An Engineering Change Methodology using Simulation Relations

Sunil P. Khatri[1] (linus@ic.eecs.berkeley.edu)

Amit Narayan[1] (anarayan@ic.eecs.berkeley.edu)

Sriram C. Krishnan[1] (krishnan@ic.eecs.berkeley.edu)

Kenneth L. McMillan[2] (mcmillan@cadence.com)

Alberto Sangiovanni-Vincentelli[1] (alberto@ic.eecs.berkeley.edu)

Robert K. Brayton[1] (brayton@ic.eecs.berkeley.edu)

[1]Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA 94720
tel: (510)-642-1462
fax: (510)-643-5052

[2]Cadence Berkeley Laboratories, Berkeley, CA

Abstract

In this paper, we address the problem of Engineering Change in a non-deterministic finite state machine framework. We are given a deterministic implementation FSM, and a non-deterministic specification FSM, such that the implementation does not meet the specification. The problem consists of controlling the implementation FSM in such a way that for all possible sequences of external inputs, the generated outputs are allowed in the specification.

We propose a new formalism for the Engineering Change problem which is applicable to non-deterministic specifications. We use the notion of *Simulation Relations* from the theory of concurrent systems, to develop this new formalism. Our method is cast in the form of a simulation of the implementation by the specification. We prove the necessary and sufficient condition for the existence of a solution to the problem. We also provide an algorithm to obtain all possible solutions under this setting. We have implemented this algorithm, using implicit state enumeration and Reduced Ordered Binary Decision Diagrams (ROBDDs).

An important feature of our method is that the algorithm gives us a solution which is correct by construction, and accordingly we do not need to perform a separate verification step in the design.

1 Introduction

We address the problem of **Engineering Change** (henceforth EC) [5] in a non-deterministic Finite State Machine (henceforth FSM) setting. We are given a specification, described as a FSM. Further, we are given an implementation, also a FSM, with a different set of inputs. The goal is to create a Controller FSM, which, when composed with the implementation, generates the same output sequence (for a given input sequence) as the specification. In other words, we desire that the composed machine has the same output language as the specification.

In integrated circuit design practice, one often encounters a situation where it is found that 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 according to the specification. If it does, the change is incorporated in the next mask revision. This capability significantly reduces the cost of EC, by reducing the number of EC iterations, and also reduces turnaround time between mask revisions.

In current day technology, this experimental change in the functionality of a single die is done by a *Focussed Ion Beam* (*FIB*) apparatus. The operator has the capability to view a section of the die under an electron microscope, identify the wire on the die that is to be cut, perform the cut, and deposit new wires on the die. These wires contact both the endpoints of the cut wire, and run over the passivation layer to an uncommitted gate. The wire is cut by bombarding the passivation layer and the wire with a focussed beam of ions. In the pit that is thus formed, new wires can be deposited and run to an uncommitted gate. There are regions of the layout that contain a variety of different uncommitted gates. In this way, the new gate is introduced in the circuit.

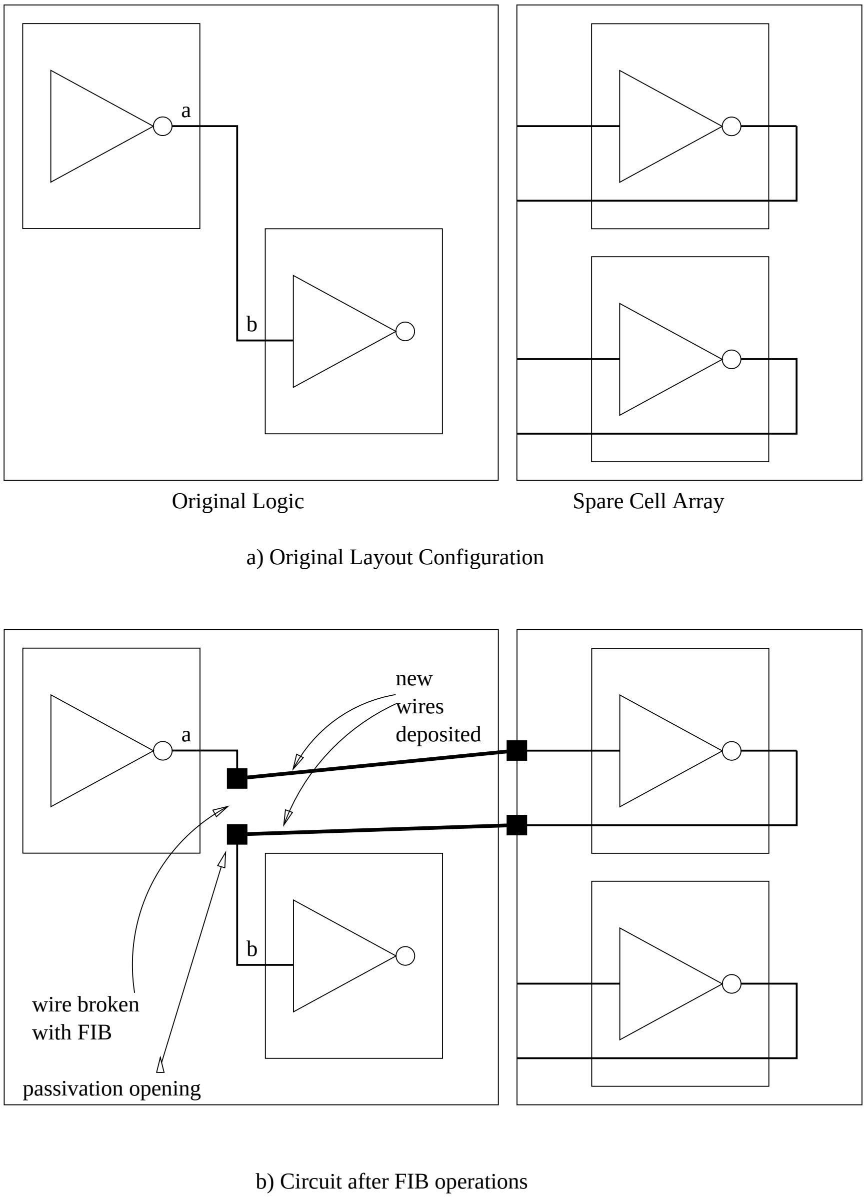

Figure 1: Changing circuit functionality with FIB operations

The FIB operations are illustrated in figure 1. In this figure, it was determined that an inversion was required between signals a and b. The wire between a and b is cut by a FIB, and new wires are deposited and run to a spare inverter, shown on the right side of the figure.

In practice, this process is mainly used to alter combinational circuit functionality. The changes are usually performed in an ad-hoc manual fashion. Recent research [6] addresses this problem, and an algorithm is provided which determines whether a given function is implementable using an array of gates. This array could be the gates of the implementation, along with the spare gates. For sequential circuits, however, we need to come up with a new FSM, such that the combined behavior of the old FSM and the new one meets the specification. This is too difficult to be done manually; it is for this reason that sequential circuits are not effectively handled in the engineering change scenario. Our paper provides an automated approach to engineering change in a sequential, non-deterministic setting.

In the engineering change scenario outlined above, it is important to keep the number of wires to be cut and re-routed to a minimum. The cost of a FIB cut is greater than that of opening up the passivation layer to form a contact. For this reason, the FSM configuration in figure 6 is a preferred one, since the outputs of the implementation are unaltered in this formalism. Accordingly, in our discussion, we focus on this configuration alone.

We prove the necessary and sufficient condition for the existence of a solution to the EC problem. Further, we present an algorithm to construct a maximal solution called the "controller", and prove that it contains all the feasible solutions. The advantage of our approach is that the maximal controller is verified by construction, so a separate design phase is not required.

Our problem formulation makes for easier understanding of the proofs and, we believe, is a good launching point for future research in the field.

2 Previous Work

Previous work in the EC for FSMs has been mainly in two directions. We briefly discuss these, and indicate how our approach is different.

The area of **Model Matching** within the area of **Discrete Event Dynamical Systems** (DEDS) has recently received attention in the Control Systems Engineering area. Here, the specification (called a Model) and the implementation (called a Plant) are given. A Controller FSM is to be synthesized, such that the composed behavior of the controller and the plant "matches" the behavior of the model in some specialized sense.

Recent developments in this area include the **Strong Model Matching** [4] (henceforth SMM) and

Generalized Strong Model Matching [3] (henceforth GSMM) works. In these works, the implementation and the specification are assumed deterministic, and the desired goal is output language equality. In SMM, the controller is free to alter the inputs from the environment to "trick" the specification into behaving appropriately. But, unlike GSMM, the controller cannot alter the outputs of the specification.

We propose a more general approach than SMM by relaxing the requirement of determinism on the specification. In addition, our approach can also handle a restricted notion of non-determinism (known as $pseudo\ non-determinism$), in the implementation FSM.

In another work, Watanabe and Brayton [13] addressed the problem of finding the "maximum set of permissible behaviors" for a component FSM in a system of interacting FSMs. They derive a non-deterministic FSM called the $E-machine$ which "captures" this maximal set of permissible behaviors.

We utilize the notion of a simulation relation [8] from concurrency theory to develop a simpler and more elegant treatment of the EC problem, and find a "correcting" FSM which minimally restricts the behavior of the implementation when composed with it. In this way, we derive an alternate characterization of the "maximal set of permissible behaviors"; we show that there exists a simulation relation from any admissible FSM into the non-deterministic FSM representing the set of all permissible behaviors.

3 Preliminaries and Definitions

We represent a set by an upper case English letter; an element of a set is represented by a lower case English letter.

Definition 1 *A* **Finite State Machine** *(FSM) is a 5-tuple M(I, O, S, R, r) where I is the input alphabet, O the output alphabet, S a finite set of states, $R \subseteq S \times S \times I \times O$ the output and transition relation, and r the initial state. In some cases, the relation R is alternately represented as a output relation $\gamma \subseteq S \times I \times O$ and a state transition relation $\lambda \subseteq S \times S \times I$.*

If the output and the next state are uniquely defined for a given input and present state, then the FSM is called a **Deterministic Finite State Machine** (DFSM), else it is called a **Non-deterministic Finite State Machine** (NDFSM). Examples are shown in figures 2 and 3.

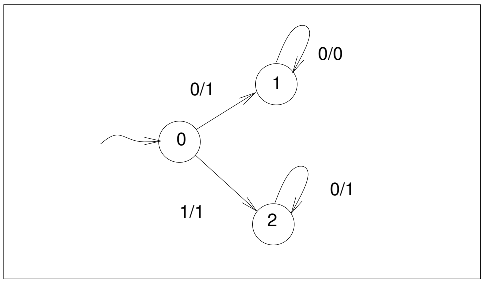

Figure 2: Deterministic FSM

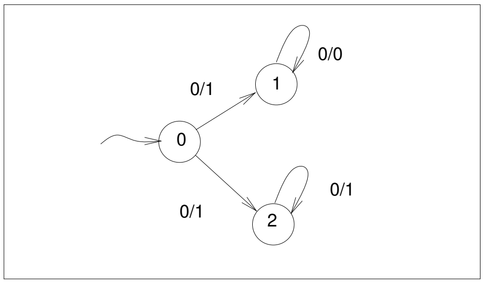

Figure 3: Non-deterministic FSM

If the next state is uniquely defined for a given present state, input and output, then the FSM is called **Pseudo Non-deterministic**. An example is shown in figure 4.

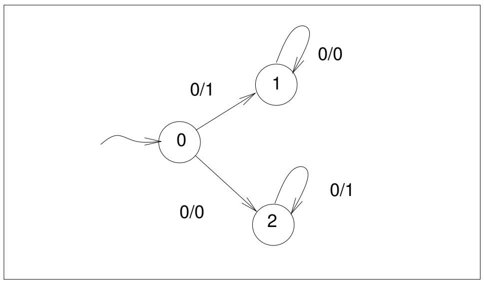

Figure 4: Pseudo Non-deterministic FSM

If the relations γ and λ are total, then M is said to be **completely specified**, else it is **incompletely specified**. If an input $i \in I$ is not applicable at state s, then $\lambda(s,i)$ is empty. In such a situation, we assign $\lambda(s,i) = \theta$, where θ is the **dead state**.

Definition 2 *There exists a* **Simulation Relation** *(abbreviated SR) from a FSM M_1 to a FSM M_2 if there exists a relation $\psi \subseteq S_1 \times S_2$ such that*

1. $(r_1, r_2) \in \psi$

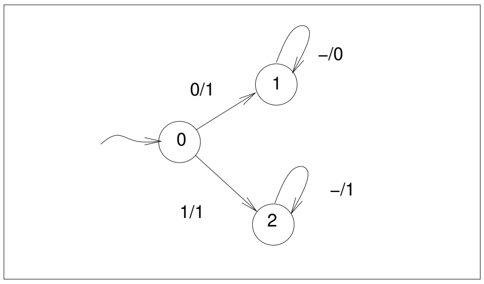

Figure 5: Example FSM

2. $(s_1, s_2) \in \psi \Rightarrow \forall i, \forall o, \forall s_1^{'}[(R_1(s_1, s_1^{'}, i, o) = 1) \Rightarrow \exists s_2^{'}((R_2(s_2, s_2^{'}, i, o) = 1) \wedge (\psi(s_1^{'}, s_2^{'}) = 1))]$

We denote the above condition $M_1 \preceq M_2$. We alternately denote $R_1(s_1, s_1^{'}, i, o) = 1$ by $s_1 \xrightarrow{i/o} s_1^{'}$. This means that there is a transition from state s_1 to state $s_1^{'}$ under an input i which produces output o.

Definition 3 *Given FSMs $M_1(U, Y, S_1, R_1, r_1)$ and $M_2((V, Y), U, S_2, R_2, r_2)$, 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. $\hat{R}(\hat{s}, \hat{s}^{'}, v, y) = 1$ *iff* $\exists u[[R_1(s_1, s_1^{'}, u, y) = 1] \wedge [R_2(s_2, s_2^{'}, (v, y), u)] = 1]$

The composed machine $\hat{S}$ makes a transition $\hat{s} \xrightarrow{v/y} \hat{s}^{'}$ iff $\exists u[(s_1 \xrightarrow{u/y} s_1^{'})$ in $M_1 \wedge s_2 \xrightarrow{(v,y)/u} s_2^{'}]$

4 Problem Statement

The EC problem is stated formally as follows:

Problem Statement 4.1 *Given FSMs $M_1(U, Y, S_1, R_1, r_1)$ and $M(V, Y, S, R, r)$, the specification matching problem of* **EC** *consists of finding a FSM $M_2((V, Y), U, S_2, R_2, r_2)$ such that $M_1 \circ M_2 \preceq M$.*

For our purpose, we assume that M_1 is deterministic, and M_2 is non-deterministic. The configuration of M_1 and M_2 is as in [4], and is shown in figure 6. We borrow the terminolgy of [4] and call M_1 as plant, M_2 as the controller and M as the model.

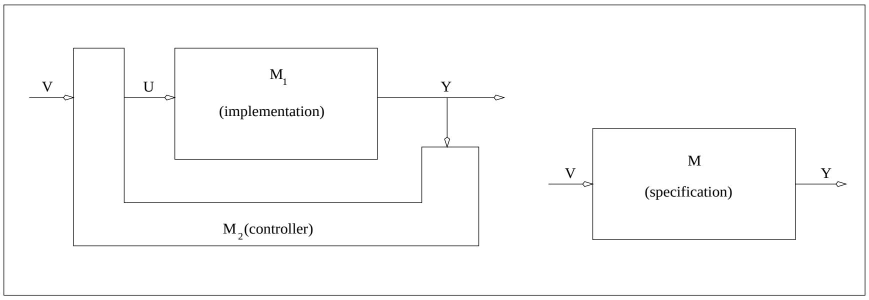

Figure 6: EC - Configuration of FSMs

5 Our Approach

In this paper, we assume that M is non-deterministic, and that M_1 is deterministic. The assumption that M_1 is deterministic is reasonable, since in most real problems, the implementation is a deterministic one. However, M can be non-deterministic in real problems. This corresponds to the designer of the system providing multiple acceptable specification behaviors. The designer would like the implementation to match any acceptable specification behavior.

We would like to do output language equality if possible, otherwise we perform output language containment. In a sense, we are trying to match as much of the specification's behavior as possible.

Our formulation of the problem is a Simulation Relation from the implementation into the specification. We define a relation H_{max}, which extracts the "maximal" behavior of the specification that can be matched by the implementation. Also, it "removes" behaviors of the implementation that are not contained in the specification.

Definition 4 $(s_1, s) \in H_{max} \Leftrightarrow \forall v \exists u \forall y \forall s_1^{'} [(s_1 \xrightarrow{u/y} s_1^{'}) \Rightarrow \exists s^{'} [(s \xrightarrow{v/y} s^{'}) \wedge (s_1^{'}, s^{'}) \in H_{max}]]$

Intuitively, s_1 can simulate s 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 the next state of M_1 can simulate the next state of M.

Intuitively, the above definition is like a simulation relation with different labels on the transitions. The above definition suggests a symbolic greatest fixed point implementation [7] of H_{max}.

In the following theorem we provide the necessary and sufficient condition for the solution of the problem to exist. We prove that the problem of EC is solvable if and only if the initial states r_1 and r of M_1 and M

respectively are related in H_{max}.

Theorem 5.1 *The specification matching problem has a solution iff* $(r_1, r) \in H_{max}$

Proof: **Only if part:** Given that $(r_1, r) \in H_{max}$. We have to show that for any arbitrary k and for all sequences $v_0 v_1 .... v_k$ there exists a sequence $u_0 u_1 .... u_k$ such that the outputs $y_0 y_1 ... y_k$ produced by both implementation and the specification are the same. We prove this by induction on the length of the sequence.

- length = 1: By the definition of H_{max}, $\forall v_0\ \exists u_0$ s.t. if $r_1 \stackrel{u_0/y_0}{\longrightarrow} s_1$ in M_1, then $r \stackrel{v_0/y_0}{\longrightarrow} s$ in M, and $(s_1, s) \in H_{max}$

- Induction Hypothesis: Assume, for any sequence $v_0 v_1 v_2 ... v_k\ \exists u_0 u_1 .... u_k$ s.t. if $r_1 \stackrel{u_0/y_0}{\longrightarrow} s_1 \stackrel{u_1/y_1}{\longrightarrow} s_1^1 \stackrel{u_2/y_2}{\longrightarrow}$ .... $\stackrel{u_k/y_k}{\longrightarrow} s_1^k$, in M_1 then, $r \stackrel{v_0/y_0}{\longrightarrow} s \stackrel{v_1/y_1}{\longrightarrow} s^1 \stackrel{v_2/y_2}{\longrightarrow}$ .... $\stackrel{v_k/y_k}{\longrightarrow} s^k$ and $(s_1^k, s^k) \in H_{max}$ in M

- length = k+1: By definition of H_{max}, for an arbitrary $v_{k+1}\ \exists u_{k+1}$ s.t. if $s_1^k \stackrel{u_{k+1}/y_{k+1}}{\longrightarrow} s_1^{k+1}$, then $s^k \stackrel{v_{k+1}/y_{k+1}}{\longrightarrow} s^{k+1}$, and $(s_1^{k+1}, s^{k+1}) \in H_{max}$

Therefore, the claim follows by induction on the length of the sequence.

If part: Since the specification matching problem is solvable, we know that $\exists M_2$ such that $M_1 \circ M_2 \preceq M$. We need to show that $(r_1, r) \in H_{max}$.

From the definition of SR, we know that there exists a relation φ relating state (r_1, r_2) of $M_1 \circ M_2$ with state r of M, such that

- $((r_1, r_2), r) \in \varphi$

- $((s_1, s_2), s) \in \varphi \Rightarrow \forall v \forall y$ if $[(s_1, s_2) \stackrel{v/y}{\longrightarrow} (s_1^{'}, s_2^{'})]$ then $\exists s^{'}$ s.t. $[s \stackrel{v/y}{\longrightarrow} s^{'}]$.

By definition of composition, if $[(s_1, s_2) \stackrel{v/y}{\longrightarrow} (s_1^{'}, s_2^{'})]$ in $M_1 \circ M_2$ then $\exists u$ such that $s_1 \stackrel{u/y}{\longrightarrow} s_1^{'}$ in M_1. Now we define a new relation ϕ relating these states of M and M_1 such that

- $(r, r_1) \in \phi$

- $(s, s_1) \in \phi \Rightarrow (s^{'}, s_1^{'}) \in \phi$

Hence for any infinite sequence $v_0 v_1 v_2 ....$, $\exists u_0 u_1 u_2 ...$ s.t. if $r_1 \stackrel{u_0/y_0}{\longrightarrow} s_1 \stackrel{u_1/y_1}{\longrightarrow} s_1^1 \stackrel{u_2/y_2}{\longrightarrow}$ .... in M_1, then $r \stackrel{v_0/y_0}{\longrightarrow} s \stackrel{v_1/y_1}{\longrightarrow} s^1 \stackrel{v_2/y_2}{\longrightarrow}$ .... in M, Therefore, by definition of H_{max}, we have $(r_1, r) \in H_{max}$.

■

5.1 Deriving the Maximal Controller

In this section we characterize all the possible solutions to the engineering change problem by an NDFSM M_c called the controller. This controller is maximal in the sense that any FSM which is a solution to the problem will have a simulation into M_c and any FSM which has a simulation into M_c will be a solution of the problem. The maximal controller $M_c((V, Y), U, S_c, R_c, r_c)$ is constructed as follows:

- $S_c \subseteq S_1 \times S = \{(s_1, s) \mid (s_1, s) \in H_{max}\}$
- $r_c = (r_1, r)$
- $R_c((s_1, s), (s_1^{'}, s^{'}), (v, y), u) = 1$ iff $[(R_1(s_1, s_1^{'}, u, y) = 1) \wedge (R(s, s^{'}, v, y) = 1)]$

The states of the controller are those in the cross product of the set of states of the plant and the specification which are present 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

In the following theorems, we claim the maximality of M_c by

- showing that M_c composed with M has a simulation into M,
- showing that any solution of the problem has a simulation into M_c, and
- any FSM that has a simulation into M_c is a solution.

Theorem 5.2 *Let $M_1 \circ M_c = \hat{M}(V, Y, \hat{S}, \hat{R}, \hat{r})$. Then $\hat{M} \preceq M$.*

Proof: By the definition of composition, for $\hat{s} = (s_1, (\tilde{s}_1, s))$ and $\hat{s}^{'} = (s_1^{'}, (\tilde{s}_1^{'}, s^{'}))$, if $[\hat{s}, \xrightarrow{v/y} \hat{s}^{'}]$, then $\exists u$ such that $[s_1 \xrightarrow{u/y} s_1^{'}]$ in M_1 and $[(\tilde{s}_1, s) \xrightarrow{(v,y)/u} (\tilde{s}_1^{'}, s^{'})]$ in M_c.

Similarly, if $(\tilde{s}_1, s) \xrightarrow{(v,y)/u} (\tilde{s}_1^{'}, s^{'})$ in M_c then, by definition of R_c

$[\tilde{s}_1 \xrightarrow{u/y} \tilde{s}_1^{'}]$ in M_1 and $[s \xrightarrow{v/y} s^{'}]$ in M.

Now, we define a relation ϕ such that for all $\hat{s} = (s_1, (\tilde{s}_1, s)), (\hat{s}, s) \in \phi$.

Clearly,

- $(\hat{r}, r) \in \phi$, since $\hat{r} = (r_1, (r_1, r))$, and
- $(\hat{s}, s) \in \phi \Rightarrow [\forall v \forall y \forall (\hat{s}^{'} = (s_1^{'}, (\tilde{s}_1^{'}, s^{'})))\hat{s} \xrightarrow{v/y} \hat{s}^{'}]$ in $M_c \Rightarrow \exists s^{'}[s \xrightarrow{v/y} s^{'}]$ in M and $(\hat{s}^{'}, s^{'}) \in \phi]$

Accordingly, there is a simulation from $\hat{M}$ to M. ■

Theorem 5.3 $M_2 \preceq M_c \Leftrightarrow M_1 \circ M_2 \preceq M$.

Proof: **Only if part:** Since $M_2 \preceq M_c$, and $M_1 \circ M_c \preceq M$ (by theorem 5.2), it follows that $M_1 \circ M_2 \preceq M$.

If part: Given that $M_1 \circ M_2 \preceq M$, there exists a relation φ such that

- $((r_1, r_2), r) \in \varphi$
- $((s_1, s_2), s) \in \varphi \Rightarrow [\forall v, \forall y, \forall (s_1^{'}, s_2^{'})$ if $[(s_1, s_2) \xrightarrow{v/y} (s_1^{'}, s_2^{'})]$ in $M_1 \circ M_2 \Rightarrow \exists s^{'}[s \xrightarrow{v/y} s^{'}]$ in M and $((s_1^{'}, s_2^{'}), s^{'}) \in \varphi]$

Also, $[(s_1, s_2) \xrightarrow{v/y} (s_1^{'}, s_2^{'})]$ in $M_1 \circ M_2$ implies, from the definition of composition,

$\exists u[s_2 \xrightarrow{(v,y)/u} s_2^{'}]$ in M_2 and $[s_1 \xrightarrow{u/y} s_1^{'}]$

Since $(r_1, r) \in H_{max}$ and M_1 is deterministic, by definition of H_{max}; $s_c = (s_1, s) \in H_{max}$ and $s_c^{'} = (s_1^{'}, s^{'}) \in H_{max}$.

Also, since $[s_1 \xrightarrow{u/y} s_1^{'}]$ in M_1 and $[s \xrightarrow{v/y} s^{'}]$ in M, then by definition of R_c, $[(s_1, s) \xrightarrow{(v,y)/u} (s_1^{'}, s^{'})$.

So, define a relation τ such that

- $(r_2, r_c) \in \tau$
- $(s_2, s_c) \in \tau \Rightarrow (s_2^{'}, s_c^{'}) \in \tau$ where $s_2^{'}, s_c^{'}$ denote the same states that are referred before.

Accordingly, $M_2 \preceq M_c$. ■

6 Implementation and Results

The approach presented in the section 5 has been implemented (about 1500 lines of C) in the SIS [10] environment. The implementation assumes a fully specified NDFSM description for the specification and an incompletely specified DFSM description for the implementation.

About 1500 lines of C code has been written. Starting with an FSM description in the *kiss* format, the program builds the transition relation [12], 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 done implicitly using ROBDDs [2] [1].

We present the results of our experiments below. In Table 1, '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. Figure 7 shows two example machines from Table 1.

7 Conclusions

We have addressed the problem of EC of FSM's in a non-deterministic setting. This problem consists of synthesizing a finite state controller for a implementation such that the closed loop behavior of the controller composed with the implementation can be simulated in the specification. Both implementation and specification are represented as FSM's. Our contributions are as follows:

- A new formalism for the EC problem was proposed. Our formalism handles both deterministic and non-deterministic specifications.
- The problem is cast as a simulation of the implementation into the specification.

	Result	Implementation			Specification			Controller	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

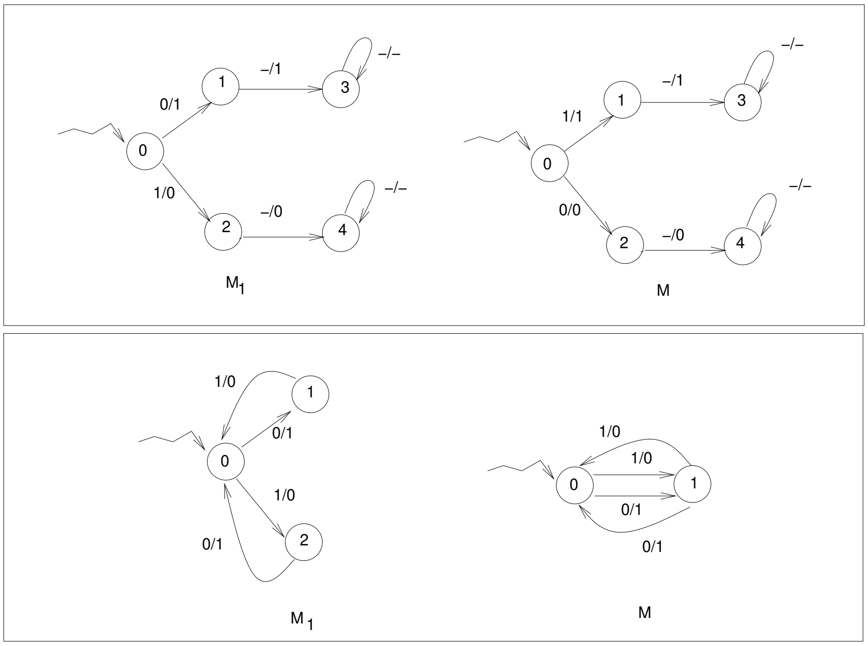

Figure 7: Examples oex4 and oex5 from table of results

- A necessary and sufficient condition for the solvabilty of the problem is given.
- A procedure to construct the 'maximal' controller M_c was presented.
- We prove that any FSM which has a simulation into M_c is a solution to the problem and all the solutions to the problem are contained in M_c.
- It can be shown easily that the strong model matching problem [4] is a special case of this formulation.
- The entire code has been implemented in SIS. All computations are done implicitly using BDDs for greater efficiency.

8 Future Work

There are some definite directions that remain to be explored as a sequel to this work. We would like to use our formulation to address the problem of stochastic languages [9]. Currently, in our work, we do not consider the infinitary properties of the FSMs. This work has received some interesting treatment in [11], and warrants study in our framework.

We see many opportunities to improve the code that has been written so far. For example, we could explore the use of partitioned transition relations. Various heuristics for ROBDD variable ordering are available; we need to find out the most suitable one in the context of specification matching. Other techniques to improve efficiency, like early quantification of variables and the use of different encoding techniques for FSMs have to be fully investigated. The code needs to be interfaced with an FSM determinization program to select the minimum state controller contained in M_c.

References

[1] 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.

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

[3] M. D. DiBenedetto, A. Saldanha, and A. Sangiovanni-Vincentelli. Model Matching for Finite State Machines. 1994. To Appear.

[4] M. D. DiBenedetto, A. Saldanha, and A. Sangiovanni-Vincentelli. Strong Model Matching for Finite State Machines. 1994. To Appear.

[5] 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.

[6] 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.

[7] Kenneth L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.

[8] R. Milner. *Communication and Concurrency*. Prentice Hall, New York, 1989.

[9] H. Mortazavian. Controlled Stachastic Languages. In *Proceedings of the 31st Annual Allerton Conference on Communication, Control, and Computing*, September 1993.

[10] 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.

[11] J. Thistle and W. Wonham. Infinite Behavior of Discrete-Event Systems. In *SIAM Journal of Control and Optimization*, pages 1099–1113, January 1987.

[12] H. Touati, J. Savoj, B. Lin, R. Brayton, and A Sangiovanni-Vincentelli. Implicit state enumeration of finite state machines using bdds. *International Conference on Computer Aided Design*, 1991.

[13] 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.