or p_2 such that there are incident edges from nodes with variable i_1 or i_2. The corresponding input combinations for producing $S_1(p)$, $f_1(i)$, can be easily computed as follows : assign those edges which point to node b_1 to "1" node, and those directed edges which point to b_2 or b_3 to "0" node. This results in an unreduced BDD of $f_1(i)$ as shown in Figure 5(b), and its corresponding reduced BDD is in Figure 5(c).

We define the set of *border nodes*, $\mathcal{B}$, in the BDD of $G(i, p)$ with the variable ordering $i \prec p$ as follows: $\mathcal{B} = \{ b \mid b$ is a constant node or a node with a variable in p, such that there are incident edges from nodes with variables in i. $\}$. For example, $\mathcal{B} = \{b_1, b_2, b_3\}$ in Figure 5(a).

Theorem 6.3 *Let $G(i, p)$ and $\mathcal{B}$ be defined as above. The variable ordering is $i \prec p$. Each node in $\mathcal{B}$ corresponds to a subset generated from $S_j(p)$ and vice versa, i. e. the number of nodes in $\mathcal{B}$ equals to the number of subsets generated from $S_j(p)$.*

Proof Let $\mathcal{B} = \{b_1, \ldots, b_m\}$, and $f_1(i), \ldots, f_m(i)$ be the corresponding input combinations. Then, $G(i, p) = f_1(i) \cdot b_1(p) +$

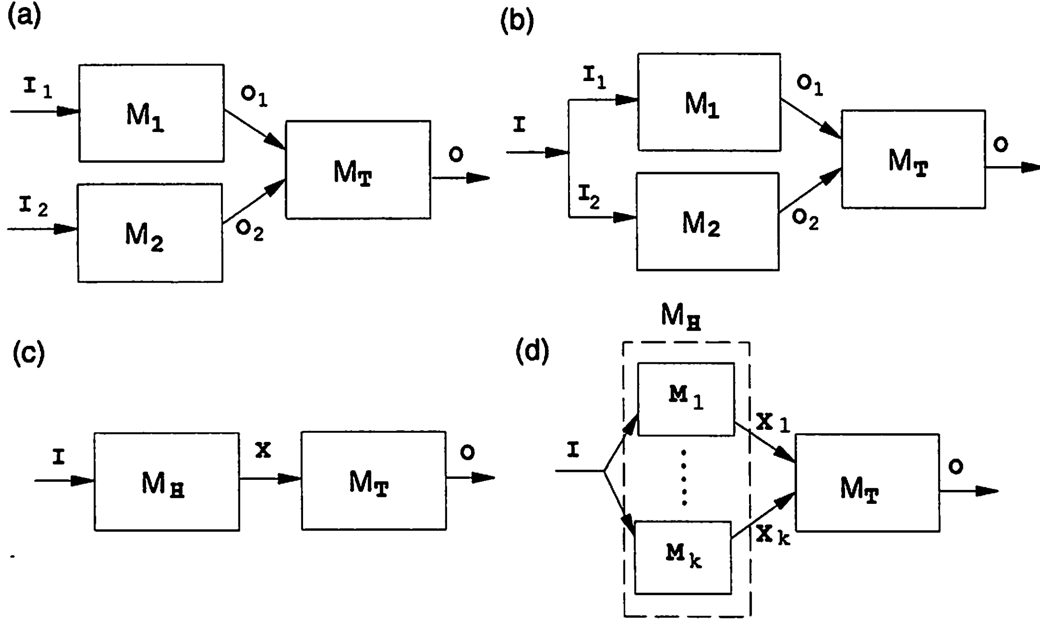

Figure 6: (a) An FSM network with $I_1 \neq I_2$. (b) An FSM network with $I = I_1 = I_2$. (c) A cascade circuit $M_H \rightarrow M_T$. (d) Decomposed M_H cascaded with M_T.

$\cdots + f_m(i) \cdot b_m(p)$. Then $b_k(p)$ corresponds to a subset generated from $S_j(p)$, since $f_k(i) \neq \emptyset$ by definition and for each minterm a of $f_k(i)$, $G(i,p)_{(i=a)} = b_k(p)$. Suppose on the contrary there is a subset $b' \notin B$, and its corresponding set of input combinations, $f'(i)$, is not empty. However, $f_1(i), \ldots f_m(i)$ are distinct and $f_1(i) + \cdots + f_m(i)$ is a tautology. Let a be a minterm of f'. Then by $G(i,p)_{i=a}$ we get its corresponding b in B. Since $b \neq b'$, this is a contradiction. ∎

This method is efficient when there are many input combinations associated with the same subset. Checking whether a new subset is generated is easy, because every subset is expressed in terms of BDD's. Since generating and checking new subsets can be done efficiently using BDD's, this method is more efficient for a large state space and medium-size input combinations.

7 Approximate Computation in Large FSM networks

An *abstract driving machine* may be the system itself. Constructing the transition relation of the whole system and then performing the approximation methods in Section 6 may be still expensive. We discuss how to control the complexity of approximating input don't care sequences for a component in an large FSM network.

Consider the FSM networks shown in Figure 6(a), (b). Let the transition relations of M_1, M_2 be $T_1(i_1, p_1, n_1, o_1)$ and $T_2(i_2, p_2, n_2, o_2)$, respectively. Let $\mathcal{A}'$, $\mathcal{A}_1{}'$, $\mathcal{A}_2{}'$ be the NFA, derived from step 1 of the K-N procedure, accepting output sequences of $M_1 \times M_2$, M_1, M_2, respectively. $\mathcal{A}$, $\mathcal{A}_1$, $\mathcal{A}_2$ are their corresponding DFA derived from the bounded subset construction for $\mathcal{A}'$, $\mathcal{A}_1{}'$, $\mathcal{A}_2{}'$ respectively. $T_{\mathcal{A}'}$, $T_{\mathcal{A}_1{}'}$, $T_{\mathcal{A}_2{}'}$, derived from (4), denote the transition relations of $\mathcal{A}'$, $\mathcal{A}_1{}'$, $\mathcal{A}_2{}'$, respectively[5].

Lemma 7.1 *([25]) Let $f : B^n \times B^m \rightarrow B$ and $g : B^m \rightarrow B$ be two Boolean functions. Then*

$$\exists_x (f(x,y) \cdot g(y)) = (\exists_x f(x,y)) \cdot g(y) .$$

Lemma 7.2 *([17]) Let $f : B^n \times B^m \rightarrow B$ and $g : B^n \times B^m \rightarrow B$ be two Boolean functions. Then*

$$\exists_x (f(x,y) \cdot g(x,y)) \subseteq (\exists_x f(x,y)) \cdot (\exists_x g(x,y)) .$$

[5]Note that the alphabet of $\mathcal{A}'$, $\mathcal{A}_1{}'$, $\mathcal{A}_2{}'$ is $O_1 \times O_2$.

Lemma 7.3 *([5]) Let* $A_1 = (K_1, \Sigma, \delta_1, q_{0_1}, F_1)$, $A_2 = (K_2, \Sigma, \delta_2, q_{0_2}, F_2)$, *be two DFA.* $A_1 \times A_2$ *is a quintuple* $(K_1 \times K_2, \Sigma, \delta, (q_{0_1}, q_{0_2}), F_1 \times F_2)$, *where for all* $p_1 \in K_1$, $p_2 \in K_2$, *and* $a \in \Sigma$, $\delta((p_1, p_2), a) = (\delta_1(p_1, a), \delta_2(p_2, a))$. *Then*

$$\mathcal{L}(A_1 \times A_2) = \mathcal{L}(A_1) \cap \mathcal{L}(A_2).$$

This lemma also holds for the NFA discussed here (e.g. $\mathcal{A}'$, $\mathcal{A}_1{}'$, $\mathcal{A}_2{}'$), since they have no ϵ-transitions, and the input string in each transition is of length one.

Consider the FSM networks in Figure 6(a), (b). In the following, we demonstrate that output sequences of $M_1 \times M_2$ can be approximated with the output sequences of M_1 and M_2 separately.

Theorem 7.4 *Let* M_1, *and* M_2 *be FSM's as shown in Figure 6(a), where* $I_1 \neq I_2$. *Then*

$$\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A}_1{}' \times \mathcal{A}_2{}').$$

Proof The transition relation of $\mathcal{A}'$, $T_{\mathcal{A}'}$, can be computed using (4):

$$\begin{aligned} T_{\mathcal{A}'} &= \exists_{i_1, i_2}(T_1(i_1, p_1, n_1, o_1) \cdot T_2(i_2, p_2, n_2, o_2)) \\ &= (\exists_{i_1} T_1(i_1, p_1, n_1, o_1)) \cdot (\exists_{i_2} T_2(i_2, p_2, n_2, o_2)) \quad \text{(by Lemma 7.1)} \\ &= T_{\mathcal{A}_1{}'}(p_1, n_1, o_1) \cdot T_{\mathcal{A}_2{}'}(p_2, n_2, o_2). \end{aligned}$$

Therefore, $T_{\mathcal{A}'} = T_{\mathcal{A}_1{}'} \cdot T_{\mathcal{A}_2{}'} = T_{\mathcal{A}_1{}' \times \mathcal{A}_2{}'}$. This implies $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A}_1{}' \times \mathcal{A}_2{}')$. ∎

Corollary 7.5 *Let* M_1, *and* M_2 *be FSM's as shown in Figure 6(a), where* $I_1 \neq I_2$. *Then* $\mathcal{L}(\mathcal{A}') \subseteq \mathcal{L}(\mathcal{A}_1 \times \mathcal{A}_2)$.

Proof

$$\begin{aligned} \mathcal{L}(\mathcal{A}') &= \mathcal{L}(\mathcal{A}_1{}' \times \mathcal{A}_2{}') \\ &= \mathcal{L}(\mathcal{A}_1{}') \cap \mathcal{L}(\mathcal{A}_2{}') \quad \text{(by Lemma 7.3)} \\ &\subseteq \mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2) \quad (\mathcal{L}(\mathcal{A}_1{}') \subseteq \mathcal{L}(\mathcal{A}_1), \text{and } \mathcal{L}(\mathcal{A}_2{}') \subseteq \mathcal{L}(\mathcal{A}_2) \text{ by Theorem 6.1}) \\ &= \mathcal{L}(\mathcal{A}_1 \times \mathcal{A}_2). \end{aligned}$$

∎

By Theorem 7.4, output sequences of $M_1 \times M_2$ in Figure 6(a) are the same as the intersection of output sequences of M_1 and M_2. This provides a way to compute output sequences of $M_1 \times M_2$. First derive $\mathcal{A}_1{}'$, $\mathcal{A}_2{}'$, and then apply the bounded subset construction to compute $\mathcal{A}_1$ and $\mathcal{A}_2$. By Corollary 7.5, $\mathcal{L}(\mathcal{A}_1 \times \mathcal{A}_2)$ is guaranteed to satisfy (9). Using this approach, we can avoid performing the bounded subset construction on $\mathcal{A}'$ directly. Thus, this reduces the complexity for performing approximations.

Theorem 7.6 *Let* M_1, *and* M_2 *be FSM's as shown in Figure 6(b), where* $I_1 = I_2 = I$. *Then*

$$\mathcal{L}(\mathcal{A}') \subseteq \mathcal{L}(\mathcal{A}_1{}' \times \mathcal{A}_2{}').$$

Proof The transition relation of $\mathcal{A}'$, $T_{\mathcal{A}'}$, can be computed using (4):

$$\begin{aligned} T_{\mathcal{A}'} &= \exists_i(T_1(i, p_1, n_1, o_1) \cdot T_2(i, p_2, n_2, o_2)) \\ &\subseteq (\exists_i T_1(i, p_1, n_1, o_1)) \cdot (\exists_i T_2(i, p_2, n_2, o_2)) \quad \text{(by Lemma 7.2)} \\ &= T_{\mathcal{A}_1{}'}(p_1, n_1, o_1) \cdot T_{\mathcal{A}_2{}'}(p_2, n_2, o_2). \end{aligned}$$

Therefore, $T_{\mathcal{A}'} \subseteq T_{\mathcal{A}_1{}'} \cdot T_{\mathcal{A}_2{}'} = T_{\mathcal{A}_1{}' \times \mathcal{A}_2{}'}$. The unspecified transitions are implicitly assigned to the dead state, the only nonaccepting state. Thus, this implies $\mathcal{L}(\mathcal{A}') \subseteq \mathcal{L}(\mathcal{A}_1{}' \times \mathcal{A}_2{}')$. ∎

Corollary 7.7 *Let* M_1, *and* M_2 *be FSM's as shown in Figure 6(b), where* $I_1 = I_2 = I$. *Then* $\mathcal{L}(\mathcal{A}') \subseteq \mathcal{L}(\mathcal{A}_1 \times \mathcal{A}_2)$.

circuit	M_1			M_2			X	Y	total		CPU	
	I	O	S	I	O	S			B	A	DC	SM
KN2	1	1	5	1	1	5	1	1	10	5	0.2	0.1
S2	10	10	5	10	3	4	3	3	9	8	0.5	0.1
L1 (bbara=ex2)	4	2	7	2	2	29	2	2	36	10	0.8	3.1
L2 (ex6=s1)	5	8	8	8	6	21	8	4	29	13	1.5	0.1
L3 (keyb=dk16)	7	2	19	2	3	76	2	3	95	20	1.1	20.9
L4 (s1=ex4)	8	6	20	6	9	14	6	3	34	18	1.7	10.9
L5 (pma=s1488)	8	8	24	8	19	61	8	4	85	38	6.0	24.1
L6 (s386=keyb)	7	7	13	7	2	23	7	2	36	20	1.2	653.0

Table 2: Experimental results for two-way-communcation circuits.

M_1, M_2: interacting FSM's
I, O, S: number of PI's, PO's, states, respectively
X (Y): number of signals from M_1 to M_2 (from M_2 to M_1)
B (A): sum of number of states in M_1 and M_2 before (after) exploiting don't cares
DC: time for computing don't cares (in seconds on a DEC 5000/240)
SM: time for STAMINA (in seconds on a DEC 5000/240)

By Theorem 7.6, output sequences of $M_1 \times M_2$ in Figure 6(b) are contained in the intersection of output sequences of M_1 and M_2. This provides a way to approximate input don't care sequences for M_T. Similar to the FSM network in Figure 6(a), we can use $\mathcal{L}(\mathcal{A}_1 \times \mathcal{A}_2)$ to approximate output sequences of $M_1 \times M_2$. By Corollary 7.7, it is guaranteed to satisfy (9). Also, this reduces the complexity for performing approximations.

Consider the cascade, $M_H \rightarrow M_T$ in Figure 6(c). M_H may be the *abstract driving machine* of M_T in a large FSM network. M_H may have a large state space and many interacting signals, denoted as x, to M_T. As explained in Section 6.3, the implicit bounded subset construction can handle the case when there are many input combinations, but it is not suitable for a large state space. However, together with *filtering the NFA* method which limits the number of states of $\mathcal{A}'$ before the subset construction, approximate computation of output sequences from M_H can be preformed.

Another approach which enhances the ability to manage the complexity to perform approximate computation in large FSM networks is the following. Decompose M_H into $M_1, \ldots, M_k$ as shown in Figure 6(d). This can be done as follows. Partition the interacting signals x into $x_1, \ldots, x_k$. The transition relation of M_i is $\exists_{x_j \neq x_i} T_{M_H}$. By such a construction, this reduces the problem to one in Figure 6(b). We can then individually compute the approximate output sequences of $M_1, \ldots, M_k$. Suppose these are $\mathcal{A}_1, \ldots, \mathcal{A}_k$ respectively. The partially implicit bounded subset construction in Section 6.4 can be employed. It is more efficient for a large state space and medium-size input combinations, since generating and checking new subsets can be done efficiently using BDD's. Then construct $\mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_k$. The state minimization on $\mathcal{A}$ can be done using the method in [16]. Note that if the state space of the minimized DFA is still large, a similar method to the bounded subset construction can be applied to limit the number of states to make further approximation.

8 Experimental Results

In this section, we present preliminary results on small networks. Due to the lack of FSM network benchmark examples, most of the examples here are obtained by connecting FSM's from MCNC benchmarks. These FSM's are completely specified and state-minimal. We use STAMINA [11] to exploit input don't care sequences.

Table 2 shows some experimental results for two-way communication circuits with two FSM's. The circuit topology of these examples is shown in Figure 2. The bound on the number of states for subset construction is 32. Example KN2 is obtained by connecting the output of the driven machine of the cascade circuit in [13] to the input of its driving machine. Example S2 is obtained by decomposing s386 into a two-way-communication network. The other examples are obtained by connecting two FSM's from MCNC benchmarks. To prevent direct information feedback, we attach latches to the interacting signals from M_2 to M_1, i.e. y.

Example KN2 is the only two-way-communication circuit considered in [20]. The total number of states reported by [20] for the two FSM's of example KN2 is 7, and CPU time is 96.0 seconds (on a DEC 5000/200) after performing two iterations

circuit	I	X	O	S_1	S_2		CPU	
					B	A	DC	SM
C1 (ex7-dk16)	2	2	3	10	27	15	0.65	0.2
C2 (keyb-dk16)	7	2	3	19	27	19	0.49	0.1
C3 (s510-keyb)	19	7	2	47	19	15	1.92	21.2
C4 (bbsse-keyb)	7	7	2	16	19	18	0.67	3.9
C5 (bbsse-planet)	7	7	19	16	48	42	0.97	548.6
C6 (sand-ex1)	11	9	19	32	20	8	1.94	117.9
C7 (s1488-s510)	8	19	7	48	47	4	4.80	0.6
C8 (ex1-s510)	9	19	7	20	47	7	1.89	0.3

Table 3: Experimental results of one-way-communcation circuits.

M_1 (M_2): driving machine (driven machine)
I, O, X: number of PI's, PO's, interacting signals of $M_1 \rightarrow M_2$, respectively
S_1 (S_2): number of states of M_1 (M_2)
B (A): number of states of M_2 before(after) exploiting don't cares
DC: time for computing don't cares (in seconds on a DEC 5000/240)
SM: time for STAMINA (in seconds on a DEC 5000/240)

around the loop and state minimization. The number of states of the DFA for capturing input don't care sequences grows too much after two iterations. Thus, much CPU time is spent in state minimization [19]. In contrast, our approach employs the notion of an *abstract driving machine* and takes much less CPU time to achieve a better result as shown in Table 2.

Table 3 shows some experimental results for cascade circuits consisting of two FSM's. The purpose of these experiments is to examine the case when there are many interacting signals from an *abstract driving machine* to its corresponding driven machine. The bound on the number of states for subset construction is 64.

We have implemented most of our algorithms and are studying various trade-offs offered by different approximation methods. Our preliminary results indicate that the notion of *abstract driving machines* is very promising for computing input don't care sequences in general FSM networks. We plan to do more experiments on large FSM networks using the approximation methods in Section 6 and 7.

9 Conclusion

We presented a novel approach to compute the exact input don't care sequences for a component in an FSM network with arbitrary topology by converting this problem into a cascade circuit consisting of this component and its corresponding *abstract driving machine*. In case the exact computation and exploitation are too expensive, we provided approximation methods to compute subsets of input don't care sequences. We have also discussed how to implement the algorithms using implicit enumeration techniques. For large FSM networks, we proposed methods to manage the complexity to perform approximate computations. Preliminary results look promising but larger networks must be experimented on.

10 Acknowledgements

The authors are thankful to Szu-Tsung Cheng and Thomas Shiple for helpful discussions on the use of the BDD package.

References

[1] K. L. Brace, R. E. Bryant, and R. L. Rudell. Efficient Implementation of a BDD Package. In *27th ACM/IEEE Design Automation Conference*, pages 40–45, June 1990.

[2] R. K. Brayton and F. Somenzi. Boolean Relations and the Incomplete Specification of Logic Networks. In *VLSI'89*, August 1989.

[3] R. E. Bryant. Graph Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

[4] J. R. Burch, E. M. Clarke, K. L. McMillan, and D. L. Dill. Sequential Circuit Verification Using Symbolic Model Checking. In *27th ACM/IEEE Design Automation Conference*, pages 46–51, Orlando, June 1990.

[5] J. Carroll and D. Long. *Theory of Finite Automata : with an Introduction to Formal Languages*. Prentice Hall, Englewood Cliffs, NJ, 1989.

[6] E. Cerny and M. A. Marin. An Approach to Unified Methodology of Combinational Switching Circuits. In *IEEE Transactions on Computers*, pages 745–756, August 1977.

[7] O. Coudert, C. Berthet, and J. C. Madre. Verification of Sequential Machines Based on Symbolic Execution. In *Proceedings of the Workshop on Automatic Verification Methods for Finite State Systems*, Grenoble, France, 1989.

[8] O. Coudert and J.C. Madre. A Unified Framework for the Formal Verification of Sequential Circuits. In *IEEE International Conference on Computer-Aided Design*, pages 126–129, November 1990.

[9] M. Damiani and G. De Micheli. Recurrence Equations and the Optimization of Synchronous Circuits. In *28th ACM/IEEE Design Automation Conference*, pages 556–561, June 1992.

[10] S. Devadas. Optimizing Interacting Finite State Machines Using Sequential Don't Cares. In *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, pages 1473–1484, December 1991.

[11] G. Hachtel, J. K. Rho, F. Somenzi, and R. Jacoby. Exact and Heuristic Algorithms for the Minimization of Incompletely Specified State Machines. In *The European Conference on Design Automation*, 1991.

[12] J. E. Hopcroft. An nlog(n) Algorithm for Minimizing the States in a Finite Automaton. In *The Theory of Machines and Computation, ed. Z. Kohavi*, 1971.

[13] J. Kim and M. M. Newborn. The Simplification of Sequential Machines With Input Restrictions. In *IEEE Transactions on Computers*, pages 1440–1443, December 1972.

[14] B. Lin and A. R Netwon. Implicit Manipulation of Equivalence Classes Using Binary Decision Digrams. In *International Workshop on Logic Synthesis*, 1991.

[15] B. Lin and F. Somenzi. Minimization of Symbolic Relations. In *IEEE International Conference on Computer-Aided Design*, pages 88–91, November 1990.

[16] B. Lin, H. Touati, and A. R. Newton. Don't Care Minimization of Multi-Level Sequential Logic Networks. In *IEEE International Conference on Computer-Aided Design*, pages 414–417, November 1990.

[17] P. McGeer. *On the Interaction of Functional and Timing Behavior of Combinational Logic Circuits*. PhD thesis, U.C. Berkeley, November 1989.

[18] M. Rabin and D. Scott. Finite Automata and Their Decision Problems. In *IBM Journal of Research and Development*, pages 114–125, 1959.

[19] J. K. Rho. Private Communication. April 1993.

[20] J. K. Rho, G. Hachtel, and F. Somenzi. Don't Care Sequences and the Optimization of Interacting Finite State Machines. In *International Workship on Logic Synthesis*, May 1991.

[21] J. K. Rho, G. Hachtel, and F. Somenzi. Don't Care Sequences and the Optimization of Interacting Finite State Machines. In *IEEE International Conference on Computer-Aided Design*, pages 418–421, November 1991.

[22] H. Savoj and R. K. Brayton. Observability Relations and Observability Don't Cares. In *IEEE International Conference on Computer-Aided Design*, pages 518–521, November 1991.

[23] A. Srinivasan, T. Kam, S. Malik, and R. K. Brayton. Algorithms for Discrete Function Manipulation. In *IEEE International Conference on Computer-Aided Design*, pages 92–95, November 1990.

[24] H. Touati, R. K. Brayton, and R. Kurshan. Testing Language Containment for ω-Automata using BDD's. In *Proceedings of ACM/SIGDA International Workshop on Formal Method s in VLSI Designs*, Miami, January 1991.

[25] H. Touati, H. Savoj, B. Lin, R. K. Brayton, and A. Sangiovanni-Vincentelli. Implicit State Enumeration of Finite State Machines using BDD's. In *IEEE International Conference on Computer-Aided Design*, pages 130–133, November 1990.

[26] S. H. Unger. *Asynchronous Sequential Switching Circuits*. John Wiley, 1969.