Engineering Change in a Non-Deterministic FSM Setting

Sunil P. Khatri[1] Amit Narayan[1] Sriram C. Krishnan[1]
Kenneth L. McMillan[2] Robert. K. Brayton[1] A. Sangiovanni-Vincentelli[1]

Abstract

We propose a new formalism for the Engineering Change (EC) problem in a finite state machine (FSM) setting. Given an implementation that violates the specification, the problem is to alter the behavior of the implementation so that it meets the specification. The implementation can be a pseudo-nondeterministic FSM while the specification may be a nondeterministic FSM. The EC problem is cast as the existence of an "appropriate" simulation relation from the implementation into the specification. We derive the necessary and sufficient conditions for the existence of a solution to the problem. We synthesize all possible solutions, if the EC is feasible. Our algorithm works in space which is linear, and time which is quadratic, in the product of the sizes of implementation and specification. Previous formulations of the problem which admit nondeterministic specifications, although more general, lead to an algorithm which is exponential. We have implemented our procedure using Reduced Ordered Binary Decision Diagrams.

1 Introduction

The Engineering Change (henceforth EC) problem occurs frequently in integrated circuit design. One often encounters situations where the circuit implemented on silicon does not perform according to the specification. The designer would like to alter the functionality of a single die, on an experimental basis, and see if the altered circuit performs within the specification. If it does, the change is incorporated in the next mask revision. This capability significantly reduces the cost and time-to-market. An entire mask revision is not required to test the change.

There are regions of the layout that contain a variety of different uncommitted gates and latches. This uncommitted logic can be used to change the functionality of a circuit on an experimental basis by using a Focussed Ion Beam (FIB) apparatus. The FIB machine allows one to cut a wire on silicon, and also to deposit new wires over the passivation oxide. Designs can also be altered using programmable logic which is often available on-chip.

In the past EC has been used to alter the functionality of the combinational part of circuits [7, 9, 10]. Often it is not possible to rectify a design by changing only the combinational part of the circuit. In such cases the sequential behavior of the machine may be altered by adding/deleting latches, in addition to making changes in the combinational part. Sequential circuits are usually modeled as Finite State Machines (FSMs). In this context the EC problem can be stated as follows: Given an implementation FSM that does not conform to the specification FSM, the goal is to synthesize a controller FSM, which when composed with the implementation, generates output sequences (for any given input sequence) that are allowed by the specification.

This work has applications in various other practical scenarios also. In the context of a system of interacting machines, a certain component of the system may have to be replaced with another that has better characteristics such as area, delay, testability, etc. Our techniques can be employed to determine all possible replacements for the component. There are applications in the control systems area as well, where a controller for a plant has to be synthesized.

In this paper, we allow the specification to be nondeterministic and the plant to be pseudo-nondeterministic. Nondeterminism is very convenient for specifying properties (specification) and in modeling the environment for a design [13]. In theory, it is always possible to represent the behavior of a nondeterministic FSM by a deterministic FSM, but in practice, the best known construction for converting a nondeterministic machine into an equivalent deterministic machine is exponential in the worst case. Therefore the use of nondeterminism in specification allows convenient and compact modeling.

The central question in the EC problem is that of determining what machines, when composed with a component, can satisfy or "match" the specification. In this work, we provide a simple and clear formulation, and solution to this problem, using the formalism of *simulation relations* from concurrency theory[11]. We cast the EC problem as that of finding an implementable FSM that when composed with the implementation has a simulation relation into the nondeterministic specification. We derive the necessary and sufficient conditions for the existence of a feasible controller. In case the engineering change is feasible, we construct a nondeterministic FSM which contains all possible controllers and from which a feasible deterministic controller is easily synthesized. The entire procedure works in space linear and time quadratic in the product of the sizes of the implementation and the specification. In contrast, the previous works that admitted nondeterministic specifications [15, 2] essentially required a determinization, paying an exponential price in the worst case, thus losing the benefits of the compactness of nondeterminism and limiting the practical utility of their procedure. Our approach provides a comprehensive and simultaneous treatment of the practical issues relating to implementability, while other approaches dealt with it in an ex post facto manner.

Once the engineering change has been determined to be feasible and all possible solutions characterized, the next step is to synthesize the rectifying controller subject to the constraints of the available uncommitted logic. We plan to address this "constrained synthesis" problem in the future.

The rest of the paper is organized as follows. In Section 2, we define the terminology used in the sequel. We state the EC problem in Section 3, and present our approach in Section 4. We review related previous work and contrast it with our approach in Section 5. In Section 6 we present some preliminary experimental results, and conclude in Section 7.

2 Preliminaries and Definitions

We represent sets by upper case alphabets, and elements of sets by lower case letters. A lower case letter represents an element from the set denoted by the corresponding upper case letter. For example, v represents an element of the set V. Similarly, $\forall v$ and $\exists v$ are assumed to quantify over the set V. $|V|$ is the cardinality of set V.

Definition 1 *A* **Finite State Machine** *(FSM) M is a 5-tuple (I, O, S, R, r) where I is the input alphabet, O the output alphabet (both assumed to be finite), S a finite set of states, $R \subseteq S \times I \times S \times O$ the output and transition relation, and r the initial state. $R(s, i, s', o)$ means that for input i, there is a transition from state s to state s' producing output o. This is also denoted by $s \xrightarrow{i/o}_M s'$.*

If the output and the next state are uniquely defined for a given input and present state, the FSM is said to be a **deterministic FSM** (DFSM); a

[1]Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA 94720
[2]Cadence Berkeley Laboratories, Berkeley, CA

33rd Design Automation Conference ®

DAC 96 - 06/96 Las Vegas, NV, USA

FSM is said to be **pseudo-nondeterministic** (PNDFSM) if the next state is uniquely defined for a given present state, input, *and* output. A FSM is said to be **nondeterministic** (NDFSM) if there is some state, input, and output for which there is more than one next state. Examples are shown in Figures 1, 2, 3. If at least one next state and output is defined for each input and present

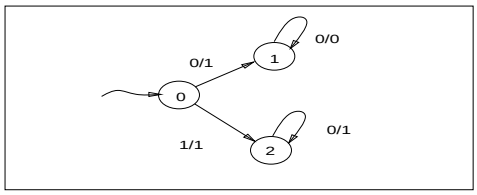

Figure 1: Deterministic FSM

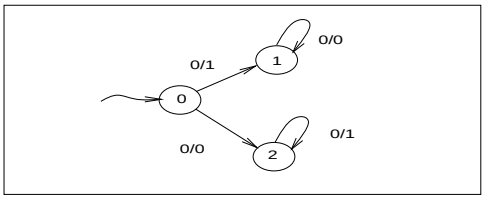

Figure 2: Pseudo Non-deterministic FSM

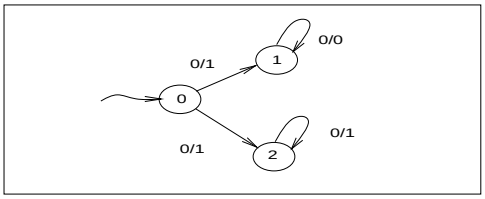

Figure 3: Non-deterministic FSM

state, we say that the FSM is completely specified. Otherwise, it is said to be incompletely specified.

A FSM can be interpreted as an automaton over the alphabet $I \times O$. The set of all pairs (I_k, O_k) such that sequence O_k is produced as output on applying sequence I_k as the input, gives the language of the automaton. Both deterministic and pseudo-nondeterministic FSMs are deterministic in the automaton sense (i.e., the underlying automaton for a pseudo-nondeterministic machine makes a unique transition for a given input-ouput pair), while a nondeterministic FSM is nondeterministic in the automaton sense.

The notion of a simulation relation between two machines was introduced by Park in [11].

Definition 2 *We say $\psi \subseteq S_1 \times S_2$ is a* **Simulation Relation** *(abbreviated SR) from a FSM M_1 to a FSM M_2 if*

1. $(r_1, r_2) \in \psi$, *and*
2. $(s_1, s_2) \in \psi \Rightarrow \{ \forall i \forall o \forall s_1^{'} [(s_1 \xrightarrow{i/o}_{M_1} s_1^{'}) \Rightarrow \exists s_2^{'} [(s_2 \xrightarrow{i/o}_{M_2} s_2^{'}) \wedge (s_1^{'}, s_2^{'}) \in \psi]\ ]\ \}$

In this case we say that M_2 simulates M_1 and denote it by $M_1 \preceq M_2$. Alternatively, we say that M_1 has a simulation into M_2.

Condition 2, says that s_2 in M_2 simulates s_1 in M_1, if each transition from s_1 is simulated by s_2, and the next states in M_1 in turn are simulated by the next states in M_2, and so on.

Definition 3 *Given FSMs $M_1(U, Y, S_1, R_1, r_1)$ and $M_2((V \times Y), U, S_2, R_2, r_2)$, the* **composition** *$\hat{M}(V, Y, \hat{S}, \hat{R}, \hat{r}) = M_1 \circ M_2$ satisfies the following properties:*

1. $\hat{S} = S_1 \times S_2$
2. $\hat{r} = (r_1, r_2)$
3. $\hat{R}(\ (s_1, s_2), v, (s_1^{'}, s_2^{'}), y\ )$ *iff* $\exists u [R_1(s_1, u, s_1^{'}, y) \wedge R_2(s_2, (v, y), s_2^{'}, u)]$

The composed machine makes a transition $((s_1, s_2) \xrightarrow{v/y}_{\hat{M}} (s_1^{'}, s_2^{'}))$ iff there exists a u s.t. $(s_1 \xrightarrow{u/y}_{M_1} s_1^{'})$, and $(s_2 \xrightarrow{(v,y)/u}_{M_2} s_2^{'})$.
The composition is said to be **well defined** provided a transition is allowed for every possible input v (i.e., the machine $\hat{M}$ is complete). Further, we say that this composition is **implementable** in the hardware sense provided no combinational loops occur in the composed machine. This can be ensured if either M_1 is $Moore$ (i.e., the output of M_1 is independent of the inputs), or M_2 is *Moore with respect to Y* (i.e., the output of M_2 is independent of Y).

3 Problem Statement

The EC problem is formally stated as follows:

Problem Statement 3.1 *Given FSMs $M_1(U, Y, S_1, R_1, r_1)$ and $M(V, Y, S, R, r)$, the* **EC** *problem is to find a FSM $M_2((V \times Y), U, S_2, R_2, r_2)$ such that the following conditions hold:*

1. *$M_1 \circ M_2$ is well-defined.*
2. *$M_1 \circ M_2$ is implementable.*
3. *For every reachable state (s_1, s_2) in $M_1 \circ M_2$, $\forall v \forall y$ if $(s_1, s_2) \xrightarrow{v/y}_{M_1 \circ M_2} (s_1^{'}, s_2^{'})$, then for each u s.t. $(s_2 \xrightarrow{(v,y)/u}_{M_2} s_2^{'})$, we have that $[\forall y^{'} \forall s_1^{''} (s_1 \xrightarrow{u/y^{'}}_{M_1} s_1^{''}) \Rightarrow \exists s_2^{''} ((s_1, s_2) \xrightarrow{v/y^{'}}_{M_1 \circ M_2} (s_1^{''}, s_2^{''}))]$.*
4. *$M_1 \circ M_2 \preceq M$.*

Condition 3, needed when M_1 is pseudo-nondeterministic or nondeterministic, says that any u M_2 supplies M_1 does not "block" M_1, i.e. M_1 is free to produce any legal output for that u and transition to any legal next state. It is stating that M_2 is controlling the behavior of M_1 just by providing appropriate u's. Once the controller has chosen a u for a v, M_1 is free to produce *any* output $y^{'}$ and go to any next state $s_1^{''}$. Condition 3 is not needed for a deterministic M_1, because for a given u the output y and next state $s_1^{'}$ of M_1 are unique. Our approach can admit a pseudo-nondeterministic M_1, and a nondeterministic M. The configuration of M_1 and M_2 is as in [6], and is shown in figure 4. Here M_2 can observe all the inputs to the system and change the inputs seen by M_1 (i.e. all the inputs to M_1 are *controllable*), but it can only observe the outputs of M_1. This configuration can very easily be generalized to the case when outputs are also controllable. The topology of Figure 4 is very natural and modular in the EC context, as the controller treats M_1 as a black box and looks only at the input-output behavior of the implementation.

Using the terminology of [6] we call M_1 the plant, M_2 the controller, and M the model.

4 Our Approach

The EC problem can be divided into three parts: 1) A *solvability* problem, where we have to check if a solution exists, 2) A *synthesis* problem, in which we have to synthesize a controller which when composed with the implementation results in a system that meets the specification, and 3) An *optimal* synthesis problem, in which a controller has to be chosen according to some optimality criterion (which could be area, delay, power, testability, etc.). We address the first two problems and provide efficient algorithms to solve them. We give necessary and sufficient conditions for the existence of a solution. If a solution exists, we find the "maximal controller", M_c, which

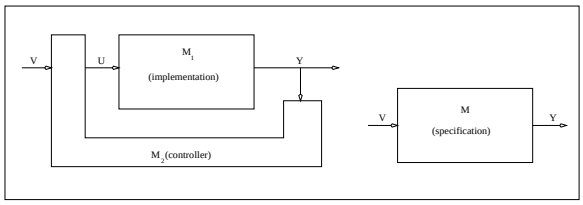

Figure 4: EC - Configuration of FSMs

contains all the solutions to the problem, and from which a feasible controller can be easily derived. The optimal synthesis problem of deriving a minimum state controller from M_c can be solved by using techniques of [8].

The relation $H_{max} \subseteq S_1 \times S$ given below, relates state s_1 in M_1 with state s in M if s "simulates" s_1.

Definition 4

$$(s_1, s) \in H_{max} \Leftrightarrow \quad \{\forall v \exists u \forall y \forall s_1^{'}[\ (s_1 \xrightarrow{u/y}_{M_1} s_1^{'}) \Rightarrow \exists s^{'}[(s \xrightarrow{v/y}_{M} s^{'}) \wedge (s_1^{'}, s^{'}) \in H_{max}]]\} \quad (1)$$

Intuitively, M_1 can be controlled if and only if for all v that the environment can produce, the controller can give the plant an input u such that both M_1 and M produce the same output and go to next states $s_1^{'}$ and $s^{'}$ respectively, and the same is true at $(s_1^{'}, s^{'})$. The following lemma states that *any* pair of states (s_1, s) such that s "simulates" s_1 and this continues successively, will be in H_{max}.

Lemma 4.1 *Let $\phi \subseteq S_1 \times S_2$ be a relation s.t. $(s_1, s) \in \phi \Rightarrow \{\forall v \exists u \forall y \forall s_1^{'}[(s_1 \xrightarrow{u/y}_{M_1} s_1^{'}) \Rightarrow \exists s^{'}[(s \xrightarrow{v/y}_{M} s^{'})$, and $(s_1^{'}, s^{'}) \in \phi]\ ]\}$. Then, $(s_1, s) \in \phi \Rightarrow (s_1, s) \in H_{max}$.*

Sketch of Proof: H_{max} can be computed as a greatest fixed point starting from $S_1 \times S$. Every $(s_1, s) \in \phi$ "survives" every stage of the fixed point computation. ∎

In the following sections we derive the necessary and sufficient conditions for the existence of a solution. We compute the maximal controller M_c and describe how we extract a feasible solution from it. We also discuss the complexity of our procedure.

4.1 Existence of a Solution

Theorem 4.2 *The EC problem has a solution as in Problem Statement 3.1 iff $(r_1, r) \in H_{max}$.*

Sketch of Proof: **Only if part:** Assume that there exists a FSM M_2 satisfying Conditions 1-4 of Problem Statement 3.1. We need to show that $(r_1, r) \in H_{max}$. Since $M_1 \circ M_2 \preceq M$ (Condition 4), there exists a SR $\psi \subseteq ((S_1 \times S_2) \times S)$, s.t. $((r_1, r_2), r) \in \psi$ and,

$$((s_1, s_2), s) \in \psi \Rightarrow \quad \{\ \forall v \forall y[((s_1, s_2) \xrightarrow{v/y}_{M_1 \circ M_2} (s_1^{'}, s_2^{'})) \Rightarrow \exists s^{'}[(s \xrightarrow{v/y}_{M} s^{'}) \wedge ((s_1^{'}, s_2^{'}), s^{'}) \in \psi]\ ]\ \} \quad (2)$$

Define a new relation $\phi \subseteq S_1 \times S$, where $\phi = \{(s_1, s) \mid \exists s_2$ s.t. $[(s_1, s_2), s) \in \psi]\}$.

Lemma 4.3

$$(s_1, s) \in \phi \Rightarrow \quad \{\ \forall v \exists u \forall y \forall s_1^{'}[\ (s_1 \xrightarrow{u/y}_{M_1} s_1^{'}) \Rightarrow \exists s^{'}[(s \xrightarrow{v/y}_{M} s^{'}) \wedge (s_1^{'}, s^{'}) \in \phi)\ ]\ ]\ \} \quad (3)$$

Sketch of Proof: Since $(s_1, s) \in \phi$, by the definition of ϕ, $\exists s_2$ s.t. $((s_1, s_2), s) \in \psi$. Since $M_1 \circ M_2$ is well-defined (Condition 1 of Problem Statement 3.1) for every v there exists a transition $(s_1, s_2) \xrightarrow{v/y}_{M_1 \circ M_2} (s_1^{''}, s_2^{''})$. By definition of composition it follows that there is a u s.t. $(s_1 \xrightarrow{u/y}_{M_1} s_1^{''})$. For this u, by Condition 3 of Problem Statement 3.1, $\forall y^{'} \forall s_1^{'}\ (s_1 \xrightarrow{u/y^{'}}_{M_1} s_1^{'})$, there exists $s_2^{'}$ s.t. $(s_1, s_2) \xrightarrow{v/y^{'}}_{M_1 \circ M_2} (s_1^{'}, s_2^{'})$. From Condition 4 (SR ψ), we know that there exists $s^{'}$ s.t. $(s \xrightarrow{v/y^{'}}_{M} s^{'})$ and $((s_1^{'}, s_2^{'}), s^{'}) \in \psi$. From the definition of ϕ, we have $(s_1^{'}, s^{'}) \in \phi$, and thus (3) is established. ∎

Since $((r_1, r_2), r) \in \psi$, by the definition of ϕ, $(r_1, r) \in \phi$. From Lemma 4.3 and Lemma 4.1, it follows that $(r_1, r) \in H_{max}$.

If part: Given that $(r_1, r) \in H_{max}$, we have to show that there exists a controller M_2 which satisfies conditions 1-4 in Problem Statement 3.1. We define a controller $M_2((V \times Y), U, S_2, R_2, r_2)$, where $S_2 \subseteq S_1 \times S$, and $S_2 = \{(s_1, s) \mid (s_1, s) \in H_{max}\}$, r_2 is (r_1, r), and the transition relation R_2 is defined as follows: for any state $(s_1, s) \in S_2$, we know that $(s_1, s) \in H_{max}$; therefore for any v there exists at least one u satisfying Equation 1; we pick one such u; now $\forall y \forall s_1^{'}$ s.t. $(s_1 \xrightarrow{u/y}_{M_1} s_1^{'})$, from Equation 1, there exists at least one $s^{'}$ s.t. $[(s \xrightarrow{v/y}_{M} s^{'}) \wedge (s_1^{'}, s^{'}) \in H_{max}]$; we choose one such $s^{'}$ resulting in transition $(s_1, s) \xrightarrow{(v,y)/u}_{M_2} (s_1^{'}, s^{'})$ in M_2. It is easy to see that M_2 satisfies Conditions 1-4 of Problem Statement 3.1. $M_1 \circ M_2$ is well-defined because we have picked a transition in M_2 for all v. $M_1 \circ M_2$ is implementable because the choice of u is independent of y, i.e. M_2 is Moore in Y. Conditions 3 is satisfied because we insert the transition $(s_1, s) \xrightarrow{(v,y)/u}_{M_2} (s_1^{'}, s^{'})$ for every y and every $s^{'}$ s.t. $(s_1 \xrightarrow{u/y}_{M_1} s_1^{'})$. Condition 4 is satisfied because the state $(s_1, (s_1, s))$ in $M_1 \circ M_2$ is simulated by state s in M. ∎

Note that $(r_1, r) \in H_{max}$ is a necessary condition for a solution to exist even if M_1 is nondeterministic (rather than pseudo-nondeterministic), but it can be shown that this is not sufficient.

4.2 Maximal Controller

In this section we address the synthesis problem; we construct a NDFSM M_c, called the maximal controller, such that any controller which is a solution to the problem will have a simulation into M_c, and any FSM which satisfies Conditions 1-3 in Problem Statement 3.1, and has a simulation into M_c will be a solution of the problem. If $(r_1, r) \in H_{max}$, then:

Definition 5 *$M_c((V \times Y), U, S_c, R_c, r_c)$ is defined as follows:*

- *$S_c \subseteq S_1 \times S$, and $S_c = \{(s_1, s) \mid (s_1, s) \in H_{max}\}$*
- *$r_c = (r_1, r)$*
- *$((s_1, s) \xrightarrow{(v,y)/u}_{M_c} (s_1^{'}, s^{'})) \Leftrightarrow [((s_1 \xrightarrow{u/y}_{M_1} s_1^{'}) \wedge (s \xrightarrow{v/y}_{M} s^{'}))$, and $(s_1^{'}, s^{'}) \in H_{max}]$*

There is a transition in the controller from (s_1, s) to $(s_1^{'}, s^{'})$ on (v, y) with an output u, if and only if there is a transition from s_1 to $s_1^{'}$ on u in M_1, and a transition from s to $s^{'}$ on v in M, both producing the same output y, and $(s_1^{'}, s^{'}) \in H_{max}$.

In the following theorems, we first show that M_c composed with M_1 has a simulation into M. Then, we claim the maximality of M_c by showing that any solution of the problem has a simulation into M_c; and any FSM that satisfies Conditions 1-3 of Problem Statement 3.1 *and* has a simulation into M_c, is a solution.

Theorem 4.4 *Let $M_1 \circ M_c = \hat{M}(V, Y, \hat{S}, \hat{R}, \hat{r})$. Then $\hat{M} \preceq M$.*

Sketch of Proof: We define a relation $\phi \subseteq (S_1 \times (S_1 \times S)) \times S$, where $\phi = \{((s_1,(s_1,s)),s) \mid (s_1,s) \in S_c\}$ between the states of $M_1 \circ M_c$ and M. It is easy to show that ϕ is a SR: since $r_c = (r_1,r)$, it follows that $((r_1,(r_1,r)),r) \in \phi$. Now assume that $(s_1,(s_1,s)),s) \in \phi$. If there is a transition $(s_1,(s_1,s)) \xrightarrow{v/y}_{M_1 \circ M_c} (s_1^{'},(s_1^{'},s^{'}))$, then $\exists u$ such that $[s_1 \xrightarrow{u/y}_{M_1} s_1^{'}]$ and $[(s_1,s) \xrightarrow{(v,y)/u}_{M_c} (s_1^{'},s^{'})]$. From the definition of M_c it follows that $(s_1 \xrightarrow{u/y}_{M_1} s_1^{'})$ and $(s \xrightarrow{v/y}_{M} s^{'})$, and $(s_1^{'},s^{'}) \in S_c$. Therefore, $((s_1^{'},(s_1^{'},s^{'})),s^{'}) \in \phi$. ∎

Theorem 4.5 *Let FSM M_2 satisfy conditions 1-3 of Problem Statement 3. Then $M_2 \preceq M_c \Leftrightarrow M_1 \circ M_2 \preceq M$.*

Proof: **($\Rightarrow$):** Since $M_2 \preceq M_c$, and $M_1 \circ M_c \preceq M$ (by Theorem 4.4), it follows that $M_1 \circ M_2 \preceq M$.
($\Leftarrow$): Given that $M_1 \circ M_2 \preceq M$, there exists a SR $\psi \subseteq (S_1 \times S_2) \times S$, relating the states of $M_1 \circ M_2$ and M. We define a new relation $\phi \subseteq S_2 \times (S_1 \times S)$ relating the states of M_2 with M_c, such that $(s_2,(s_1,s)) \in \phi$ iff $((s_1,s_2),s) \in \psi$. It is easy to show that ϕ is a SR from M_2 to M_c. ∎

4.3 Deriving an Implementation

The proof of the **if part** of the Theorem 4.2 basically is a recipe to pull out an arbitrary M_2 satisfying Conditions 1-4 of Problem Statement 3.1 from M_c.

We note that M_c derived in Section 4.2 is a nondeterministic automaton. In general, deciding if a feasible controller exists in a nondeterministic automaton proceeds by first determinizing the automaton (paying an exponential price). It is interesting to note that in our case, we can synthesize a feasible controller from M_c easily.

4.4 Computational Complexity of the procedure

In the construction of H_{max} we assume that initially all the states in the two machines are related. For a given pair (s_1, s), we leave it in H_{max} iff for every v, there exists a u, such that for every y, M_1 and M make transitions to $s_1^{'}$ and $s^{'}$ respectively, such that $(s_1^{'}, s^{'})$ is also in H_{max}. If this condition is not met we drop $(s_1^{'}, s^{'})$ from H_{max}. This process is iterated until a fixed point is obtained. In each iteration we check at most $O(|S| \cdot |S_1|)$ states, and at most $O(|S| \cdot |S_1|)$ iterations are needed (since at least one state is being dropped in each iteration). In each iteration, we do $O(|M_1 \circ M|)$ amount of work. Therefore the time complexity of the entire procedure is $O(|S| \cdot |S_1| \cdot |M_1 \circ M|)$, while the space complexity is $O(|S| \cdot |S_1|)$.

We should note however that our algorithms are implemented using Reduced Ordered Binary Decision Diagrams (ROBDDs) [4] and the complexity analysis given above is not valid for ROBDDs.

5 Comparison with the Previous Work

The EC problem was studied in the context of synthesizing interacting finite state machines by Watanabe and Brayton[15] and Aziz *et. al.*[2]. They address the problem of finding the "maximum set of permissible behaviors" for a component FSM in a system of interacting FSMs. [15, 2] consider the topology shown in Figure 5.

The topology shown in Figure 5 is more general than ours, as it allows inputs and outputs in M_1 which are not visible to M_2. This is of significance in a system of interacting FSMs because when synthesizing M_2, we do not want to disturb the interface which M_1 has with the environment. But in the EC scenario, we know that M_1 is not working correctly and M_2 is trying to alter the behavior of M_1. It is thus natural that the controller M_2 can observe all the outputs of M_1 and control all the inputs of M_1.

In [15], a pseudo-nondeterministic FSM, called the **E-Machine**, is derived which "captures" the maximal set of permissible behaviors. Each state of the E-Machine represents a subset of $S_1 \times 2^S$, where S_1 and S are the sets of states in M_1 and M respectively. Thus, the E-Machine in the worst case can have

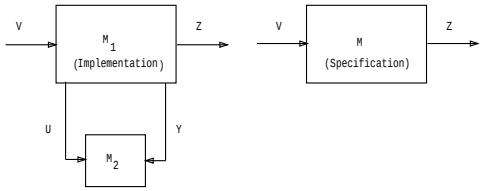

Figure 5: The E-Machine topology

$2^{|S_1| \cdot 2^{|S|}}$ states. When the specification is deterministic, each state of the E-Machine corresponds to a subset of $S_1 \times S$ and the worst case complexity is only singly exponential. The construction in [15] is very complicated.

Recently, Aziz *et al.* [2] have proposed the use of S1S to study the properties of sequential systems. S1S is the monadic second-order theory of one successor and is interpreted over the natural numbers with the less than ($<$) predicate and the successor ($+1$) function. Buchi, in his seminal work in the 1960s [5], showed that anything definable in S1S is regular, i.e. is representable by an automaton (known today as Buchi automata). He in fact calls S1S the Sequential Calculus due to its suitableness for describing the properties of sequential systems. In this framework, the language of a machine can be represented by a formula ϕ. If $\phi^{M_1}(V,U,Y,Z)$ and $\phi^{M}(V,Z)$ represent the languages of M_1 and M, then the E-Machine is given by $\neg((\exists V \exists Z \phi^{M_1} \wedge \neg \phi^{M})$ [2]. If M is nondeterministic complementing it is exponential [2]. Similarly, after quantifying V and Z, the resulting automaton can again be nondeterministic. Therefore, complementing it again, results in a total complexity of two exponentials. This complexity is inherent in the problem.

We are able to avoid the two exponentials by:

- observing that in the EC scenario all the inputs and outputs of M_1 are "available" to the controller.
- using the notion of simulation relations instead of complementing the nondeterministic specification in the first stage.

Intuitively, since the controller can see all the inputs and outputs, there is no need for quantification of these variables. Thus the resulting machine $(\phi^{M_1} \wedge \neg \phi^{M})$ in the second step is deterministic and can be easily complemented. Similarly, by using simulation relations, we avoid complementing the nondeterministic specification in the first step, and avoid an exponential construction. However, this gain does not come without a cost. Although we find all the controllers M_2 such that $M_1 \circ M_2 \preceq M$, there can be other "reasonable" controllers $M_2^{'}$, such that the language of $M_1 \circ M_2^{'}$ is contained in the language of M, but $M_1 \circ M_2^{'} \not\preceq M$. This is because for general nondeterministic specifications, a simulation pre-order is a more restrictive notion than language containment. However, for deterministic or pseudo nondeterministic specifications and implementations, language containment and simulation pre-orders are equivalent notions, and in these cases our scheme is complete. Therefore, given a nondeterministic specification, we can attempt to determinize it by the subset construction. This construction, though exponential in general, may be possible in many cases. If this step can be performed without an exponential blowup, our approach will give the same solution as the E-Machine. However, if this construction blows up, we can still attempt to find a controller under our notion of simulation pre-order while it will not be possible in the E-Machine approach.

Recently the same problem has been studied in the Control Systems domain as the problem of supervisory control of Discrete Event Dynamical Systems [1, 6]. The complexity of the procedures [1, 6] to determine the existence of a feasible solution is linear, the same as ours, but in their approach *both* the implementation and the specification are assumed to be deterministic. [1] cannot handle pseudo-nondeterministic plants. Our scheme is complete (in the language sense) for pseudo-nondeterministic specifications *and* plants. In addition, our scheme can handle nondeterministic specifications while maintaining its computational advantages.

In the previous approaches a distinction was made between a solution and

an implementable solution. A special construction was needed to derive implementable solutions from the set of all "solutions". Our approach provides a simultaneous treatment of implementability.

6 Implementation and Results

The approach presented in Section 4 has been implemented in the SIS [12] environment. The implementation assumes a NDFSM description for the specification and an incompletely specified DFSM description for the implementation.

Starting with an FSM description in the $kiss$ format, the program builds the transition relation [14], performs the fixed point computation for H_{max} and checks that $(r_1, r) \in H_{max}$, i.e. the problem is solvable. If so, then the transition relation for M_c is built. All computations are performed implicitly using ROBDDs [4, 3].

We present the results of our experiments in Table 1. Here, 'Controllable' represents that the implementation can be controlled to match the specification's behavior for all inputs and 'Not Controllable' represents that the implementation cannot match the specification's behavior. Experiments were performed on DECstation 5000/260 with 128MB of memory. We use the same examples as in [15] after modifying them for our topology. Figure 6 shows two example machines from Table 1.

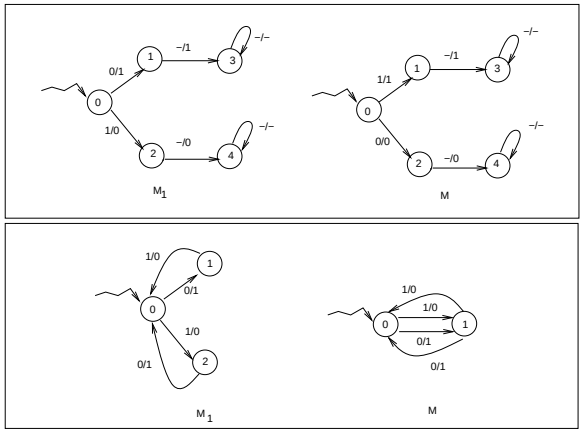

Figure 6: Examples oex4 and oex5 from table of results

7 Conclusions

We have addressed the problem of altering a pseudo-nondeterministic FSM implementation to conform to a possibly nondeterministic FSM specification. We cast the EC problem as that of finding an implementable FSM that when composed with the implementation has a simulation relation into the specification. We admit nondeterministic specifications without requiring determinization; other procedures, if they did admit nondeterminism at all, essentially determinize, often losing the benefits of the compactness of nondeterminism. In case the engineering change is feasible, we construct a nondeterministic FSM which contains all possible controllers. It is interesting to note that although the maximal controller is a nondeterministic automaton, we can easily decide if it contains a feasible controller and synthesize any possible controller from it. Our approach provides a comprehensive and simultaneous treatment of the practical issues relating to implementability, while other approaches dealt with it ex post facto.

In the future, we would like to extend our simulation relation approach to systems with fairness, and also timed and hybrid systems.

8 Acknowledgements

The first and the third authors were supported by California State Micro program grant #94-023. The second author was supported by CA State MICRO program grant #94-110 and the SRC 95-DC-324.

References

[1] A. Aziz, F. Balarin, R. K. Brayton, M. D. DiBenedetto, A. Saldanha, and A. L. Sangiovanni-Vincentelli. Supervisory Control of Finite State Machines. In *Computer Aided Verification 1995*, volume 939 of *LNCS*, pages 279–292. Springer-Verlag, 1995.

[2] A. Aziz, F. Balarin, R. K. Brayton, and A. Sangiovanni-Vincentelli. Sequential Synthesis Using S1S. In *Proc. of the Intl. Conf. on Computer-Aided Design*, pages 612–617, November 1995.

[3] K. S. Brace, R. L. Rudell, and R. E. Bryant. Efficient Implementation of a BDD Package. In *Proc. of the Design Automation Conf.*, pages 40–45, June 1990.

[4] R. Bryant. Graph-based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C-35:677–691, August 1986.

[5] J. R. Buchi. On a Decision Method in Restricted Second Order Arithmetic. *International Congress on Logic, Methodology, and Philosophy of Sciences*, pages 1–11, 1960.

[6] M. D. DiBenedetto, A. Saldanha, and A. Sangiovanni-Vincentelli. Model Matching for Finite State Machines. In *33rd IEEE Conference on Decision and Control*, volume 3, pages 3117–24, 1994.

[7] M. Fujita. Methods for automatic design error correction in sequential circuits. In *Proceedings of the European Conference on Design Automation with the European Event in ASIC Design*, February 1993.

[8] T. Kam, T. Villa, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. Implicit State Minimization of non-deterministic FSMs. In *Proc. of the Intl. Conf. on Computer Design*, pages 250–257, October 1995.

[9] Y. Kukimoto, M. Fujita, and R. K. Brayton. A Redesign Technique for Combinational Circuits based on Gate Reconnections. In *Proc. of the Intl. Conf. on Computer-Aided Design*, pages 632–637, November 1994.

[10] C-C Lin, K-C Chen, S-C Cheng, and M. Marek-Sadowska. Logic Synthesis for Engineering Change. In *Proc. of the Design Automation Conf.*, pages 647–652, June 1995.

[11] D. Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, *Proc. of the 5th GI conference*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer Verlag, 1981.

[12] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. SIS: A System for Sequential Circuit Synthesis. Technical Report UCB/ERL M92/41, Electronics Research Lab, Univ. of California, Berkeley, CA 94720, May 1992.

[13] S. Tasiran, R. Hojati, and R. K. Brayton. Language containment of non-deterministic ω-automata. In P. E. Camurati, editor, *Correct Hardware Design and Verification Methods. IFIP WG 10.5 Advanced Research Working Conference, CHARME '95*, pages 261–277, October 1995.

[14] H. Touati, H. Savoj, B. Lin, R. K. Brayton, and A. Sangiovanni-Vincentelli. Implicit State Enumeration of finite state machines using BDDs. In *Proc. of the Intl. Conf. on Computer-Aided Design*, November 1991.

[15] Y. Watanabe and R. K. Brayton. The Maximum Set of Permissible Behaviors for FSM Networks. In *Proc. of the Intl. Conf. on Computer-Aided Design*, 1993.

	Result	Implementation			Specification			M_c	Time
		Inputs	Outputs	States	Inputs	Outputs	States	States	(sec)
oex1	Controllable	1	1	2	1	1	2	2	0.004
oex2	Controllable	1	1	2	1	1	2	2	0.008
oex3	Not Controllable	1	1	3	1	1	3	-	0.011
oex4	Controllable	1	1	5	1	1	5	11	0.024
oex5	Not Controllable	1	1	3	1	1	2	-	0.004
oex6	Controllable	1	1	5	1	1	5	11	0.023
oex7	Controllable	1	1	3	1	1	3	7	0.008
oex9	Controllable	5	1	8	1	1	10	9	0.097
oex8	Controllable	5	1	20	2	1	21	14	1.988
ex7	Not Controllable	3	3	4	1	3	10	-	0.039
ex6	Not Controllable	6	2	13	3	2	16	-	0.235
ex5	Not Controllable	5	3	8	2	3	13	-	0.137
ex4	Not Controllable	5	3	20	2	3	12	-	0.266
ex14	Controllable	3	1	20	1	1	11	15	0.312
ex13	Controllable	3	1	22	1	1	7	12	0.140
ex12	Not Controllable	3	1	19	2	1	7	-	0.340
ex1	Not Controllable	2	2	20	1	2	8	-	0.113
ax9	Controllable	5	1	8	1	1	9	9	0.078
ax6	Not Controllable	6	2	13	3	2	7	-	0.070
ax4	Not Controllable	5	3	30	2	3	3	-	0.079
ex10	Not Controllable	3	3	19	1	3	3	-	0.055

Table 1: Implementation Results