THE MAXIMUM SET OF PERMISSIBLE BEHAVIORS FOR FSM NETWORKS

-Synthesis of Interacting Finite State Machines-

by

Yosinori Watanabe and Robert K. Brayton

Memorandum No. UCB/ERL M93/61

3 August 1993

THE MAXIMUM SET OF PERMISSIBLE BEHAVIORS FOR FSM NETWORKS

-Synthesis of Interacting Finite State Machines-

by

Yosinori Watanabe and Robert K. Brayton

Memorandum No. UCB/ERL M93/61

3 August 1993

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

THE MAXIMUM SET OF PERMISSIBLE BEHAVIORS FOR FSM NETWORKS

-Synthesis of Interacting Finite State Machines-

by

Yosinori Watanabe and Robert K. Brayton

Memorandum No. UCB/ERL M93/61

3 August 1993

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

The Maximum Set of Permissible Behaviors for FSM Networks

- Synthesis of Interacting Finite State Machines -

Yosinori Watanabe and Robert K. Brayton*
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley, CA 94720, USA

Abstract

For systems of interacting finite state machines (FSM's), manual designs sometimes use information derived from the other components to optimize one of them. An associated problem is to find the set of *permissible* sequential functionalities that can be implemented at a component while preserving the behavior of the total system. Most conventional approaches attempt to find such a set using the notion of don't care sequences, but in general, the complete set of permissible finite state machines are difficult to compute. As a result, only small subsets are derived and used in designing interacting components. However, there is no knowledge of how much optimality is lost using these subsets.

This paper proposes a method for computing and representing the complete set of permissible finite state machines. We show that the complete set can be computed and represented by a single non-deterministic finite state machine, called the *E-machine*. The computation is different from any based on don't care sequences. The transition relation of the E-machine is obtained by a fixed point computation. The procedure has been implemented and initial experimental results are given.

*This research is supported in part by the National Science Foundation and the Defense Advanced Research Projects Agency under contract number NSF/DARPA-MIP-871-9546. We also thank AT&T, DEC, IBM, Intel, and Motorola for their support.

1 Introduction

In combinational logic synthesis, a common procedure focuses on a node of a logic network and derives a set of *don't cares*[1]. Associated with each node is the current logic implementation (completely specified Boolean function) which gives an initial representation. The don't care set provides the condition of the rest of the network under which one can alter the functionality of the node while preserving the functionality of the entire network. Such a functionality of the node is said to be *permissible*[13]. The resulting node function is then treated as an incompletely specified Boolean function, to locally optimize the node. Intensive research has been made on how to derive the don't care conditions as well as how to optimize the resulting node functions[1, 2, 8, 10, 13]. More recently, these ideas have been generalized to networks where each node may have more than one output (and hence multi-valued variables are allowed), and the specification on the functionality of the entire network is given as a Boolean relation between the primary inputs and the primary outputs of the network[3, 16]. One wants to derive and represent all possible permissible functions of a particular node of the network. It is known that in the combinational domain, the set of permissible functions can be represented by a single Boolean relation for each node[3, 16].

For sequential logic synthesis, the analogous concept is an FSM network, where associated with each node is a completely specified deterministic finite state machine (FSM) and the network's specification is given by a non-deterministic finite state machine. Each finite state machine is represented by its transition relation (a Boolean relation), relating inputs, outputs, present states, and next states. This situation is shown in Figure 1. The only distinction between this and the combinational logic situation is what each node represents - in one case a finite state machine, in the other a pure Boolean function.

As an analogy to the combinational case, one wants to derive and represent all possible *permissible* finite state machines for a particular node of an FSM network, where a completely specified finite state machine is said to be permissible if it can be implemented at the node while the resulting sequential functionality of the entire network still meets the specification. This set of permissible machines is then used in some optimal search procedure for a best choice.

The problem of finding permissible finite state machines at a given node of an FSM network can be viewed as an interaction between two finite state machines, as shown in Figure 2, where M_1 is the initial machine associated with the node being optimized, M_2 represents the functionality of the rest of the network, and M gives the specification. This problem has been studied extensively[6, 9, 15]. Most work is based on *don't care sequences*; sequences of the inputs of M_1 which never occur and sequences of outputs of M_1,

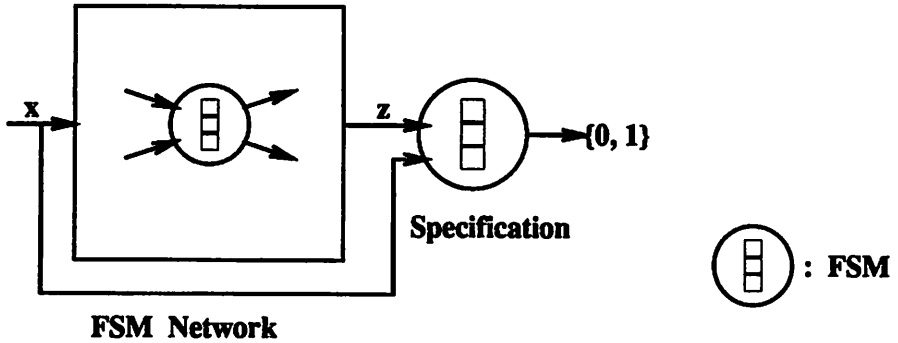

Figure 1: FSM Network and its Specification

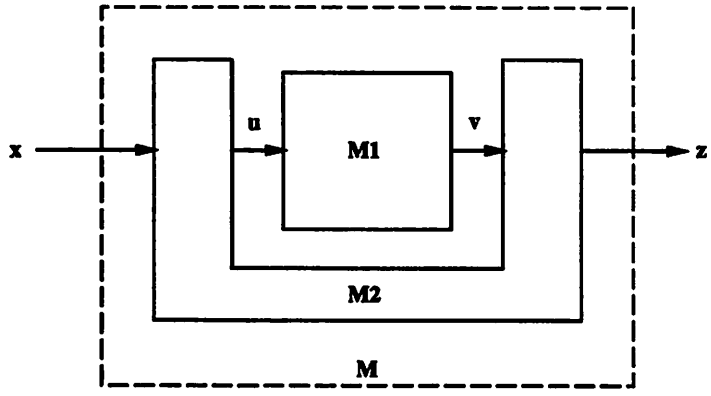

Figure 2: Interaction between Two Machines

given sequences of the network inputs x, such that the resulting network outputs z meet the specification M. However, approaches based on don't care sequences have several limitations. First, since the inputs of M_1 (outputs of M_2) may depend on the outputs of M_1, the sequences that appear at the inputs of M_1 can be controlled by changing the functionality of M_1, which then defines a different set of input don't care sequences. Thus the previous work either makes an assumption on the topology of the network of M_1 and M_2, such as cascaded machines where M_2 is independent of M_1, or restrict themselves to compute only a subset of don't care sequences[6, 9, 15]. Furthermore, due to complexity, often the sequences are only partially considered, up to a certain, typically small, length. As a result, even though one finds the *best* machine among the set of permissible machines computed, there is no guarantee that the machine is best among all permissible machines; the search for an optimum is severely limited.

In this report, we ask if it is possible to compute and represent easily the complete set of permissible finite state machines at M_1. The answer is yes and it can be represented by a single non-deterministic finite state machine, which we call the *E-machine*. The result is obtained by a simple fixed point computation which provides the transition relation of the E-machine. The procedure has been implemented and the E-machines for a small set of artificial examples of moderate size have been derived.

There are a couple of problems or applications that can be viewed as a variation of the focus of this report. One such problem is to find the set of permissible behaviors of the outside component M_2 when the internal component M_1 and the global behavior M are given. This problem can be solved in exactly the same way as the original problem, since the interaction shown in Figure 2 can be redrawn as shown in Figure 3-(a), which yields to the same picture of Figure 2 by modifying M_1 so that the global input X and the global output Z pass through M_1. This is illustrated in Figure 3-(b). Such a problem arises in a rectification problem[7, 17], where the designer wants to change the functionality of the design by attaching a small logic around the original circuitry.

Another related problem is a supervisory control problem for discrete event processes[14]. The problem is that for a given generator of discrete events and a specification on the generated events, we observe the events provided by the generator and control them by feeding control events to the generator so that the resulting output events meet the specification. If the generator and the supervisor are synchronizing, this

problem can be deemed as a variation of our problem, as shown in Figure 4, where M_1 is the generator, M_2 is the supervisor, and V and Z are identical while M_1 may be a non-deterministic machine.

Finally, the core problem that we are dealing with is the division of finite state machines; given M and M_1, find the quotient Q of the two (See Figure 5). This is the central problem of factorization and decomposition of finite state machines.

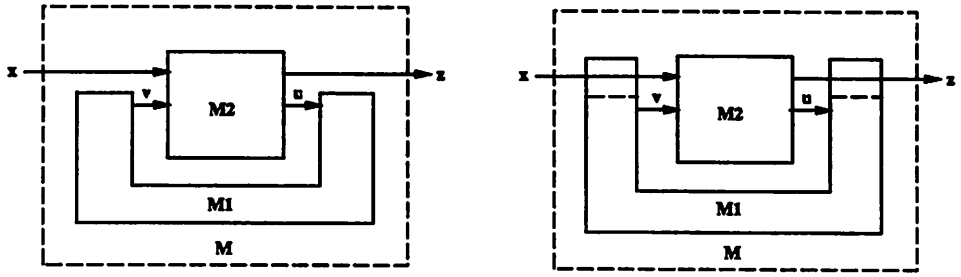

Figure 3: Application for Rectification Problem

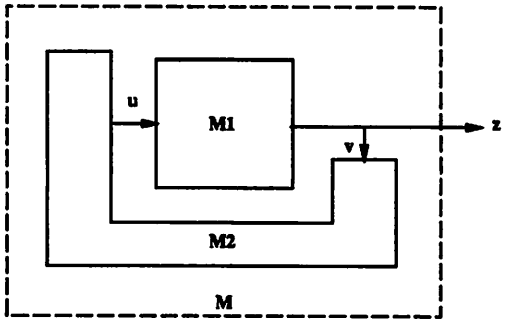

Figure 4: Application for Supervisory Control Problem

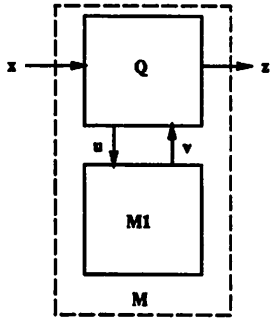

Figure 5: Application for FSM Division

2 Terminology

In this section, we define basic terminology used throughout the report.

Definition: Finite State Machine

> A **finite state machine** is a 5-tuple (I, O, S, T, r), where I is the set of (binary) input variables, O is the set of (binary) output variables, S is the set of states, $T : S \times B^{|I|} \times B^{|O|} \times S \rightarrow B$ is a characteristic function with $B = \{0, 1\}$, and r is an element of S. The machine stays in an exactly one state, say s_p, of S at any given time. The machine takes as input a minterm $\mathbf{i} \in B^{|I|}$, by which a transition is caused. A transition at a state s_p consists of a pair of state and an output minterm, $(s_n, \mathbf{o}) \in S \times B^{|O|}$, which indicates that the machine moves from the state s_p to the state s_n and outputs $\mathbf{o}$. It is assumed that a transition takes no time. The function T defines the valid transitions of the machine, i.e. $T(s_p, \mathbf{i}, \mathbf{o}, s_n) = 1$ if and only if the transition $(s_n, \mathbf{o})$ can be caused at the state s_p by the input $\mathbf{i}$. There exists at least one (could be more than one) valid transition for each state s_p and an input $\mathbf{i}$. The state r, defined in the 5-tuple, designates the state at which the machine stays initially, and is called the reset state.

The function $T : S \times B^{|I|} \times B^{|O|} \times S \rightarrow B$ is called the transition relation of the finite state machine[1]. For a given state $s_0 \in S$ and a sequence $\sigma_i = (\mathbf{i}_0, \ldots, \mathbf{i}_{t-1})$ of the input minterms, where $\mathbf{i}_j \in B^{|I|}$ for each $j = 0, \ldots, t-1$ and $t \geq 0$,[2] there exists a sequence of output minterms $\sigma_o = (\mathbf{o}_0, \ldots, \mathbf{o}_{t-1})$ and a sequence of states $\sigma_s = (s_0, s_1, \ldots, s_t)$ with the property that $T(s_j, \mathbf{i}_j, \mathbf{o}_j, s_{j+1}) = 1$ for all $j = 0, \ldots, t-1$. Such a sequence of output minterms (a sequence of states, respectively) is called an output sequence (state transition) defined at s_0 by σ_i. In particular, if s_0 is set to the reset state r, we say that the sequence σ_i leads the machine to the state s_t. In this case, we also say that the pair (σ_i, σ_o) is realized by the machine. The integer t of the sequence σ is called the *length* of σ and is denoted by $|\sigma|$.

Definition: Deterministic Finite State Machine

> A finite state machine (I, O, S, T, r) is said to be **deterministic** if there exists a pair of functions, $\lambda : S \times B^{|I|} \rightarrow \{0, 1, *\}^{|O|}$ and $\delta : S \times B^{|I|} \rightarrow S \cup \{*\}$, such that for all $(s_p, \mathbf{i}, \mathbf{o}, s_n) \in S \times B^{|I|} \times B^{|O|} \times S$, $T(s_p, \mathbf{i}, \mathbf{o}, s_n) = 1$ if and only if
>
> $$\begin{array}{ll} (1) & \mathbf{o} \subseteq \lambda(s_p, \mathbf{i}) \text{ and} \\ (2) & \delta(s_p, \mathbf{i}) = * \text{ or } \delta(s_p, \mathbf{i}) = s_n, \end{array}$$
>
> where for $\tilde{\mathbf{o}} = \lambda(s_p, \mathbf{i})$, $\mathbf{o} \subseteq \tilde{\mathbf{o}}$ designates that for each output variable j, $\tilde{o}_j \neq *$ implies that $o_j = \tilde{o}_j$.
>
> A deterministic finite state machine is said to be *completely specified* if for all $(s_p, \mathbf{i}) \in S \times B^{|I|}$, $\lambda(s_p, \mathbf{i}) \in B^{|O|}$ and $\delta(s_p, \mathbf{i}) \in S$. Otherwise, the machine is said to be *incompletely specified.*

By definition, the valid transitions of a deterministic finite state machine can be represented using the functions λ and δ given above. Therefore, we may represent the machine by a 6-tuple $(I, O, S, \lambda, \delta, r)$. Note

[1] We make no distinction between a set and its characteristic function.

[2] Such a sequence σ_i is called an input sequence. If $t = 0$, the sequence is null.

that at each state, the output sequence and the state transition defined at the state by a given input sequence are unique for a completely specified deterministic machine.

A deterministic finite state machine $(I, O, S, \lambda, \delta, r)$ is called a *Moore machine*[12] if for each state $s_p \in S$, there exists a unique $\mathbf{o} \in \{0, 1, *\}^{|O|}$ such that $\lambda(s_p, \mathbf{i}) = \mathbf{o}$ for all $\mathbf{i} \in B^{|I|}$. Otherwise, it is called a *Mealy machine*[11]. Note that the function λ of a Moore machine depends only on the states S and not on the inputs $B^{|I|}$.

A finite state machine that is not deterministic is said to be *non-deterministic*.

Definition: Reachable States

> Given a completely specified deterministic finite state machine $(I, O, S, \lambda, \delta, r)$, a state $s \in S$ is said to be **reachable** if there exists a sequence of the input minterms which leads the machine to s.

A state that is not reachable is said to be unreachable.

Definition: Equivalent States

> Given a completely specified deterministic finite state machine $(I, O, S, \lambda, \delta, r)$, a pair of states $(s, \tilde{s}) \in S \times S$ is said to be **equivalent** if for all sequences of the input minterms, say σ, the output sequence defined by σ at s is identical with that defined at $\tilde{s}$.
>
> A set of states of S is said to be equivalent if every pair of states in the set is equivalent in M.

The set of states S can be uniquely divided into a set of disjoint classes, where each class consists of maximal number of equivalent states. Each such class is called an *equivalent class*.

Definition: Equivalent Machines

> Two completely specified deterministic machines $M = (I, O, S, \lambda, \delta, r)$ and $\tilde{M} = (I, O, \tilde{S}, \tilde{\lambda}, \tilde{\delta}, \tilde{r})$ are **equivalent** if for all sequences of the input minterms, say σ, the output sequence define at r by σ in M is identical with that defined at $\tilde{r}$ by σ in $\tilde{M}$.

In this report, we often discuss the *behaviors* of finite state machines. Intuitively, a behavior between the input variables I and the output variables O is the set of pairs of input and output sequences realized by a completely specified deterministic finite state machine with the input I and the output O. In this sense, we say that the machine *represents* the behavior. Although this intuitive definition will suffice to understand the report, we provide a formal definition of a behavior using notion of finite automata.

Definition: Finite Automaton

> A (deterministic) **finite automaton** is a 5-tuple (X, S, δ, F, r), where X is the set of (binary) input variables, S is the set of states, $\delta : S \times B^{|X|} \rightarrow S$ is the transition function, $F \subseteq S$ is the set of final states, and $r \in S$ is the reset state.

A finite automaton (X, S, δ, F, r) has a one-to-one correspondence with a completely specified deterministic finite state machine with a single output o, $(X, o, S, \lambda, \delta, r)$, which has the identical transition function δ, where $\lambda(s, \mathbf{x}) = 1$ if and only if $\delta(s, \mathbf{x}) \in F$ in the original automaton. Hence, terminology, defined for finite state machines, will be used for finite automata as well. A sequence on X which leads the automaton to a state in F is said to be *accepted* by the automaton. We now define a behavior as follows.

Definition: Behavior

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

1. For an arbitrary sequence σ_i on I, there exists a unique pair in $\mathcal{B}$ whose input sequence is equal to σ_i.
2. For an arbitrary pair $p = (\sigma_i, \sigma_o) \in \mathcal{B}$, where $\sigma_i = (\mathbf{i}_0, \ldots, \mathbf{i}_t)$ and $\sigma_o = (\mathbf{o}_0, \ldots, \mathbf{o}_t)$ with $t > 0$, let $\bar{\sigma}_i = (\mathbf{i}_0, \ldots, \mathbf{i}_{t-1})$ and $\bar{\sigma}_o = (\mathbf{o}_0, \ldots, \mathbf{o}_{t-1})$. Then $(\bar{\sigma}_i, \bar{\sigma}_o) \in \mathcal{B}$.
3. For an arbitrary pair $p = (\sigma_i, \sigma_o) \in \mathcal{B}$, where $\sigma_i = (\mathbf{i}_0, \ldots, \mathbf{i}_t)$ and $\sigma_o = (\mathbf{o}_0, \ldots, \mathbf{o}_t)$ with $t \geq 0$, let $\sigma(p)$ be a sequence on $I \cup O$ defined as $\sigma(p) = (\mathbf{i}_0\mathbf{o}_0, \ldots, \mathbf{i}_t\mathbf{o}_t)$. Then there exists a finite automaton with the input $I \cup O$ which accepts all and only the sequences of the set given by $\{\sigma(p) \mid p \in \mathcal{B}\}$.

For each pair (σ_i, σ_o) of a behavior, we say that (σ_i, σ_o) is *realized* by the behavior.

For a non-deterministic finite state machine, there might exist more than one valid transition for some state and an input. In this sense, we can regard that a non-deterministic machine represents a set of behaviors represented by a set of completely specified deterministic machines. We call each of such behaviors a *contained behavior.*

Definition: Contained Behavior

Given a finite state machine $T = (I, O, S, T, r)$, a behavior between I and O is said to be **contained** in T if every pair of input and output sequences of the behavior is realized by T.

By definition, if T is a completely specified deterministic machine, there is a unique behavior contained in it.

3 The Maximum Set of Permissible Behaviors

3.1 The Problem and Assumptions

Consider the case of two interacting finite state machines shown in Figure 2. M_1 takes input u and outputs v, while M_2 takes input x and v and outputs u and z. M is a finite state machine with input x and output z, which represents the behavior of the entire system composed of M_1 and M_2.

Specifically, let $M = (X, Z, S, T, r)$ and $M_2 = (X \cup V, U \cup Z, S_2, \lambda_2, \delta_2, r_2)$ be given. We assume that M may be a non-deterministic machine while M_2 is deterministic. By allowing non-determinism on M, we can specify a set of behaviors, rather than a single behavior, for the entire system. In the sequel, we often discuss which outputs *can* be obtained from M at a particular state and a particular input. For this purpose, let us introduce two functions $\Lambda : 2^{|S|} \times B^{|X|} \rightarrow 2^{|B^{|Z|}|}$ and $\Delta : 2^{|S|} \times B^{|X|} \rightarrow 2^{|S|}$ defined as follows:

$$\begin{aligned} \Lambda(s^*, \mathbf{x}) &= \{\mathbf{z} \in B^{|Z|} \mid \exists(\bar{s}, s) \in s^* \times S : T(\bar{s}, \mathbf{x}, \mathbf{z}, s) = 1\} \\ \Delta(s^*, \mathbf{x}) &= \{s \in S \mid \exists(\bar{s}, \mathbf{z}) \in s^* \times B^{|Z|} : T(\bar{s}, \mathbf{x}, \mathbf{z}, s) = 1\}, \end{aligned}$$

where we denote by s^* a subset of states of M and by $2^{|S|}$ the power set of S.

Note that the output of the function λ_2 is a pair of minterms $(\mathbf{u}, \mathbf{z}) \in B^{|U|} \times B^{|Z|}$. In the sequel, we may represent λ_2 using two functions $\lambda_2^{(u)} : S_2 \times B^{|X \cup V|} \rightarrow B^{|U|}$ and $\lambda_2^{(z)} : S_2 \times B^{|X \cup V|} \rightarrow B^{|Z|}$ such that $\lambda_2(s_2, \mathbf{xv}) = (\lambda_2^{(u)}(s_2, \mathbf{xv}), \lambda_2^{(z)}(s_2, \mathbf{xv}))$.

We are interested in finding a set of behaviors represented by finite state machines permissible at M_1. Here, we assume that a circuit implementation of M_1 and M_2 does not contain a combinational loop, i.e. a cycle without a register. Note that in general, if both M_1 and M_2 are Mealy machines, since the outputs depend on the inputs, there might exist a variable of V which depends on a variable of U in M_1, while the variable of U depends on the variable of V in M_2. We exclude this situation and consider only the machines that can be implemented at M_1 without introducing combinational loops. Specifically, we define implementable machines as follows.

Definition: Implementable Finite State Machine

> Given $M_2 = (X \cup V, U \cup Z, S_2, \lambda_2, \delta_2, r_2)$, a completely specified deterministic finite state machine $(U, V, S_1, \lambda_1, \delta_1, r_1)$ is said to be **implementable** at M_1 if there exists a pair of circuit implementations of M_1 and M_2 respectively such that no combinational loop is created by connecting them together at U and V.

We will discuss implementability in more detail in Section 3.4 and provide a necessary and sufficient condition under which M_1 is implementable. We say a behavior between U and V is implementable at M_1 if there exists an implementable machine at M_1. Note that for an implementable machine M_1, and for an arbitrary sequence of $B^{|X|}$, say $\sigma = (\mathbf{x}_0, \ldots, \mathbf{x}_t)$, if we denote by $(s_1, s_2) \in S_1 \times S_2$ the pair of states of M_1 and M_2 led by $(\mathbf{x}_0, \ldots, \mathbf{x}_{t-1})$, then $\mathbf{x}_t$ defines the pair $(\mathbf{u}, \mathbf{v}) \in B^{|U|} \times B^{|V|}$ such that $\mathbf{u} = \lambda_2^{(u)}(s_2, \mathbf{x}_t\mathbf{v})$ and $\mathbf{v} = \lambda_1(s_1, \mathbf{u})$.

For an implementable machine $M_1 = (U, V, S_1, \lambda_1, \delta_1, r_1)$, we define the product machine of M_1 and M_2, denoted by $M_1 \times M_2$, as a completely specified deterministic finite state machine $(X, Z, S_p, \lambda_p, \delta_p, r_p)$ such that $S_p = S_1 \times S_2$, $r_p = (r_1, r_2)$, and for a state $(s_1, s_2) \in S_p$ and a minterm $\mathbf{x} \in B^{|X|}$, $\lambda_p((s_1, s_2), \mathbf{x}) = \mathbf{z}$ if and only if there exist $\mathbf{u} \in B^{|U|}$ and $\mathbf{v} \in B^{|V|}$ such that $\lambda_1(s_1, \mathbf{u}) = \mathbf{v}$ and $\lambda_2(s_2, \mathbf{xv}) = (\mathbf{u}, \mathbf{z})$. Similarly, $\delta_p((s_1, s_2), \mathbf{x}) = (\bar{s}_1, \bar{s}_2)$ if and only if there exist $\mathbf{u} \in B^{|U|}$ and $\mathbf{v} \in B^{|V|}$ such that $\lambda_1(s_1, \mathbf{u}) = \mathbf{v}$, $\lambda_2^{(u)}(s_2, \mathbf{xv}) = \mathbf{u}$, and $(\delta_1(s_1, \mathbf{u}), \delta_2(s_2, \mathbf{xv})) = (\bar{s}_1, \bar{s}_2)$.

We now define a permissible machine as follows.

Definition: Permissible Finite State Machine

> Given $M = (X, Z, S, T, r)$ and $M_2 = (X \cup V, U \cup Z, S_2, \lambda_2, \delta_2, r_2)$, a completely specified deterministic finite state machine $M_1 = (U, V, S_1, \lambda_1, \delta_1, r_1)$ is said to be **permissible** if M_1 is implementable and the behavior of $M_1 \times M_2$ is contained in M.

The behavior represented by a permissible machine is called a *permissible behavior*. Our objective is to find the complete set of permissible behaviors at M_1. Note that we are not interested in finding the complete set of permissible machines at M_1; we need enough machines which represent the complete set of permissible behaviors. We show that the complete set of permissible behaviors can be computed and represented by a single finite state machine, which we call the E-machine.

3.2 Prime Machines

Our objective is to show how the complete set of permissible behaviors can be computed and represented by a single finite state machine called the E-machine. The key idea is to represent the behavior of a machine implementable at M_1 using a special deterministic machine called a prime machine. In this section, we present the definition of prime machines.

For given $M = (X, Z, S, T, r)$ and $M_2 = (X \cup V, U \cup Z, S_2, \lambda_2, \delta_2, r_2)$, consider an implementable machine $M_1 = (U, V, S_1, \lambda_1, \delta_1, r_1)$.

Definition: $\Sigma(s_1, t)$

For a state $s_1 \in S_1$ and an integer $t \geq 0$, let $\Sigma(s_1, t)$ be a subset of $S_2 \times 2^{|S|}$ such that $(s_2, s^*) \in S_2 \times 2^{|S|}$ is in $\Sigma(s_1, t)$ if and only if there exists a sequence of $B^{|X|}$ with the length t which leads $M_1 \times M_2$ to (s_1, s_2) and M to all and only the states of s^*. As a special case, we define $\Sigma(s_1, -1) = \phi$ for all $s_1 \in S_1$.

$\Sigma(s_1, t)$ consists of all possible states of M_2 and M that can be associated with s_1 after t transitions starting from the reset states.

Definition: $N(\tilde{s}_1, \Sigma, \mathbf{u}, s_1)$

Given $s_1 \in S_1$, $\tilde{s}_1 \in S_1$, $\mathbf{u} \in B^{|U|}$, and $\Sigma \subseteq S_2 \times 2^{|S|}$, let $N(\tilde{s}_1, \Sigma, \mathbf{u}, s_1)$ be a subset of $S_2 \times 2^{|S|}$ given by

$$N(\tilde{s}_1, \Sigma, \mathbf{u}, s_1) = \{(s_2, s^*) \in S_2 \times 2^{|S|} \mid \exists \mathbf{x} \in B^{|X|}, (\tilde{s}_2, \tilde{s^*}) \in \Sigma : \begin{array}{ll} \mathbf{u} = \lambda_2^{(u)}(\tilde{s}_2, \mathbf{xv}), & s_1 = \delta_1(\tilde{s}_1, \mathbf{u}), \\ s_2 = \delta_2(\tilde{s}_2, \mathbf{xv}), & s^* = \Delta(\tilde{s^*}, \mathbf{x}) \end{array} \},$$

where $\mathbf{v} = \lambda_1(\tilde{s}_1, \mathbf{u})$.

Intuitively, $N(\tilde{s}_1, \Sigma, \mathbf{u}, s_1)$ defines all possible states of M_2 and M such that $M_1 \times M_2$ (respectively M) can move from $(\tilde{s}_1, \tilde{s}_2)$ (respectively from $\tilde{s^*}$) in a single transition for some $(\tilde{s}_2, \tilde{s^*}) \in \Sigma$, where the transition causes M_1 to move to s_1 with the input minterm set to $\mathbf{u}$.

Definition: Prime Machine

An implementable machine $M_1 = (U, V, S_1, \lambda_1, \delta_1, r_1)$ is said to be **prime** if for each state $s_1 \in S_1$, there exists a subset $\Sigma(s_1) \subseteq S_2 \times 2^{|S|}$ with the following property:

(1) $\forall t \geq 0 : \Sigma(s_1, t) \neq \phi \Rightarrow \Sigma(s_1, t) = \Sigma(s_1)$
(2) $\forall \mathbf{u} \in B^{|U|}, \tilde{s}_1 \in S_1 : s_1 = \delta_1(\tilde{s}_1, \mathbf{u}) \Rightarrow N(\tilde{s}_1, \Sigma(\tilde{s}_1), \mathbf{u}, s_1) = \Sigma(s_1)$
(3) $\nexists t \geq 0 : \Sigma(s_1, t) \neq \phi \Rightarrow \Sigma(s_1) = \{\phi\}$

In other words, each state of a prime machine, whenever it is reached, is identified with exactly one subset of $S_2 \times 2^{|S|}$. Note that if M is a completely specified deterministic machine, then $\Sigma(s_1)$ is a set of pairs of states of M_2 and M, i.e. a subset of $S_2 \times S$.

Theorem 3.1 *For an implementable machine M_1, there exists an equivalent prime machine.*

We present the proof of this theorem in the Appendix. The theorem claims that the set of prime machines provides the complete set of implementable behaviors. Hence only prime machines need to be considered in order to be able to represent all permissible behaviors. Let us present another property that holds for permissible prime machines. We use this property in constructing the E-machine.

Theorem 3.2 *Suppose a prime machine M_1 is permissible. Consider a state $s_1 \in S_1$ such that $\Sigma(s_1, t) \neq \phi$ for some $t \geq 0$. Then the following property holds.*

$$\forall (s_2, s^*) \in \Sigma(s_1), \forall \mathbf{x} \in B^{|X|} : \lambda_2^{(z)}(s_2, \mathbf{xv}) \in \Lambda(s^*, \mathbf{x}),$$

where $\mathbf{v} \in B^{|V|}$ is the output minterm of M_1 uniquely defined for the input $\mathbf{x}$ at the state (s_1, s_2) of $M_1 \times M_2$.

Proof: Suppose for contrary that $\lambda_2^{(z)}(s_2, \mathbf{xv}) \notin \Lambda(s^*, \mathbf{x})$. Since $\Sigma(s_1, t) \neq \phi$ for some $t \geq 0$ and since $(s_2, s^*) \in \Sigma(s_1)$, there exists a sequence σ on X which leads $M_1 \times M_2$ to (s_1, s_2) and M to the states of s^*. Then at the state (s_1, s_2), the output of $M_1 \times M_2$ with the input $\mathbf{x}$ is different from any output that can be obtained by M at a state of s^* with the same input $\mathbf{x}$. It follows that the behavior of $M_1 \times M_2$ is not contained in M, which is conflict with the fact that M_1 is permissible. This completes the proof. ■

Example 1 *Consider M_2 and M shown in Figure 6, where each of X, V, U, and Z consists of a single variable, while a node and an edge represents a state and a transition, respectively. The label associated with an edge shows the minterms of the inputs and the outputs for the transition corresponding to the edge. The label associated with a node is the name of the corresponding state. The reset states of M_2 and M are the state 1 and the state A, respectively.*

Three permissible machines for these M_2 and M are shown in Figure 7-(a), (b), and (c), respectively. For each machine, the one shown on the right-hand side is an equivalent prime machine, where the label associated with each state s_1 is $\Sigma(s_1)$.

3.3 The E-machine and its Properties

3.3.1 The E-machine

Consider the transition relation of a non-deterministic machine given by the following computation. Let $\mathcal{S}^{(0)} = \{(r_2, \{r\})\}$ and compute $T^{(t+1)}$ and $\mathcal{S}^{(t+1)}$ for a given $\mathcal{S}^{(t)} \subseteq S_2 \times 2^{|S|}$. Let Σ_p and Σ_n be subsets of $S_2 \times 2^{|S|}$, respectively, and $\mathbf{u}$ and $\mathbf{v}$ be minterms of $B^{|U|}$ and $B^{|V|}$. $T^{(t+1)}(\Sigma_p, \mathbf{u}, \mathbf{v}, \Sigma_n) = 1$ if and only if the following three conditions are satisfied:

(1) $\Sigma_p \in \mathcal{S}^{(t)}$

(2) $\forall (\mathbf{x}, \tilde{s}_2, \tilde{s^*}) \in B^{|X|} \times S_2 \times 2^{|S|}$: $(\tilde{s}_2, \tilde{s^*}) \in \Sigma_p$ and $\mathbf{u} = \lambda_2^{(u)}(\tilde{s}_2, \mathbf{xv})$
$\Rightarrow$ (a) $\lambda_2^{(z)}(\tilde{s}_2, \mathbf{xv}) \in \Lambda(\tilde{s^*}, \mathbf{x})$
(b) $(\delta_2(\tilde{s}_2, \mathbf{xv}), \Delta(\tilde{s^*}, \mathbf{x})) \in \Sigma_n$

(3) $\forall (s_2, s^*) \in S_2 \times 2^{|S|}$: $(s_2, s^*) \in \Sigma_n$:
$\Rightarrow$ $\exists (\mathbf{x}, \tilde{s}_2, \tilde{s^*}) \in B^{|X|} \times S_2 \times 2^{|S|}$
(a) $(\tilde{s}_2, \tilde{s^*}) \in \Sigma_p$
(b) $\mathbf{u} = \lambda_2^{(u)}(\tilde{s}_2, \mathbf{xv})$
(c) $s_2 = \delta_2(\tilde{s}_2, \mathbf{xv})$
(d) $s^* = \Delta(\tilde{s^*}, \mathbf{x})$.

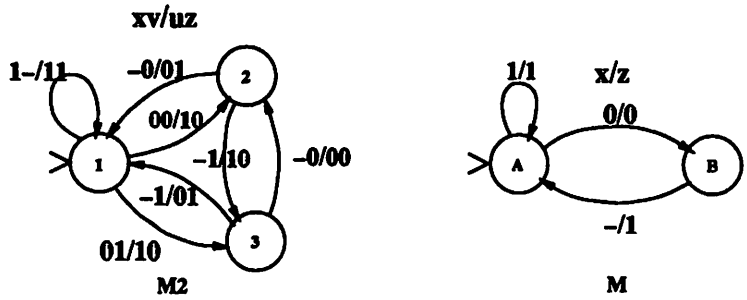

Figure 6: Example of M_2 and M

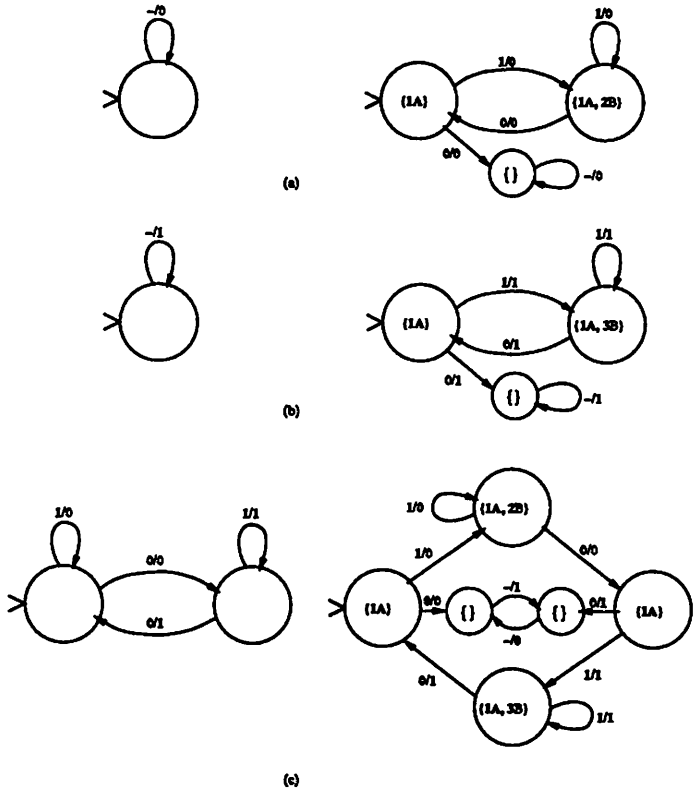

Figure 7: Permissible Machines M_1 (u/v)

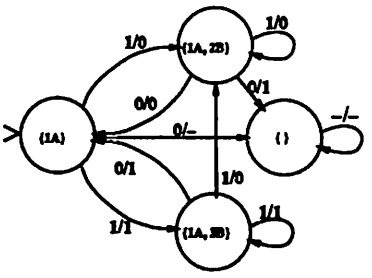

Figure 8: The E-machine T

In each computation, $S^{(t)}$ is a set of subsets of $S_2 \times 2^{|S|}$. Note that the empty set $\{\phi\}$ may be in $S^{(t)}$. Given $T^{(t+1)}$, we compute $S^{(t+1)}$ as follows. $S^{(t+1)}(\Sigma_p) = 1$ if and only if $S^{(t)}(\Sigma_p) = 1$ or there exist $\hat{\Sigma}_p \in S^{(t)}$, $\mathbf{u} \in B^{|U|}$, and $\mathbf{v} \in B^{|V|}$ such that $T^{(t+1)}(\hat{\Sigma}_p, \mathbf{u}, \mathbf{v}, \Sigma_p) = 1$. Intuitively, what we are computing is a transition relation that on each step is being extended to a new set of states, where each state corresponds to a subset of $S_2 \times 2^{|S|}$. These states are added to the transition relation. This is continued until nothing new is seen.

Let K be the smallest positive integer such that $S^{(K)}(\Sigma_p) = S^{(K-1)}(\Sigma_p)$. Such K always exists since the number of the elements of the set $S^{(t)}$ is not decreasing during the computation and the number of subsets of $S_2 \times 2^{|S|}$ is finite. Let $S = S^{(K)} \cup \{\phi\}$.

Let $T : S \times B^{|U|} \times B^{|V|} \times S \rightarrow B$ be a relation such that $T(\Sigma_p, \mathbf{u}, \mathbf{v}, \Sigma_n) = 1$ if and only if

(1) $\Sigma_p = \Sigma_n = \{\phi\}$ or
(2) $T^{(K)}(\Sigma_p, \mathbf{u}, \mathbf{v}, \Sigma_n) = 1$.

We finally define the E-machine as a 5-tuple $T = (U, V, S, T, \Sigma_r)$, where $\Sigma_r = \{(r_2, \{r\})\}$. We recall that each state of the E-machine represents a subset of $S_2 \times 2^{|S|}$. Note that if M is a completely specified deterministic machine, a state of the E-machine is a set of pairs of states of M_2 and M.

Example 2 *For M_2 and M in Example 1, the transition relation of the E-machine is shown in Figure 8.*

Note that in case the machine M is deterministic, each state of the E-machine corresponds to a subset of $S_2 \times S$, rather than $S_2 \times 2^{|S|}$.

3.3.2 Properties of the E-machine

The objective in this section is to show that the E-machine captures the complete set of permissible behaviors. More specifically, a behavior implementable at M_1 is permissible if and only if the behavior is contained in the E-machine. We first claim that a behavior implementable at M_1 contained in the E-machine is permissible.

Theorem 3.3 *A behavior implementable at M_1 contained in the E-machine is permissible.*

Proof: Consider an arbitrary sequence $\sigma = (\mathbf{x}_0, \ldots, \mathbf{x}_k)$ of $B^{|X|}$. Let $\sigma_t = (\mathbf{x}_0, \ldots, \mathbf{x}_t)$ be the subsequence of σ with the length $t+1$, where $0 \leq t \leq k$ and we define σ_{-1} as the null sequence. Since the behavior is implementable, σ_t uniquely defines the pair of sequences of $B^{|U|}$ and $B^{|V|}$, say $(\sigma_u^{(t)}, \sigma_v^{(t)})$ where $\sigma_u^{(t)} = (\mathbf{u}_0, \ldots, \mathbf{u}_t)$ and $\sigma_v^{(t)} = (\mathbf{v}_0, \ldots, \mathbf{v}_t)$, such that $(\sigma_u^{(t)}, \sigma_v^{(t)})$ is realized by the behavior and for each i, $0 \leq i \leq t$, $\mathbf{u}_i = \lambda_2^{(u)}(s_2^{(i)}, \mathbf{x}_i\mathbf{v}_i)$ and $s_2^{(i+1)} = \delta_2(s_2^{(i)}, \mathbf{x}_i\mathbf{v}_i)$, where $s_2^{(0)} = r_2$. Let us define $(\sigma_u^{(-1)}, \sigma_v^{(-1)})$ as the pair of null sequences. Note that M_2 is led to $s_2^{(t+1)}$ by applying σ_t. Also, for $t \geq 0$, let $s^* \subseteq S$ and $\tilde{s}^* \subseteq S$ be the set of states to which M is led by σ_t and σ_{t-1}, respectively, where in case $t = 0$, we define $\tilde{s}^* = \{r\}$.

We show by induction on $t \geq 0$ that by applying the subsequence σ_t,

1. $\lambda_2^{(z)}(s_2^{(t)}, \mathbf{x}_t\mathbf{v}_t) \in \Lambda(\tilde{s}^*, \mathbf{x}_t)$, and
2. There exists a unique state $\Sigma^{(t+1)} \in \mathcal{S}$ to which the E-machine is led by $(\sigma_u^{(t)}, \sigma_v^{(t)})$.
3. For the $\Sigma^{(t+1)}$ defined above, $(s_2^{(t+1)}, s^*) \in \Sigma^{(t+1)}$.

First, if we denote $\Sigma^{(0)} = \Sigma_r$, then $\Sigma^{(0)}$ is the unique state of the E-machine to which the E-machine can be led by $(\sigma_u^{(-1)}, \sigma_v^{(-1)})$, where we see $(r_2, \{r\}) \in \Sigma^{(0)}$. Also, by the induction hypothesis, there exists a unique $\Sigma^{(t)}$ to which the E-machine can be led by $(\sigma_u^{(t-1)}, \sigma_v^{(t-1)})$. Furthermore, $(s_2^{(t)}, \tilde{s}^*) \in \Sigma^{(t)}$. It follows that $\Sigma^{(t)} \neq \{\phi\}$.

Since $(s_2^{(t)}, \tilde{s}^*) \in \Sigma^{(t)}$ and $\mathbf{u}_t = \lambda_2^{(u)}(s_2^{(t)}, \mathbf{x}_t\mathbf{v}_t)$, $T^{(K)}(\Sigma^{(t)}, \mathbf{u}, \mathbf{v}, \{\phi\}) = 0$. However, since the behavior is contained in T and since $\Sigma^{(t)}$ is the unique state to which the E-machine can be led by $(\sigma_u^{(t-1)}, \sigma_v^{(t-1)})$, there must exist a state $\Sigma^{(t+1)} \in \mathcal{S}$ such that $T(\Sigma^{(t)}, \mathbf{u}, \mathbf{v}, \Sigma^{(t+1)}) = 1$. It follows that $T^{(K)}(\Sigma^{(t)}, \mathbf{u}, \mathbf{v}, \Sigma^{(t+1)}) = 1$. Therefore, by construction, $\lambda_2^{(z)}(s_2^{(t)}, \mathbf{x}_t\mathbf{v}_t) \in \Lambda(\tilde{s}^*, \mathbf{x}_t)$. Since $s_2^{(t+1)} = \delta_2(s_2^{(t)}, \mathbf{x}_t\mathbf{v}_t)$ and $s^* = \Delta(\tilde{s}^*, \mathbf{x}_t)$, we also see by construction $(s_2^{(t+1)}, s^*) \in \Sigma^{(t+1)}$. It remains to show that such $\Sigma^{(t+1)}$ is unique. Since we know the uniqueness of $\Sigma^{(t)}$, the proof is done if we show that for any $\Sigma \in \mathcal{S}$ such that $T(\Sigma^{(t)}, \mathbf{u}, \mathbf{v}, \Sigma) = 1$, $\Sigma = \Sigma^{(t+1)}$. Consider an arbitrary such Σ. By the argument above, $\Sigma \neq \{\phi\}$, and thus $T^{(K)}(\Sigma^{(t)}, \mathbf{u}, \mathbf{v}, \Sigma) = 1$. Then by the condition (3) of the construction of $T^{(K)}$ shown in Section 3.3.1, for an arbitrary pair $(s_2, s^*) \in \Sigma$, there exist $\mathbf{x} \in B^{|X|}$ and $(\tilde{s}_2, \tilde{s}^*) \in \Sigma^{(t)}$ such that $\mathbf{u} = \lambda_2^{(u)}(\tilde{s}_2, \mathbf{xv})$, $s_2 = \delta_2(\tilde{s}_2, \mathbf{xv})$, and $s^* = \Delta(\tilde{s}^*, \mathbf{x})$. Then since $T^{(K)}(\Sigma^{(t)}, \mathbf{u}, \mathbf{v}, \Sigma^{(t+1)}) = 1$, by the condition (2) of the construction of $T^{(K)}$, (s_2, s^*) must be a member of $\Sigma^{(t+1)}$. Thus $\Sigma \subseteq \Sigma^{(t+1)}$. The same argument holds to claim that $\Sigma^{(t+1)} \subseteq \Sigma$, and thus we see that $\Sigma^{(t+1)}$ with the property above is unique. This completes the proof for the induction step.

Hence, the sequence of the global output Z realized by the behavior together with M_2 by σ is realized by M. Since σ is arbitrary, the behavior is permissible. ■

We now claim that all the permissible behaviors can be captured by the E-machine. Let us first introduce a machine *contained* in the E-machine defined as follows.

Definition: Contained Machine

> Given a finite state machine $T = (U, V, \mathcal{S}, T, \Sigma_r)$, a completely specified deterministic finite state machine $M_1 = (U, V, S_1, \lambda_1, \delta_1, r_1)$ is **contained** in T if there exists a mapping $\varphi : S_1 \rightarrow \mathcal{S}$ such that $\varphi(r_1) = \Sigma_r$ and for all $s_1 \in S_1$ and $\mathbf{u} \in B^{|U|}$, $T(\varphi(s_1), \mathbf{u}, \lambda_1(s_1, \mathbf{u}), \varphi(\delta_1(s_1, \mathbf{u}))) = 1$.

We first claim that the behavior represented by a contained machine is contained in the E-machine.

Lemma 3.1 *Consider a machine $M_1 = (U, V, S_1, \lambda_1, \delta_1, r_1)$ contained in the E-machine. The behavior of M_1 is contained in the E-machine.*

Proof: We show by induction on $t \geq 0$ that for an arbitrary input sequence on U with the length t, the output sequence of M_1 given by the input sequence can be realized by the E-machine. The claim is trivially true when $t = 0$. Consider the case where $t > 0$. Let σ_u be an arbitrary sequence on U with the length $t-1$ and let $\mathbf{u} \in B^{|U|}$ be an arbitrary minterm. Let $\tilde{s}_1$ be the state of M_1 to which σ_u leads M_1. Let σ_v be the sequence of V given by M_1 for the input sequence σ_u. By the induction hypothesis, (σ_u, σ_v) is realized by the E-machine. Since M_1 is contained in the E-machine, $T(\varphi(\tilde{s}_1), \mathbf{u}, \lambda_1(\tilde{s}_1, \mathbf{u}), \varphi(\delta_1(\tilde{s}_1, \mathbf{u}))) = 1$. Thus the pair of sequences $(\sigma_u \mathbf{u}, \sigma_v \lambda(\tilde{s}_1, \mathbf{u}))$ is realized by the E-machine, which completes the proof for the induction step. Hence the behavior of M_1 is contained in the E-machine. ■

By this lemma, all we need to show is that for an arbitrary machine M_1 that is permissible, there exists an equivalent machine contained in the E-machine.

Theorem 3.4 *For a permissible machine M_1, there exists an equivalent machine contained in the E-machine.*

Proof: By Theorem 3.1, there exists a prime machine $M_1' = (U, V, S_1, \lambda_1, \delta_1, r_1)$ which is equivalent to M_1. Let $\varphi : S_1 \rightarrow \mathcal{S}$ be a mapping such that for each state $s_1 \in S_1$, $\varphi(s_1) = \Sigma(s_1)$. Note that $\varphi(r_1) = \Sigma_r$. We claim that M_1' is contained in T under φ.

Note first that since M_1' is prime, for a state $\tilde{s}_1$ for which there is no $t \geq 0$ such that $\Sigma(\tilde{s}_1, t) \neq \phi$, $\Sigma(\tilde{s}_1) = \{\phi\}$. Then for an arbitrary $\mathbf{u} \in B^{|U|}$, and for the next state $s_1 = \delta_1(\tilde{s}_1, \mathbf{u})$, $N(\tilde{s}_1, \Sigma(\tilde{s}_1), \mathbf{u}, s_1) = \Sigma(s_1) = \{\phi\}$. Thus by the construction of the E-machine, $T(\varphi(\tilde{s}_1), \mathbf{u}, \lambda_1(\tilde{s}_1, \mathbf{u}), \varphi(s_1)) = 1$ for such $\tilde{s}_1$.

We now show by induction on $t \geq 0$ that for all states $\tilde{s}_1 \in S_1$ such that $\Sigma(\tilde{s}_1, t) \neq \phi$ and for all $\mathbf{u} \in B^{|U|}$, $T^{(t+1)}(\varphi(\tilde{s}_1), \mathbf{u}, \mathbf{v}, \varphi(s_1)) = 1$, where $\mathbf{v} = \lambda_1(\tilde{s}_1, \mathbf{u})$ and $s_1 = \delta_1(\tilde{s}_1, \mathbf{u})$. We also prove that if $\Sigma(s_1, t+1) \neq \phi$, then $\varphi(s_1) \in \mathcal{S}^{(t+1)}$, where $\mathcal{S}^{(t+1)}$ is defined in Section 3.3.1. We show that each condition for constructing the E-machine, given in Section 3.3.1, is satisfied, where $\Sigma_p = \varphi(\tilde{s}_1) = \Sigma(\tilde{s}_1)$.

(1) First, since M_1' is prime, for its reset state r_1, $\varphi(r_1)$ must be equal to $\{(r_2, \{r\})\}$, and thus $\varphi(r_1) \in \mathcal{S}^{(0)}$. In the induction step for a general t, the induction hypothesis implies that $\varphi(\tilde{s}_1) \in \mathcal{S}^{(t)}$.

Consider an arbitrary $\mathbf{x} \in B^{|X|}$ and $(\tilde{s}_2, \tilde{s}^*) \in \Sigma(\tilde{s}_1)$ such that $\mathbf{u} = \lambda_2^{(u)}(\tilde{s}_2, \mathbf{xv})$. If there is no such $\mathbf{x}$ and $(\tilde{s}_2, \tilde{s}^*)$, then the conditions (2) and (3) are trivially true with $\Sigma_n = \{\phi\}$. Therefore, $T^{(t+1)}(\varphi(\tilde{s}_1), \mathbf{u}, \mathbf{v}, \{\phi\}) = 1$. Since in this case $N(\tilde{s}_1, \Sigma(\tilde{s}_1), \mathbf{u}, s_1)$ is empty, the primeness of M_1' implies that $\Sigma(s_1) = \{\phi\}$. Therefore $T^{(t+1)}(\varphi(\tilde{s}_1), \mathbf{u}, \mathbf{v}, \varphi(s_1)) = 1$, and the proof for this case is done.

Suppose such $\mathbf{x}$ and $(\tilde{s}_2, \tilde{s}^*)$ exist. Note that in this case, $\Sigma(s_1, t+1) \neq \phi$. We first consider the second condition.

(2) Since M_1' is permissible, Theorem 3.2 implies that $\lambda_2^{(z)}(\tilde{s}_2, \mathbf{xv}) \in \Lambda(\tilde{s}^*, \mathbf{x})$. Also, since M_1' is prime, $\Sigma(s_1)$ is equal to $N(\tilde{s}_1, \Sigma(\tilde{s}_1), \mathbf{u}, s_1)$, which is denoted by N hereafter. The definition of N implies that $(\delta_2(\tilde{s}_2, \mathbf{xv}), \Delta(\tilde{s}^*, \mathbf{x})) \in N$. Therefore, $(\delta_2(\tilde{s}_2, \mathbf{xv}), \Delta(\tilde{s}^*, \mathbf{x})) \in \Sigma(s_1)$. Since $\Sigma(s_1) = \varphi(s_1)$, the second condition holds for $\Sigma_n = \varphi(s_1)$.

(3) By the equality between $\Sigma(s_1)$ and N given above, for all $(s_2, s^*) \in \Sigma(s_1)$, $(s_2, s^*) \in N$. It follows that the condition (3) is satisfied for $\Sigma_n = \varphi(s_1)$.

Therefore, $T^{(t+1)}(\varphi(\tilde{s}_1), \mathbf{u}, \mathbf{v}, \varphi(s_1)) = 1$. It follows that $\Sigma(s_1) \in \mathcal{S}^{(t+1)}$, and thus the claim above holds. Note that by this induction, we see that for each $s_1 \in S_1$, $\varphi(s_1) \in \mathcal{S}$. Hence, by the construction of T, $T(\varphi(\tilde{s}_1), \mathbf{u}, \mathbf{v}, \varphi(s_1)) = 1$ for all $\tilde{s}_1 \in S_1$ and for all $\mathbf{u} \in B^{|U|}$. ■

We have now reached the key statement of the E-machine.

Corollary 3.1 *A behavior implementable at M_1 is permissible if and only if it is contained in the E-machine.*

Thus, the complete set of permissible behaviors can be captured by the E-machine. By Theorem 3.4, however, we see that the E-machine contains the permissible behaviors using contained machines. In this sense, we can say that the E-machine tells not only which behavior is permissible, but also how the behavior can be realized by a finite state machine. In other words, the following claim holds for permissible machines.

Corollary 3.2 *A finite state machine implementable at M_1 is permissible if and only if there exists an equivalent machine contained in the E-machine.*

Proof: If M_1 is a permissible machine, then by Theorem 3.4, we see that there exists an equivalent machine contained in the E-machine. Suppose that for an implementable machine M_1, there exists an equivalent machine contained in the E-machine. Lemma 3.1 implies that the behavior of M_1 is contained in the E-machine. Thus, by Theorem 3.3, M_1 is permissible. ■

3.3.3 The Structure of the E-machine and a Non-Deterministic Construction

As seen so far, the E-machine is in general a non-deterministic finite state machine, i.e. for a given state and input, the next state and the corresponding output may not be unique. This nature allows us to represent a set of behaviors by a single machine. However, we note that the E-machine is a special type of non-deterministic machine. Namely, for a given state $\Sigma_p \in \mathcal{S}$ and pair of input and output minterms $(\mathbf{u}, \mathbf{v}) \in B^{|U|} \times B^{|V|}$, if there exists a next state Σ_n such that $T(\Sigma_p, \mathbf{u}, \mathbf{v}, \Sigma_n) = 1$, then such Σ_n is unique. In other words, if we introduce a set of new symbols and replace each pair of input and output minterms of the E-machine by one of the symbols, then the resulting machine is a deterministic finite automaton[3]. This is true since in the construction of the E-machine, we uniquely define the next state, if exists, for a given pair of input and output minterms.

One may ask whether it is possible to construct the E-machine, so that the automaton corresponding to the machine is non-deterministic and accepts the same language with the original. In other words, if we perform the subset construction to determinize the non-deterministically constructed E-machine, where we assume each pair of input and output minterms is a single symbol, then can we obtain exactly the same E-machine as the one defined in Section 3.3.1? It is interesting to construct and represent the E-machine this way, since it is known that the subset construction, or determinization, introduces an exponentially large number of states in general. Thus we expect that the non-deterministically constructed E-machine has a smaller state space; the complete set of permissible behaviors is then represented in the more compact way.

In this section, we consider the case where the global machine M is a completely specified deterministic machine, and present a procedure, suggested by Alex Saldanha, which generates a machine such that by performing an operation similar to the subset construction we obtain the E-machine as originally defined. We recall that in case M is a deterministic machine, a state of the E-machine defined in Section 3.3.1 corresponds to a set of pairs of states of M_2 and M. A state of the machine which we will construct (called the NDE-machine) corresponds to a pair of states of M_2 and M, rather than a set of pairs.

The procedure is a fixed point iteration. We denote the transition relation and the set of states at the t-th step by $T_N^{(t)}$ and $S_N^{(t)}$ respectively, where the subscript N implies that the construction leads to

[3]We call a finite state machine with this property a pseudo non-deterministic finite state machine.

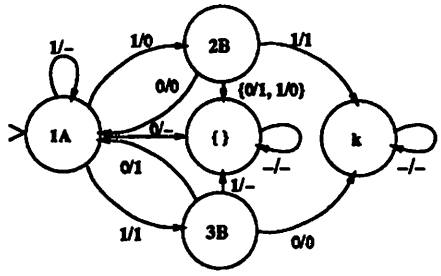

Figure 9: The Non-Deterministic E-machine

a non-deterministic E-machine in the sense above. Initially, $S_N^{(0)} = \{(r_2, r), \phi, \kappa\}$, where r_2 and r are the reset states of M_2 and M while ϕ and κ are newly introduced states. The initial transition relation $T_N^{(0)} : S_N^{(0)} \times B^{|U|} \times B^{|V|} \times S_N^{(0)} \to B$ is defined as $T_N^{(0)}(\varsigma_p, \mathbf{u}, \mathbf{v}, \varsigma_n) = 1$ if and only if either $\varsigma_p = \varsigma_n = \phi$ or $\varsigma_p = \varsigma_n = \kappa$, i.e. we start with only self-loops on ϕ and κ. In general for the step $(t+1)$, $T_N^{(t+1)}(\varsigma_p, \mathbf{u}, \mathbf{v}, \varsigma_n)$ is defined when $\varsigma_p \in S_N^{(t)} - \{\phi, \kappa\}$, i.e. the present state ς_p is a pair of states of M_2 and M, say $\varsigma_p = (\tilde{s}_2, \tilde{s})$, which has been introduced as a state in the transition relation $T_N^{(t)}$. Then $T_N^{(t+1)}(\varsigma_p, \mathbf{u}, \mathbf{v}, \varsigma_n) = 1$ if and only if one of the following three conditions holds:

(a) $\forall \mathbf{x} \in B^{|X|} : \mathbf{u} \neq \lambda_2^{(u)}(\tilde{s}_2, \mathbf{xv})$ and $\varsigma_n = \phi$, or
(b) $\exists \mathbf{x} \in B^{|X|} : \mathbf{u} = \lambda_2^{(u)}(\tilde{s}_2, \mathbf{xv})$ and $\lambda_2^{(z)}(\tilde{s}_2, \mathbf{xv}) \neq \lambda(\tilde{s}, \mathbf{x})$ and $\varsigma_n = \kappa$, or
(c) $\forall \mathbf{x} \in B^{|X|} : \mathbf{u} = \lambda_2^{(u)}(\tilde{s}_2, \mathbf{xv}) \Rightarrow \lambda_2^{(z)}(\tilde{s}_2, \mathbf{xv}) = \lambda(\tilde{s}, \mathbf{x})$ and
$\exists \mathbf{x} \in B^{|X|} : \mathbf{u} = \lambda_2^{(u)}(\tilde{s}_2, \mathbf{xv})$ and $\varsigma_n = (\delta_2(\tilde{s}_2, \mathbf{xv}), \delta(\tilde{s}, \mathbf{x}))$,

where $\lambda : S \times B^{|X|} \to B^{|Z|}$ and $\delta : S \times B^{|X|} \to S$ are the output and the next state functions of M, which are defined since M is deterministic. Condition (a) says that if there is no $\mathbf{x}$ which causes M_2 to output $\mathbf{u}$ at the state $\tilde{s}_2$ for the input $\mathbf{v}$, then we cause a transition to ϕ. Condition (b) means that if there exists an $\mathbf{x}$ which causes M_2 to output $\mathbf{u}$ at $\tilde{s}_2$ for $\mathbf{v}$ but the z output is not allowed, then we cause a transition to κ. Finally (c), if all possible z outputs are allowed and if there is at least one $\mathbf{x}$ that makes M_2 and M transit to ς_n, then this transition is put in $T_N^{(t+1)}$.

Let T_N be the transition relation of the fixed point of the computation. Namely, for positive integer K, if $S_N^{(K)} = S_N^{(K-1)}$, then $T_N = T_N^{(K)}$. Similarly, let $S_N = S_N^{(K)}$. Call the resulting finite state machine $(U, V, S_N, T_N, \varsigma_r)$ the non-deterministic E-machine, or NDE-machine for short, where $\varsigma_r = (r_2, r)$. The transition relation of the NDE-machine for M_2 and M used in Example 1 is shown in Figure 9, where the states ϕ and κ are denoted respectively by $\{\}$ and k. Note that unlike the E-machine, the NDE-machine has a property that for a state ς_p and pair of input and output minterms $(\mathbf{u}, \mathbf{v})$, there might exist more than one state ς_n such that $T_N(\varsigma_p, \mathbf{u}, \mathbf{v}, \varsigma_n) = 1$. It is because of the global input X. Namely, for different global input $\mathbf{x} \in B^{|X|}$, M_2 and M may go to different next states with the same $(\mathbf{u}, \mathbf{v})$.

Now, for a given NDE-machine $(U, V, S_N, T_N, \varsigma_r)$, consider a finite state machine $(U, V, S_D, T_D, \Sigma_{D_r})$

defined as follows. The state space $\mathcal{S}_D$ is the set of subsets of $\mathcal{S}_N$ that contain ϕ and not contain κ. The reset state Σ_{D_r} is the subset $\{\varsigma_r, \phi\}$. The transition relation $T_D : \mathcal{S}_D \times B^{|U|} \times B^{|V|} \times \mathcal{S}_D \to B$ is defined as $T_D(\Sigma_{D_p}, \mathbf{u}, \mathbf{v}, \Sigma_{D_n}) = 1$ if and only if

$$\Sigma_{D_n} = \{\varsigma_n \in \mathcal{S}_N \mid \exists \varsigma_p \in \Sigma_{D_p} : T_N(\varsigma_p, \mathbf{u}, \mathbf{v}, \varsigma_n) = 1\} \text{ and } \kappa \notin \Sigma_{D_n}$$

This construction is the subset construction, or determinization, of a non-deterministic finite automaton, where the state κ is the unique non-accepting state, meaning that a string which *can* lead the automaton to κ is not accepted. Only subsets, generated in the subset construction, which do not contain κ are allowed next-state subsets. In this way, we end up with a finite state machine which contains only permissible behaviors.

Let $\mathcal{S}'_D$ be the union of the state $\{\phi\} \in \mathcal{S}_D$ and the set of states of T_D reachable from the reset state Σ_{D_r}. Let $T'_D : \mathcal{S}'_D \times B^{|U|} \times B^{|V|} \times \mathcal{S}'_D \to B$ be the transition relation of T_D restricted to the states $\mathcal{S}'_D$. We then claim that the restricted machine $T'_D = (U, V, \mathcal{S}'_D, T'_D, \Sigma_{D_r})$ is the E-machine. More specifically, T'_D and the E-machine are isomorphic, i.e. there exists a one-to-one mapping f from the state space of the E-machine to that of T'_D such that $T(\Sigma_p, \mathbf{u}, \mathbf{v}, \Sigma_n) = 1$ if and only if $T'_D(f(\Sigma_p), \mathbf{u}, \mathbf{v}, f(\Sigma_n)) = 1$.

Theorem 3.5 *The machine T'_D and the E-machine are isomorphic.*

Proof: Given a subset Σ of pairs of states of M_2 and M, let $f(\Sigma)$ be the subset given by adding the state ϕ to Σ, i.e. $f(\Sigma) = \Sigma \cup \{\phi\}$. For the special case for the empty set $\{\phi\}$, we define $f(\{\phi\}) = \{\phi\}$. By definition, f is a one-to-one mapping and thus the inverse of f is also defined. We claim that T'_D and the E-machine are isomorphic under the mapping f.

Suppose $T(\Sigma_p, \mathbf{u}, \mathbf{v}, \Sigma_n) = 1$ holds in the E-machine. If $\Sigma_p = \{\phi\}$, then by the definition of the E-machine, $\Sigma_n = \{\phi\}$. By construction of the NDE-machine T_N, for the present state $\phi \in \mathcal{S}_N$, ϕ is the unique state that satisfies $T_N(\phi, \mathbf{u}, \mathbf{v}, \phi) = 1$. Therefore, $T_D(\{\phi\}, \mathbf{u}, \mathbf{v}, \{\phi\}) = 1$. Since $f(\{\phi\}) = \{\phi\}$, $T'_D(f(\Sigma_p), \mathbf{u}, \mathbf{v}, f(\Sigma_n)) = 1$, and the claim holds.

Consider the case where $\Sigma_p \neq \{\phi\}$. We show $T'_D(f(\Sigma_p), \mathbf{u}, \mathbf{v}, f(\Sigma_n)) = 1$ under the assumption that $f(\Sigma_p) \in \mathcal{S}'_D$. This assumption does not affect the claim, since $T'_D(f(\Sigma_p), \mathbf{u}, \mathbf{v}, f(\Sigma_n)) = 1$ implies that $f(\Sigma_n) \in \mathcal{S}'_D$ and for the reset state $\Sigma_r = \{(r_2, r)\}$, $f(\Sigma_r) \in \mathcal{S}'_D$. By construction of the E-machine, the next state Σ_n from Σ_p under $\mathbf{u}/\mathbf{v}$ is given by

$$\Sigma_n = \{(s_2, s) \in S_2 \times S \mid \exists \mathbf{x} \in B^{|X|}, (\bar{s}_2, \bar{s}) \in \Sigma_p : \begin{array}{l} \mathbf{u} = \lambda_2^{(u)}(\bar{s}_2, \mathbf{xv}), \quad s_2 = \delta_2(\bar{s}_2, \mathbf{xv}), \\ s = \delta(\bar{s}, \mathbf{x}) \end{array} \}.$$

Consider arbitrary $\mathbf{x} \in B^{|X|}$ and $(\bar{s}_2, \bar{s}) \in \Sigma_p$ such that $\mathbf{u} = \lambda_2^{(u)}(\bar{s}_2, \mathbf{xv})$. If there are no such $\mathbf{x}$ and $(\bar{s}_2, \bar{s})$, then $\Sigma_n = \{\phi\}$. In this case, for an arbitrary pair of states of M_2 and M contained in $f(\Sigma_p)$, only condition (a) holds in the definition of the NDE-machine. Therefore, for all elements $\varsigma_p \in f(\Sigma_p)$, $\varsigma_n = \phi$ is the unique state which satisfies $T_N(\varsigma_p, \mathbf{u}, \mathbf{v}, \varsigma_n) = 1$. Thus we obtain $f(\Sigma_n) = \{\varsigma_n \mid \exists \varsigma_p \in f(\Sigma_p) : T_N(\varsigma_p, \mathbf{u}, \mathbf{v}, \varsigma_n) = 1\}$. Hence $T'_D(f(\Sigma_p), \mathbf{u}, \mathbf{v}, f(\Sigma_n)) = 1$.

Suppose there exist such $\mathbf{x}$ and $(\bar{s}_2, \bar{s}) \in \Sigma_p$ with the property that $\mathbf{u} = \lambda_2^{(u)}(\bar{s}_2, \mathbf{xv})$. Then by the definition of the E-machine, $\lambda_2^{(z)}(\bar{s}_2, \mathbf{xv}) = \lambda(\bar{s}, \mathbf{x})$. Therefore, in the definition of the NDE-machine, the condition (a) and (b) do not hold and the first half of the condition (c) holds. Hence, Σ_n given above can be rewritten as

$$\Sigma_n = \{(s_2, s) \in S_2 \times S \mid \exists (\bar{s}_2, \bar{s}) \in \Sigma_p : T_N((\bar{s}_2, \bar{s}), \mathbf{u}, \mathbf{v}, (s_2, s)) = 1\}.$$

Thus by definition of T_D, $T'_D(f(\Sigma_p), \mathbf{u}, \mathbf{v}, f(\Sigma_n)) = 1$.

Conversely, suppose $T'_D(\Sigma_{D_p}, \mathbf{u}, \mathbf{v}, \Sigma_{D_n}) = 1$. We will show $T(f^{-1}(\Sigma_{D_p}), \mathbf{u}, \mathbf{v}, f^{-1}(\Sigma_{D_n})) = 1$, where f^{-1} is the inverse of f. Note that the function $f^{-1}(\Sigma_{D_p})$ simply removes the state $\phi \in S_N$ from the subset Σ_{D_p}, where in case $\Sigma_{D_p} = \{\phi\}$, $f^{-1}(\Sigma_{D_p}) = \{\phi\}$. We employ the assumption that $f^{-1}(\Sigma_{D_p}) \in \mathcal{S}$. The assumption does not affect the claim for the same reason above. If $\Sigma_{D_p} = \{\phi\}$, then $\Sigma_{D_n} = \{\phi\}$. Since $T(\{\phi\}, \mathbf{u}, \mathbf{v}, \{\phi\}) = 1$, the claim holds.

Consider the case where $\Sigma_{D_p} \neq \{\phi\}$. Since $f^{-1}(\Sigma_{D_p}) \in \mathcal{S}$, there exists some t such that $f^{-1}(\Sigma_{D_p}) \in \mathcal{S}^{(t)}$, where $\mathcal{S}^{(t)}$ is defined in Section 3.3.1. We will show that conditions (2) and (3) defined in the definition of $T^{(K)}$ in Section 3.3.1 hold, and thus $T^{(t+1)}(f^{-1}(\Sigma_{D_p}), \mathbf{u}, \mathbf{v}, f^{-1}(\Sigma_{D_n})) = 1$. Hereafter, let us denote $\Sigma_p = f^{-1}(\Sigma_{D_p})$ and $\Sigma_n = f^{-1}(\Sigma_{D_n})$. Consider arbitrary $\mathbf{x} \in B^{|X|}$ and $(\bar{s}_2, \bar{s}) \in S_2 \times S$ such that $(\bar{s}_2, \bar{s}) \in \Sigma_{D_p}$ and $\mathbf{u} = \lambda_2^{(u)}(\bar{s}_2, \mathbf{xv})$. If there are no such $\mathbf{x}$ and $(\bar{s}_2, \bar{s})$, then the condition (2) trivially holds. Also in this case, for each $\varsigma_p = (\bar{s}_2, \bar{s}) \in \Sigma_{D_p}$, only the condition (a) holds in the construction of the NDE-machine T_N, and thus $\Sigma_{D_n} = \{\phi\}$. Since condition (3) trivially holds if $\Sigma_n = \{\phi\}$, we obtain $T^{(t+1)}(\Sigma_p, \mathbf{u}, \mathbf{v}, \Sigma_n) = 1$.

Suppose such $\mathbf{x}$ and $(\bar{s}_2, \bar{s})$ exist. Since $\kappa \notin \Sigma_{D_n}$, condition (b) does not hold for this pair $(\bar{s}_2, \bar{s})$ in the definition of the NDE-machine. Thus the first half of the condition (c) holds, and $\lambda_2(\bar{s}_2, \mathbf{xv}) = \lambda(\bar{s}, \mathbf{x})$. Since $(\delta_2(\bar{s}_2, \mathbf{xv}), \delta(\bar{s}, \mathbf{x})) \in \Sigma_{D_n}$, by definition of T_D, $(\delta_2(\bar{s}_2, \mathbf{xv}), \delta(\bar{s}, \mathbf{x})) \in \Sigma_n$, and the condition (2) holds.

For condition (3), consider an arbitrary $(s_2, s) \in \Sigma_n$. Since $(s_2, s) \in \Sigma_{D_n}$, there exists $(\bar{s}_2, \bar{s}) \in \Sigma_{D_p}$ such that $T_N((\bar{s}_2, \bar{s}), \mathbf{u}, \mathbf{v}, (s_2, s)) = 1$. Hence, condition (c) in the definition of the NDE-machine holds, and there exists $\mathbf{x} \in B^{|X|}$ such that $\mathbf{u} = \lambda_2^{(u)}(\bar{s}_2, \mathbf{xv})$ and $(s_2, s) = (\delta_2(\bar{s}_2, \mathbf{xv}), \delta(\bar{s}, \mathbf{x}))$. Thus condition (3) holds, and we obtain $T^{(t+1)}(\Sigma_p, \mathbf{u}, \mathbf{v}, \Sigma_n) = 1$. ■

Thus, we see that the E-machine can be obtained by applying an operation similar to the subset construction to the NDE-machine T_N. One might wonder why the operation like the subset construction is necessary. In other words, how is the set of behaviors contained in the E-machine related to that of the NDE-machine? The answer is that the NDE-machine contains *more* implementable behaviors than the E-machine. Specifically, in the NDE-machine, an implementable behavior is not permissible if there exists a pair (σ_u, σ_v) of sequences of U and V in the behavior which can lead the NDE-machine to the state κ, since it means that the corresponding sequence on the global output Z is inconsistent with what is required by M.[4] Therefore, we need to know the set of pairs that have a possibility to lead the NDE-machine to κ. It is analogous to finding the set of strings that have a possibility to lead a non-deterministic finite automaton to an accepting state. Hence, we employ the subset construction to remove those additional behaviors, and then guarantee that an arbitrary implementable behavior contained in the resulting machine (E-machine) is permissible. It is illustrated in the following example.

Example 3 *Consider M_2 and M shown in Figure 10, which are slightly different from those used in Example 1. The corresponding E-machine and the NDE-machine are shown in Figure 11-(a) and Figure 11-(b) respectively.*

Consider a behavior at M_1 which always outputs 0 *for all input sequences. This is equivalent to setting the variable V to a constant* 0*, and thus the behavior is implementable. However, the behavior is not*

[4]Note that the pair (σ_u, σ_v) is not allowed even if it can also lead the NDE-machine to a state other than κ.

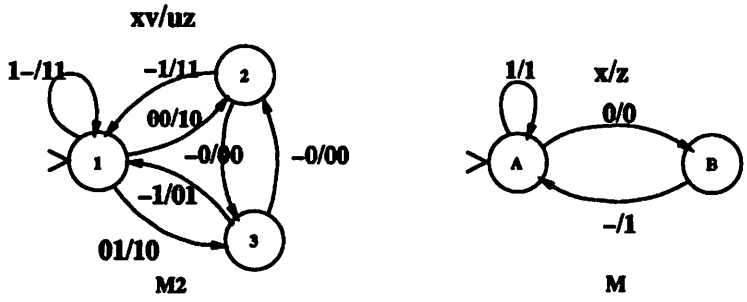

Figure 10: Example of M_2 and M

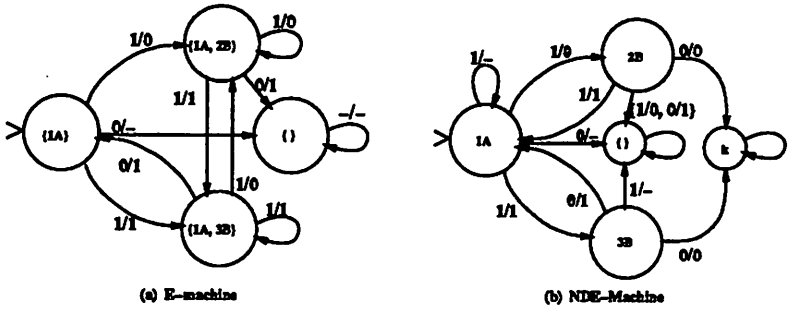

Figure 11: The E-machine (left) and the NDE-machine (right)

permissible since if a sequence σ_x of the global input X is set to $(0,0)$, then the corresponding pair of sequences (σ_u, σ_v) on U and V realized by the behavior and M_2 is given by $\sigma_u = (1,0)$ and $\sigma_v = (0,0)$, and thus the global output sequence σ_z is obtained as $\sigma_z = (0,0)$, while the global machine M requires that σ_z must be $(0,1)$. It is easy to see that the pair (σ_u, σ_v) above can lead the NDE-machine to the state κ through the states $1A$ and $2B$. Note that this behavior is not contained in the E-machine.

3.4 Implementability of Interacting Machines

3.4.1 Implementability

As we have seen in the previous sections, the permissibility of M_1 requires that M_1 is implementable, i.e. there exists a pair of implementations for M_1 and M_2 where no combinational loop is created by connecting them together at U and V. Therefore, when a permissible machine is sought, we need to check whether the machine is implementable or not. In this section, we provide a condition on the implementability.

Let a completely specified deterministic machine $M_2 = (X \cup V, U \cup Z, S_2, \lambda_2, \delta_2, r_2)$ be given. We want to know if a given completely specified deterministic machine $M_1 = (U, V, S_1, \lambda_1, \delta_1, r_1)$ is implementable for the M_2. The key idea to check the implementability is the *dependencies*. Let us first present the following definition.

Definition: Dependencies

For a set of Boolean variables $X = \{x_1, \ldots, x_n\}$ consider a function $f : B^n \rightarrow B$ defined with the input X. Given an input variable $x_i \in X$, f is **dependent** on x_i if $f|_{x_i=0} \neq f|_{x_i=1}$, where $f|_{x_i=0}$ designates the cofactor of f with respect to $x_i = 0$.

If f is not dependent on x_i, we say that f is independent of x_i. The dependency of f for an input x_i is related to whether it is possible to implement the function f with no combinational path from x_i to the output, where we define a combinational path as a sequence of gates which does not contain registers. More specifically, the following lemma is known.

Lemma 3.2 *Given a function $f : B^n \rightarrow B$ with the input $X = \{x_1, \ldots, x_n\}$, there exists an implementation for f such that there is no combinational path from x_i to the output if and only if f is not dependent on x_i.*

Proof: Suppose there exists such an implementation. Then for an arbitrary minterm $\mathbf{x} \in B^n$, the output value $f(\mathbf{x})$ does not change even if we flip the value of x_i in the minterm $\mathbf{x}$. Thus $f|_{x_i=0} = f|_{x_i=1}$.

Conversely, suppose that f is not dependent on x_i. Consider an implementation of f. If the implementation does not contain a combinational path from x_i to the output, the proof is done. Suppose there is a combinational path. We claim that the implementation given by setting x_i to a constant value, say 0, still implements f. Let $\tilde{f}$ be the function defined by the resulting implementation. Note that $\tilde{f}$ does not depend on x_i. The proof is done if we show that $\tilde{f}(\mathbf{x}) = f(\mathbf{x})$ for all $\mathbf{x} \in B^n$. Suppose $\tilde{f}(\mathbf{x}) \neq f(\mathbf{x})$. Then the value of x_i in the minterm $\mathbf{x}$ must be 1 since $\tilde{f}$ is obtained by setting $x_i = 0$ in f. Then $f|_{x_i=0} \neq f|_{x_i=1}$, which is conflict with the fact that f is not dependent on x_i. ■

We now present a condition under which M_1 is implementable. Consider a directed bipartite graph $G(U \cup V, E)$, where the node set of G is divided into two classes U and V and a node of U (respectively a node of V) corresponds to a variable of the input variables U (respectively the output variables V) of M_1. The edges of G are defined as follows:

$$\begin{aligned} [u_i, v_j] \in E &\Leftrightarrow \lambda_1^{(v_j)} \text{ depends on } u_i, \\ [v_j, u_i] \in E &\Leftrightarrow \lambda_2^{(u_i)} \text{ depends on } v_j, \end{aligned}$$

where we denote by $\lambda_1^{(v_j)}$ the function of the j-th output variable v_j in M_1.

Theorem 3.6 *M_1 is implementable if and only if G is acyclic.*

Proof: Suppose M_1 is implementable. Then there exists a pair of implementations (C_1, C_2) for M_1 and M_2 respectively which does not create a combinational loop. Let $G_c(U \cup V, E_c)$ be a directed bipartite graph with the same node set of G where the edges are defined as follows:

$$\begin{aligned} [u_i, v_j] \in E &\Leftrightarrow \text{there exists a combinational path from } u_i \text{ to } v_j \text{ in } C_1, \\ [v_j, u_i] \in E &\Leftrightarrow \text{there exists a combinational path from } v_j \text{ to } u_i \text{ in } C_2. \end{aligned}$$

Since the implementation does not contain a combinational loop, G_c is acyclic. Now, if $\lambda_1^{(v_j)}$ depends on u_i, Lemma 3.2 implies that C_1 has a combinational path from u_i to v_j. A similar argument holds for $\lambda_2^{(u_i)}$, and we see that $E \subseteq E_c$. Hence G is a subgraph of G_c, and G is acyclic.

Conversely, suppose G is acyclic. If we implement the function $\lambda_1^{(v_j)}$ independently for each v_j, we obtain an implementation C_1 of M_1 where there is a combinational path from u_i to v_j if and only if $\lambda_1^{(v_j)}$ depends on u_i. Similarly, let C_2 be an implementation of M_2 such that there is a combinational path from v_j to u_i if and only if $\lambda_2^{(u_i)}$ depends on v_j. The proof is done if we show that (C_1, C_2) does not create a combinational loop. Suppose for contrary that there exists a combinational loop $c = (v_{j_0}, u_{i_0}, \ldots, v_{j_k}, u_{i_k}, v_{j_0})$. Then for each l, $0 \leq l \leq k$, $\lambda_1^{(v_{j_l})}$ depends on u_{i_l}. Similarly, $\lambda_2^{(u_{i_l})}$ depends on $v_{j_{l+1}}$, where we define $v_{j_{k+1}} = v_{j_0}$. Thus the cycle c exists in G, which is conflict with the fact that G is acyclic. ■

Since the cyclicity of a directed bipartite graph can be checked in polynomial time of the size of the graph, we can efficiently check the implementability of M_1. Note that if either M_1 or M_2 is of Moore type, then G is always acyclic, and M_1 is implementable.

3.4.2 Unimplementable Machines in the E-machine

In general, not all the machines contained in the E-machine are implementable. By definition of implementable machines, if a machine M_1 contained in the E-machine is not implementable, then any implementation of M_2 will create a combinational loop for that particular M_1. Thus, for given M and M_2, if the resulting E-machine contains no implementable machines, we see that it is impossible to realize a behavior of M without combinational loops, as long as the behavior of M_2 is used.

In this section, we discuss what we can do with unimplementable machines of the E-machine. Specifically, we show that for a machine M_1 contained in the E-machine that is not implementable, if M_1 satisfies a certain condition, then it is possible to realize a behavior of M with no combinational loops, as long as we are allowed to modify the behavior of M_2.

Let $M_1 = (U, V, S_1, \lambda_1, \delta_1, r_1)$ be a machine contained in the E-machine. Suppose that M_1 is not implementable. Suppose also that M_1 satisfies the following property:

Property 3.1 *For all pairs of states,* $(s_1, s_2) \in S_1 \times S_2$, *and for all* $\mathbf{x} \in B^{|X|}$, *there exists* $(\mathbf{u}, \mathbf{v}) \in B^{|U|} \times B^{|V|}$ *such that* $\mathbf{v} = \lambda_1(s_1, \mathbf{u})$ *and* $\mathbf{u} = \lambda_2(s_2, \mathbf{xv})$.

Consider a pair of implementations C_1 and C_2 for M_1 and M_2, respectively. Since M_1 is not implementable, the implementation made of C_1 and C_2 creates a combinational loop. Now, we first assume that we can scan the registers of C_1, i.e. it is possible to observe externally the state in which M_1 stays. Then we modify C_2 so that the resulting implementation has no combinational loop and realizes a behavior of M.

Consider a function whose inputs are the global inputs X and the states of M_1 and M_2, and the outputs are U. We denote the function by $f : S_1 \times S_2 \times B^{|X|} \rightarrow B^{|U|}$. For given $(s_1, s_2) \in S_1 \times S_2$ and $\mathbf{x} \in B^{|X|}$, the output $\mathbf{u} = f(s_1, s_2, \mathbf{x})$ has a property that there exists $\mathbf{v} \in B^{|V|}$ such that $\mathbf{v} = \lambda_1(s_1, \mathbf{u})$ and $\mathbf{u} = \lambda_2(s_2, \mathbf{xv})$. Since M_1 satisfies Property 3.1, $f(s_1, s_2, \mathbf{x})$ is defined for every input. Let C_3 be an implementaton of f. Note that C_3 is a combinational logic. We break the connection from C_2 to C_1 at U and let C_3 drive C_1, as shown in Figure 12. Since all the feedbacks from C_1 to C_3 and from C_2 to C_3 are to see the states of M_1 and M_2, there is no combinational loop in the resulting implementation.

Let us regard the circuit made of C_2 and C_3 as an implementation of a single deterministic finite state machine $\tilde{M}_2$. Note that the state space of $\tilde{M}_2$ is identical with that of M_2. By the construction of the function f, it is guaranteed that for all pairs of states, $(s_1, s_2) \in S_1 \times S_2$, and for all $\mathbf{x} \in B^{|X|}$, the pair $(\mathbf{u}, \mathbf{v}) \in B^{|U|} \times B^{|V|}$ realized by M_1 and $\tilde{M}_2$ has the property that $\mathbf{v} = \lambda_1(s_1, \mathbf{u})$ and $\mathbf{u} = \lambda_2(s_2, \mathbf{xv})$.

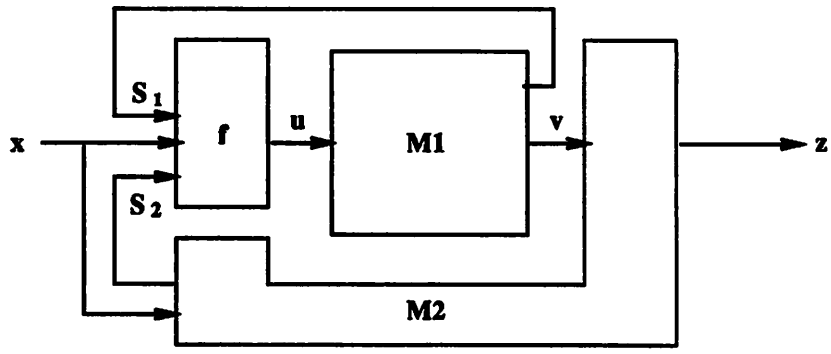

Figure 12: Modification for Unimplementable Machines

Furthermore, the state transition of $\tilde{M}_2$ does not depend on the states of M_1. Namely, for a given state s_2 of $\tilde{M}_2$ and input $\mathbf{xv} \in B^{|X \cup V|}$, the next state to which $\tilde{M}_2$ moves is uniquely defined and is given by $\delta_2(s_2, \mathbf{xv})$. Since M_1 is contained in the E-machine, exactly the same proof of Theorem 3.3 holds to claim that for an arbitrary sequence σ of $B^{|X|}$, the output sequence realized by $M_1 \times \tilde{M}_2$ can be realized by M, and therefore the behavior of $M_1 \times \tilde{M}_2$ is contained in M. We state this fact as a theorem below.

Theorem 3.7 *For a machine M_1 contained in the E-machine, suppose M_1 is not implementable and satisfies Property 3.1 above. Then for an arbitrary pair of implementations C_1 and C_2 for M_1 and M_2, if an additional circuitry C_3 given above is attached, the resulting circuitry has no combinational loop and its behavior is contained in M.*

4 Implementation and Experiments

The method of computing the transition relation T of the E-machine has been implemented. The current implementation employs a restriction that the global machine M is deterministic, and thus a state of the E-machine corresponds to a subset of pairs of states of M_2 and M. Binary decision diagrams (BDD's) [4] are used to represent the transition relations of M_2 and M, where the set of states of each machine was represented by binary variables using log-based encodings. All the set operations, such as intersection, union, complement, set comparisons, as well as quantifications, are performed on BDD's. We first compute the relation $T^{(K)}$ and then T. One straightforward way of computing $T^{(K)}$ is to first compute the relation given by the condition (2) and (3) of the definition of $T^{(K)}$ shown in Section 3.3.1, and then restrict it to the states that T can be led to by some sequences of $B^{|U|}$. However, since the total number of states of the finite state machine given by the conditions (2) and (3) is exponential in $|S_2||S|$, the BDD representing the transition relation of the machine may be too large. Instead, we perform a fixed point computation as stated in Section 3.3.1, where at each step t, instead of the set $\mathcal{S}^{(t)}$, we use a set which contains $\mathcal{S}^{(t)} \cap \neg\mathcal{S}^{(t-1)}$, is contained in $\mathcal{S}^{(t)}$, and is represented by a minimal-sized BDD. Such a set is computed by a BDD operation similar to the one known as generalized cofactor [5]. During the computation, we need to see if a given pair of states $(\bar{s}_2, \bar{s}) \in S_2 \times S$ is a member of Σ_p. For this purpose, we use a characteristic function $\chi(\bar{s}_2, \bar{s}, \Sigma_p)$

	M_2			M			T				
	In	**Out**	**State**	**In**	**Out**	**State**	**In**	**Out**	**State**	**Time**	**Iterations**
mc9	3	5	4	2	3	16	2	1	3	0.2	3
s2a9	11	3	18	10	1	72	2	1	2	1.9	2
e4t1	6	9	14	2	2	294	7	4	5	3.4	5
e69	5	8	8	4	6	32	2	1	7	22.3	3
s3t2	7	7	13	3	2	273	5	4	35	136.6	12
e4bp1	6	9	14	1	4	336	5	5	10	149.3	10
e4at2	7	9	14	2	4	294	5	4	13	449.4	13
s3p1	7	7	13	2	2	312	5	5	37	669.5	11
s2b9	11	3	18	10	1	72	2	1	19	4132.3	18

Table 1: Experimental Results

which is equal to 1 if and only if $(\bar{s}_2, \bar{s}) \in S_2 \times S$ is a member of Σ_p. However, a BDD representing the function itself, or a BDD obtained at an intermediate stage of the computations using the function, could be fairly large in practice. Therefore, we represent χ by multiple number of BDD's so that the union of these BDD's forms the function χ. We modified the formula given in (2) and (3) in the definition of $T^{(K)}$ so that the union of these BDD's are taken as late as possible by applying other commutative operations earlier. These heuristics seem to be effective in keeping the size of BDD's as small as possible.

Using the procedure implemented as stated above, we conducted some preliminary experiments. They are preliminary in the sense that the examples are not obtained during a practical design process of digital systems. We used mcnc91 benchmark examples. The objective of the experiment was to determine the size of machines M_2 and M that can be handled by the current implementation, and the size of the resulting E-machine T since its state-space size could be exponential in $|S_2||S|$. We first chose a pair of finite state machines, one for M_1, the other for M_2, and obtained M by taking the product of these, where a subset of the input variables (the output variables, respectively) of M_2 was arbitrarily chosen to connect with the output (the input, respectively) of M_1. Then M_2 and M were used as inputs of the procedure. The procedure first computes the set of states reachable from the reset state for each machine, and then computes the transition relations of the machines as well as the function χ described above. It computes the relation $T^{(K)}$ by a fixed point computation, and finally computes the transition relation T.

The results on these examples are shown in Table 1. Each row of the table corresponds to a single experiment, where **In**, **Out**, and **State** designate the number of input variables, the number of output variables, and the number of states respectively. **Time** is the CPU time used for each experiment in seconds on a DECstation 5000/240. **Iterations** shows the number of iterations required in the fixed point computation of $T^{(K)}$. As shown in Table 1, we can handle moderate sized examples with the current implementation. During the experiments, we realized that the size of the resulting E-machine T and the required CPU time vary by changing the connections between M_1 and M_2. For example, we used exactly the same machines M_1 and M_2 for the experiments **s2a9** and **s2b9** with different connections. A large difference in CPU time and number of states is observed between the two experiments. Thus, we can not make any general statement on the size of T that we can handle in practice. Nevertheless, for these experimental results, we see that the

number of states of T is negligibly smaller than $2^{|S_2||S|}$. This is not surprising in the sense that a state s_1 of T corresponds to a subset of $S_2 \times S$ with the property that M_2 and M are led to exactly the states of the subset by the input sequences of $B^{|X|}$ and the sequences of $B^{|V|}$ realized by transitions from the reset state of T to s_1. Thus if there exists a pair of states (s_2, s) not led to by any input sequence, then any of the subsets of $S_2 \times S$ which contains the pair will not appear in T, where there are $2^{(|S_2||S|-1)}$ such subsets.

5 Conclusions

In this paper, we addressed the problem of computing and representing the complete set of permissible finite state machines, where two finite state machines are interacting with each other as shown in Figure 2. We showed that the complete set can be computed and represented by a single non-deterministic finite state machine. The machine is called the E-machine and its transition relation is computed by a fixed point computation. We also considered the problem of implementing interacting finite state machines without introducing combinational loops, and provided a necessary and sufficient condition under which given machines are implementable. The proposed procedure for computing the E-machine was implemented and experimental results were presented.

In the future, we intend to address the problem of minimizing the E-machines, i.e. finding the *best* permissible behavior of M_1 for given M_2 and M.

6 Acknowledgements

The authors thank Dr. Alex Saldanha who indicated an application for the supervisory control problem and suugested the procedure presented in Section 3.3.3.

Appendix
A Proof of Theorem 3.1

In this Appendix, we prove Theorem 3.1.

Theorem 3.1 *For an implementable machine M_1, there exists an equivalent prime machine.*

Proof: We prove the theorem by presenting a procedure which takes as input an implementable machine $M_1 = (U, V, S_1, \lambda_1, \delta_1, r_1)$ and returns an equivalent prime machine M_1'. The procedure is shown in Figure 13.

The procedure first duplicates M_1, where S_1' is set to S_1, λ' and δ' are identical with λ and δ, respectively. The transitions of M_1' are then modified during the procedure. The function $E(s_1)$, used in the procedure, is defined for a state $s_1 \in S_1$. $E(s_1)$ designates the equivalent class that s_1 initially belongs to in M_1. When M_1 is copied to M_1' at the beginning of the procedure, we assume that $E(s_1)$ is associated with each state s_1 of S_1'. Furthermore, $N'(\bar{s}_1, \Sigma, \mathbf{u}, s_1)$ is given by

$$N'(\bar{s}_1, \Sigma, \mathbf{u}, s_1) = \{(s_2, s^*) \in S_2 \times 2^{|S|} \mid \exists \mathbf{x} \in B^{|X|}, (\tilde{s}_2, \tilde{s^*}) \in \Sigma : \begin{array}{ll} \mathbf{u} = \lambda_2^{(u)}(\tilde{s}_2, \mathbf{xv}), & s_1 = \delta_1'(\bar{s}_1, \mathbf{u}), \\ s_2 = \delta_2(\tilde{s}_2, \mathbf{xv}), & s^* = \Delta(\tilde{s^*}, \mathbf{x}), \end{array} \},$$

where $\mathbf{v} = \lambda_1'(\bar{s}_1, \mathbf{u})$. When a new state $\hat{s}_1$ is created in M_1', we set $\delta_1'(\hat{s}_1, \tilde{\mathbf{u}}) \leftarrow \delta_1(s_1, \tilde{\mathbf{u}})$ for each $\tilde{\mathbf{u}} \in B^{|U|}$, where s_1 is the next state of $\bar{s}_1$ in M_1' under the input $\mathbf{u}$. Note that it is always true that the next state s_1 is a state which originally existed in M_1' when M_1 was duplicated. Therefore the state corresponding to s_1 also exists in M_1, which we denote also by s_1. Hence, by $\delta_1'(\hat{s}_1, \tilde{\mathbf{u}}) \leftarrow \delta_1(s_1, \tilde{\mathbf{u}})$, we mean $\delta_1'(\hat{s}_1, \tilde{\mathbf{u}})$ is set to the state of M_1' which corresponds to the state of M_1 given by $\delta_1(s_1, \tilde{\mathbf{u}})$. In other words, the transitions of a newly created state $\hat{s}_1$ are made identical with those defined at s_1 in M_1.

We first claim that the procedure maintains the invariance that a state s_1 of M_1' is equivalent to every state of $E(s_1)$ of M_1. Namely, for all sequences σ of $B^{|U|}$, the output sequence defined at s_1 by σ in M_1' is identical with that defined at a state of $E(s_1)$ in M_1. The invariance is trivially true in the beginning since the function E is so defined. Suppose that the invariance holds immediately before a state $\bar{s}_1$ is processed. Consider the case where $\bar{s}_1$ is processed for a minterm $\mathbf{u} \in B^{|U|}$. Suppose a new state $\hat{s}_1$ is created. Since the transitions defined at $\hat{s}_1$ in M_1' are identical with those of s_1 defined in M_1, $\hat{s}_1$ of M_1' is equivalent to s_1 of M_1. Since $E(\hat{s}_1)$ is set to $E(s_1)$, the invariance holds for the state $\hat{s}_1$. Also, if the state $\delta_1'(\bar{s}_1, \mathbf{u})$ is changed from s_1 to another already existing state $\hat{s}_1$, then $E(\hat{s}_1) = E(s_1)$ holds by construction. Since the output values of $\bar{s}_1$ do not change during the process, $\bar{s}_1$ obtained after the process for $\mathbf{u}$ is still equivalent to a state of $E(\bar{s}_1)$ of M_1. Thus the invariance holds. Therefore, M_1' obtained immediately after processing $\bar{s}_1$ is equivalent to M_1.

Secondly, the procedure terminates since every state is processed exactly once and a new state is created only if there is no state $\hat{s}_1$ equivalent to s_1 with $\Sigma(\hat{s}_1) = N$, while the number of states of M_1 and the number of subsets of $S_2 \times 2^{|S|}$ are both finite.

It follows that M_1' obtained at the end of the procedure is equivalent to M_1.

Finally, we claim that M_1' is a prime machine. The condition (2) shown in the definition of prime machines holds for M_1' since all the states of M_1' are processed and the set N used in the procedure is the one defined by $N(\bar{s}_1, \Sigma(\bar{s}_1), \mathbf{u}, s_1)$ in the definition, and we set $\Sigma(s_1)$ equal to N for each next state. For the condition (3), there are two classes of states s_1 for which there is no $t \geq 0$ such that $\Sigma(s_1, t) \neq \phi$; one

is those which are not reachable in M_1' and the other is those which are reachable in M_1' but not with the existence of M_2. For a state s_1 in the first class, $\Sigma(s_1)$ remains *undefined* until it is explicitly set to $\{\phi\}$ at the end of the procedure, and thus the condition holds. For a state s_1 of the second class, the condition holds if the conditions (1) and (2) hold, since in this case, the procedure sets N to $\{\phi\}$.

Hence, the proof is done if we prove the condition (1), i.e. for each $s_1 \in S_1'$, if $\Sigma(s_1, t) \neq \phi$, then $\Sigma(s_1, t) = \Sigma(s_1)$. We claim it by induction on $t \geq 0$. Consider the case where $t = 0$. Only the state s_1 such that $\Sigma(s_1, 0) \neq \phi$ is the reset state r_1. The procedure sets $\Sigma(r_1) = \{(r_2, \{r\})\}$, which is equal to $\Sigma(r_1, 0)$.

In the induction step, let s_1 be a state such that $\Sigma(s_1, t) \neq \phi$, where $t > 0$. Consider an arbitrary $\mathbf{u} \in B^{|U|}$ and $\bar{s}_1 \in S_1'$ such that $s_1 = \delta_1'(\bar{s}_1, \mathbf{u})$ in M_1'. We claim that if $\Sigma(\bar{s}_1, t-1)$ is not empty, then $N'(\bar{s}_1, \Sigma(\bar{s}_1, t-1), \mathbf{u}, s_1) = \Sigma(s_1)$. Note that the non-emptiness of $\Sigma(s_1, t)$ implies that there exists at least one such $\bar{s}_1$. It follows that $\Sigma(s_1, t) = \Sigma(s_1)$ since $\Sigma(s_1, t)$ is given by the union of $N'(\bar{s}_1, \Sigma(\bar{s}_1, t-1), \mathbf{u}, s_1)$ over all $\mathbf{u} \in B^{|U|}$ and all $\bar{s}_1 \in S_1'$ with $s_1 = \delta_1'(\bar{s}_1, \mathbf{u})$ and since if $\Sigma(\bar{s}_1, t-1)$ is empty, then $N'(\bar{s}_1, \Sigma(\bar{s}_1, t-1), \mathbf{u}, s_1)$ is also empty. By the induction hypothesis, $\Sigma(\bar{s}_1, t-1) = \Sigma(\bar{s}_1)$, and thus $N'(\bar{s}_1, \Sigma(\bar{s}_1, t-1), \mathbf{u}, s_1)$ is equal to N defined in the procedure for $\bar{s}_1$ and $\mathbf{u}$. Since at the end of the procedure, the existence of the transition $s_1 = \delta_1'(\bar{s}_1, \mathbf{u})$ implies that $\Sigma(s_1) = N$, $N'(\bar{s}_1, \Sigma(\bar{s}_1, t-1), \mathbf{u}, s_1) = \Sigma(s_1)$. This completes the proof for the condition (1). Hence M_1' is a prime machine. ■

Note that the procedure shown in Figure 13 is presented for proving the theorem above, and there is no need to use it for computing the E-machine.

```
function prime(M_1 = (U, V, S_1, λ_1, δ_1, r_1))
    /* let M_1' = (U, V, S_1', λ_1', δ_1', r_1) */
    M_1' ← copy(M_1);
    for(each s_1 ∈ S_1'){
        Σ(s_1) ← undefined;
    }
    Σ(r_1) ← {(r_2, {r})}; /* r_1 ∈ S_1' */
    mark r_1;
start:
    while(there exists s̄_1 ∈ S_1' that is marked){
        for(each u ∈ B^|U|){
            /* let v = λ_1'(s̄_1, u) and s_1 = δ_1'(s̄_1, u) */
            N ← N'(s̄_1, Σ(s̄_1), u, s_1);
            if(∃ŝ_1 ∈ S_1' : Σ(ŝ_1) = N and E(ŝ_1) = E(s_1)){
                δ_1'(s̄_1, u) ← ŝ_1;
            }
            else if(Σ(s_1) = undefined){
                Σ(s_1) ← N;
                mark s_1;
            }
            else{
                /* create a new state ŝ_1 */
                S_1' ← S_1' ∪ {ŝ_1};
                for(each ũ ∈ B^|U|){
                    δ_1'(ŝ_1, ũ) ← δ_1(s_1, ũ);
                    λ_1'(ŝ_1, ũ) ← λ_1(s_1, ũ);
                }
                δ_1'(s̄_1, u) ← ŝ_1;
                Σ(ŝ_1) ← N;
                E(ŝ_1) ← E(s_1);
                mark ŝ_1;
            }
        }
        remove the mark of s̄_1;
    }
    for(each s_1 ∈ S_1' such that Σ(s_1) = undefined){
        Σ(s_1) ← {φ}; mark s_1;
    }
    if(there is a marked state) goto start;
    return M_1';
```


Figure 13: Procedure to Generate a Prime Machine

References

[1] K. Bartlett, R. Brayton, G. Hachtel, R. Jacoby, C. Morrison, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang. Multi-level Logic Minimization using Implicit Don't Cares. *IEEE Transactions on Computer-Aided Design*, CAD-7, June 1988.

[2] R. K. Brayton, G. D. Hachtel, C. McMullen, and A. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers, Boston, 1984.

[3] R. K. Brayton and F. Somenzi. Boolean Relations and the Incomplete Specification of Logic Networks. In *International Conference on Very Large Scale Integration*, Munich, August 1989.

[4] R. E. Bryant. Graph Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, Vol. C-35(No. 8):677–691, August 1986.

[5] 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.

[6] S. Devadas. Approaches to Multi-Level Sequential Logic Synthesis. In *26th ACM/IEEE Design Automation Conference*, 1989.

[7] M. Fujita. Methods for Automatic Design Error Correction in Sequential Circuits. In *The European Conference on Design Automation with The European Event in ASIC Design*, February 1993.

[8] S. J. Hong, R. G. Cain, and D. L. Ostapko. MINI: A Heuristic Approach for Logic Minimization. *IBM J. Res. Develop.*, pages 443–458, September 1974.

[9] J. Kim and M. Newborn. The Simplification of Sequential Machines with Input Restrictions. *IEEE Transactions on Computers*, C-21:1440–1443, December 1972.

[10] E. J. McCluskey Jr. Minimization of Boolean Functions. *Bell System Technical Journal*, Vol. 35:1417–1444, November 1956.

[11] G. Mealy. A Method for Synthesizing Sequential Circuits. Technical Report J. 34, Bell System Tech., 1955.

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

[13] S. Muroga, Y. Kambayashi, C. H. Lai, and J. N. Culliney. The Transduction Method - Design of Logic Networks based on Permissible Functions. *IEEE Transactions of Computers*, 1989.

[14] P. Ramadge and W. Wonham. Supervisory Control of a Class of Discrete Event Processes. *SIAM Journal of Control and Optimization*, Vol. 25(No. 1):206–230, January 1987.

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

[16] H. Savoj and R. Brayton. Observability Relations and Observability Don't Cares. In *IEEE International Conference on Computer-Aided Design*, November 1991.

[17] Y. Watanabe and R. K. Brayton. Incremental Synthesis for Engineering Changes. In *IEEE International Conference on Computer Design*, October 1991.