SYNTHESIZING INTERACTING FINITE STATE MACHINES

by

Adnan Aziz and Robert K. Brayton

Memorandum No. UCB/ERL M94/96

9 December 1994

SYNTHESIZING INTERACTING FINITE STATE MACHINES

by

Adnan Aziz and Robert K. Brayton

Memorandum No. UCB/ERL M94/96

9 December 1994

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

SYNTHESIZING INTERACTING FINITE STATE MACHINES

by

Adnan Aziz and Robert K. Brayton

Memorandum No. UCB/ERL M94/96

9 December 1994

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Synthesizing Interacting Finite State Machines

Adnan Aziz* Robert K. Brayton
Email:{adnan,brayton}@ic.eecs.berkeley.edu
Fax: 1 (510) 643 5052
VLSI CAD Group
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley, CA 94720, USA

Abstract

We use the logic S1S as the foundation upon which to base the theoretical analysis of issues related to the synthesis of interacting finite state systems. Using S1S we derive simple and rigorous proofs of existing synthesis algorithms, and fill in several gaps left open. We also use S1S to extend existing results to non-deterministic systems with fairness. We merge these results with classical work related to the Church solveability problem and apply it to discrete control.

Keywords: Finite state machine, Regular languages, Fairness, Sequential synthesis, Discrete control, S1S, Church solveability, Tree automata

*Supported by SRC Grant 94-DC-008

1 Introduction

The advent of modern VLSI CAD tools has radically changed the process of designing digital systems. The first CAD tools automated the final stages of design, such as placement and routing. As the low level steps became better understood, the focus shifted to the higher stages. In particular logic synthesis, the science of optimizing designs (for various measure such as area, speed, or power) specified at the gate level, has shifted to the forefront of CAD research. Another area rapidly gaining importance is design verification, the study of systematic methods for formally proving the correctness of designs.

Logic synthesis algorithms originally targeted the optimization of PLA implementations; this was followed by research in synthesizing more general multi-level logic implementations. Currently, the central thrust in logic synthesis is the automatic optimization of the entire system, including both the sequential elements (latches) in addition to combinational elements (gates).

Automated approaches to formal verification typically proceed by representing the design as a state machine and then traversing the state graph. The input system to formal verification tools is usually non-deterministic and has fairness constraints; this could be because the original design was too complex and had to be replaced by a simpler abstraction, or because at early stages of the design process components are left under specified.

Designs invariably consist of a set of interacting components. Natural questions related to such systems are what is the optimal choice of a component, and automatically deriving a component so as to satisfy given properties. In this paper we answer these questions using as our basic tool the sequential calculus S1S.

Previous work related to optimizing interacting state machines has tended to be ad hoc and incomplete. Relevant papers include [18, 7, 12, 20, 21, 22]. One attempt at formal synthesis framework based on trace and automata theory is given in [8]. The central theorem relating flexibility in a sub-circuit to the specification and the environment is incorrect; we give the correct formulation. There is a large body of theoretical work related to the existence of efficient decision procedures for deciding logics of programs, eg [23, 14, 6, 16]. In particular, [23] observes that the set of acceptable moves for a controller is an ω-regular set.

The rest of this paper is structured as follows: § 2 reviews the basic definitions and salient results. In § 3 we apply these results to systems operating on inputs and outputs bounded over time; in section § 4 we turn our attention to systems with fairness operating on infinite inputs. We finish by drawing conclusions, and suggesting extending this work to timed [1] and stochastic

systems [3].

2 Definitions and Basic Results

Definition 1 Given a finite set Σ, the set Σ^* is the set of all finite sequences over Σ, i.e. maps $f : n \to \Sigma$, $n \in \omega$. A $*$-language over Σ is a subset of Σ^*. Given $x \in \Sigma^*$, $| x |$ denotes the cardinality of x. The set Σ^ω is the set of all infinite-sequences over Σ, i.e. maps $f : \omega \to \Sigma$. An ω-language over Σ is a subset of Σ^ω.

Definition 2 A finite state automaton (FA) is a 5-tuple (Σ, S, s_0, T, A) where Σ is a finite set called the *alphabet*, S is a finite set of *states*, $s_0 \in S$ is the initial state, $T \subset S \times \Sigma \times S$ is the *transition relation*, and $A \subset S$ is the set of *accepting* states.

The automaton is said to be *deterministic* if $(\forall s)(\forall x)\,[\,|\,\{t : T(s,x,t)\}\,| \leq 1]$; otherwise it is said to be *non-deterministic.*

A string $x \in \Sigma^*$ is *accepted* by the FA if there exists a sequence of states $\sigma_0\sigma_1 \ldots \sigma_n$ such that $n = | x | +1$, $\sigma_0 = s_0$, $\sigma_n \in A$, and $(\forall i)\,[T(\sigma_i, x_i, \sigma_{i+1})]$. The language of the automaton is the set of strings accepted by it; this language is said to be $*$-regular if it is the language accepted by some automaton.

The following facts are proved in [11].

- The class of $*$-regular languages is closed under union, intersection, and projection; this is shown via construction.
- For any $*$-regular language $\mathcal{L}$, there is a deterministic FA accepting $\mathcal{L}$. Thus NFA and DFA have the same *expressive* power. Also, since DFA are immediately seen to be closed under complement, the class of $*$-regular languages is closed under complement.
- Deciding universality (are all strings accepted) for NFA is PSPACE complete; deciding universality for DFA is in P. Thus the *complexity* of NFA is higher than that of DFA.
- There is a sequence of languages $L_1, L_2, \ldots$ such that every DFA D_n accepting L_n has at least 2^{n-1} states, whereas there are NFA N_n accepting L_n with only n states. Thus NFA can be exponentially more *succinct* than DFA.
- Finding a minimum state NFA equivalent to a given NFA is PSPACE complete; furthermore the minimum state NFA's are not necessarily isomorphic. Finding a state minimal DFA equivalent to a given DFA can be performed in polynomial time; all minimum state DFA are isomorphic.

Definition 3 A Muller automaton (MA) is a 5-tuple (Σ, S, s_0, T, M) where Σ is a finite set called the *alphabet*, S is a finite set of *states*, $s_0 \in S$ is the initial state, $T \subset S \times \Sigma \times S$ is the *transition relation*, and $M \subset 2^S$ is the acceptance condition.

The automaton is said to be *deterministic* if $(\forall s)(\forall x)\,[\,|\,\{t : T(s,x,t)\}\,| \leq 1]$; otherwise it is said to be *non-deterministic.*

A string $x \in \Sigma^\omega$ is *accepted* by the FA if there exists a sequence of states $\sigma_0\sigma_1\ldots$ such that $\sigma_0 = s_0$, and $(\forall i)T(\sigma_i, x_i, \sigma_{i+1})$, and $inf(\sigma) \in M$, where $inf(\sigma)$ is the infinitary set of σ. The language of the automaton is the set of strings accepted by it; a language is said to be ω-regular if it is the language accepted by some Muller automaton.

Variants on the acceptance condition yield different classes of automaton on ω-strings.

1. The acceptance condition for Büchi automaton is a subset B of S; a run σ is accepting if $inf(\sigma) \cap B \neq \phi$.

2. The acceptance condition for Streett automaton is a set of pairs of subsets of the state space $\{(U_1, V_1), (U_2, V_2), \ldots, (U_n, V_n)\}$; σ is accepting if $(\forall i)\,(inf(\sigma) \cap U_i \neq \phi \rightarrow inf(\sigma) \cap V_i \neq \phi)$

3. The acceptance condition for Rabin automaton is a set of pairs of subsets of the state space $\{(U_1, V_1), (U_2, V_2), \ldots, (U_n, V_n)\}$; σ is accepting if $(\exists i)\,[(inf(\sigma) \cap U_i \neq \phi) \ \wedge \ (inf(\sigma) \cap V_i = \phi)]$

The following facts are proved in [19]:

- The class of ω-regular languages is closed under union, intersection, and projection; this follows from construction.

- Non-deterministic Büchi automaton (NBA) accept the class of ω-regular languages; deterministic Büchi automaton are strictly less expressive than (NBA).

- Deterministic Rabin Automaton (DRA) and deterministic Streett automaton accept the class of ω-regular languages.

- There is a sequence of ω-regular languages $L_1, L_2, \ldots$ such that the smallest BA accepting L_n has $\Omega(c^n)$ states, while the smallest RA for L_n has $O(n^k)$ states.

- Given an NBA on n states, there is a DSA on $O(2^{c \cdot n \cdot \log(n)})$ states accepting the same language.

- Deterministic Muller automaton (DMA) accept the class of ω-regular languages; from this it immediately follows that the class of ω-regular languages is closed under complement.

Definition 4 The logic S1S (*second order theory of one successor)* is a second order logic over the alphabet given by the set $\{0, S, =, <, \in, \wedge, \neg, \exists, x_1, x_2, \ldots, X_1, X_2, \ldots\}$. Lower case variables $x_1, x_2, \ldots$ are first order variables ranging over elements of the universe, and upper case variables $X_1, X_2, \ldots$ are second order variables ranging over subsets of the universe. The well formed formulae of the logic S1S are given by the following syntax:

- *Terms* are constructed from the constant 0 and first order variables by repeated applications of the successor function S. Examples of terms – 0, $SS0$, $SSSSx_3$.
- *Atomic formulae* are of the form $t_1 = t_2$, $t_1 < t_2$, $t \in X_k$. Examples of atomic formulas – $0 < S0$, $x_3 = SSSx_5$, $Sx_7 \in X_2$.
- *S1S formulas* are constructed from atomic formulas by using the boolean connectives $\wedge, \neg$ and quantification over both kinds of variables. Examples of S1S formulas – $(0 < S0) \wedge (Sx_7 \in X_2)$, $(\exists X)(\exists x)[(x \in X) \wedge (Sx \in X)]$. We write $\phi(X_1, X_2, \ldots, X_n)$ to denote that at most $X_1, X_2, \ldots, X_n$ occur free in ϕ (i.e. are not in the scope of any quantifier). We will routinely use the symbols $\vee, \leftarrow, \forall$, etc as logical abbreviations.

S1S can be interpreted over the following model: $\alpha = (\omega, 0, +1, <)$ where the underlying universe $\omega = \{0, 1, 2 \ldots\}$ is the set of natural numbers. The semantics of S1S over α is the usual Tarskian interpretation of truth; we illustrate this by means of an example.

Example 1: (Existence of least elements in non empty subsets of ω)

$$\psi \quad = \quad (\forall X)(\exists x)[(x \in X) \rightarrow (\exists y)((y \in X) \wedge \neg((\exists z)(z \in X \wedge (z < y))))]$$

The above sentence states that for every subset (X) of ω, if X is non-empty $(\exists x\,(x \in X))$, then it contains a least element (y). It is true in the model α.

Example 2: (Defining subsets of ω which contain 5 whenever they contain 3.)

$$\phi_0(X) \quad = \quad (SSS0 \in X) \rightarrow (SSSSS0 \in X)$$

Example 3: (Defining the subset of even integers)

$$\phi_1(X) \quad = \quad (0 \in X) \wedge \neg(1 \in X) \wedge (\forall x)(x \in X \leftrightarrow SSx \in X)$$

Example 4: (Defining the relation "every even number in X is in Y")

$$\phi_2(X,Y) \;=\; (\forall x)[(\forall Z)\,(\phi_1(Z) \wedge x \in Z) \rightarrow (x \in X \;\rightarrow\; x \in Y)]$$

The class of subsets of ω is in a one to one correspondence with the set of ω-sequences on $\{0,1\}$ – the 1's in the sequence can be thought of as representing the integers in the corresponding set, eg $01011 = \{1,3,4\}$. Thus, given a formula $\theta(X_1)$ in S1S, it naturally defines an ω-language on $\{0,1\}$, i.e. the set $\{\beta \subset \omega \mid \beta \models \theta(\mathbf{X_1})\}$. In general, formulas $\phi(X_1, X_2, \ldots, X_n)$ define languages over $(\{0,1\}^n)^\omega$. The following result is one of the cornerstones of descriptive set theory:

Theorem 2.1 (Büchi 1961 [19, 5]) *An ω-language is definable in S1S if and only if it is ω-regular.*

Proof: Refer to [19]. ∎

With minor modifications, Büchi's result also holds for sets of *finite* words i.e. when set quantification is restricted to finite sets only. In this case one speaks of the theory WS1S (weak S1S). The corresponding result for WS1S states that a $*$ language is definable in WS1S if and only if it is $*$-regular [19, 5].

Thus we can uniquely identify an automaton A (FA or MA) over the alphabet $\{0,1\}^n$, with a formula $\phi^A(X_1, X_2, \ldots, X_n)$. Hence, elegant yet rigorous proofs can be given to a large class of solutions to problems related to finite state systems. Furthermore these proofs are constructive; given a formula in S1S/WS1S it is possible to mechanically construct an equivalent automaton.

Definition 5 A *finite state machine* M is a 5-tuple (S, s_0, I, O, T) where S is a finite set of *states*, $s_0 \in S$ is the *initial* state, I is a finite set of *inputs*, O is a finite set of *outputs*, and $T \subseteq S \times I \times O \times S$ is the *transition relation*.
M is *deterministic* if

$$(\forall s)\,(\forall i)\,(\forall o) \;\; \mid \{t : T(s,i,o,t)\} \mid \leq 1$$

M is said to be *complete* if

$$(\forall s)\,(\forall i)\,(\exists o)\,(\exists t)\,[T(s,i,o,t)]$$

M is said to be an *implementation* if it is complete and

$$(\forall s)\,(\forall i) \;\; \mid \{t, o : T(s,i,o,t)\} \mid \leq 1$$

Given an input sequence $i = (i_1 i_2 \cdots i_n)$, and output sequence $o = (o_1 o_2 \cdots o_n)$,

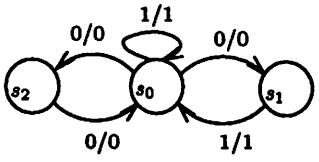

Figure 1: An example of an FSM which is not complete but still realizable

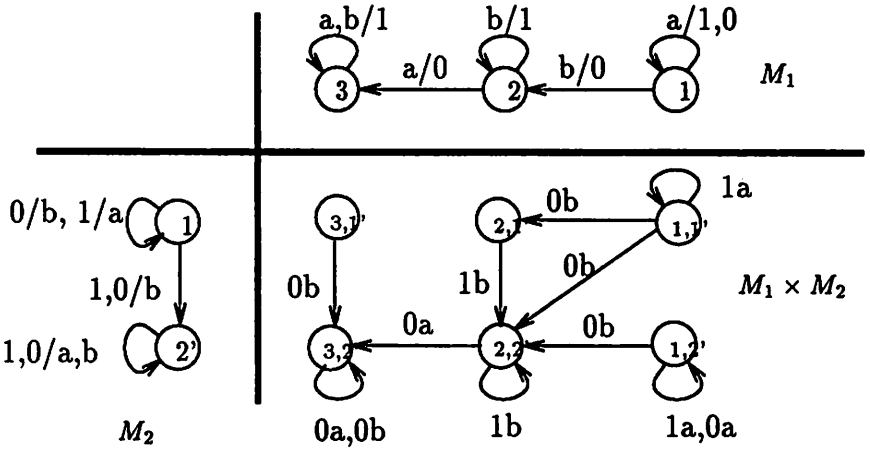

Figure 2: Product of M_1 and M_2

a *corresponding run* is a sequence of states $\sigma = (s_0 s_1 \cdots s_n)$ such that $(\forall k)\,(s_k, i_{k+1}, o_{k+1}, s_{k+1}) \in T$. The notion of a run trivially generalizes to infinite sequences.

Remark: Our definition of deterministic finite state machine has been referred to as "pseudo non-deterministic" (PNDFSM) in the past [22]. The term deterministic was reserved for what we refer to as implementable. A related notion is that of *incompletely specified* finite state machines (ISFSM), where given a state s and an input i, either $(\exists o)\,(\exists t)\,[T(s,i,o,t)]$ or $(\forall o)(\forall t)\,[T(s,i,o,t)]$.
Remark: Because of non-determinism a machine may be incompletely specified, but it may still be true that for all input sequences there are output sequences, as in figure 1.

Synchronous hardware consists of a set of interacting components driven by a common clock. Thus the composition of FSMs (referred to as the *product machine*) is synchronous; an example of the product machine construction is given in figure 2. Refer to [4] for details.

Definition 6 The $*$-language identified with an FSM $M = (S, r, I, O, T)$, denoted by $M_L(i,o)$, is the set of sequences in $(I \times O)^*$ such that for all $(i,o) \in$

$M_L(i,o)$ there exists a corresponding run.

Remark: Clearly the behavior $M_L(i,o)$ is a regular language([11, 19]) on the alphabet $I \times O$. The automaton M_A defining the language is simply the FSM itself – the states of M_A are the states S, initial states of M_A are the states r, the transition relation of the automaton is $\{(s,(i,o),t) : T(s,i,o,t)\}$, and all states are accepting. Note that the behavior of a deterministic machine is defined by a deterministic finite automaton.

Definition 7 Given a language $L \subset (\Sigma_I \times \Sigma_O)^*$, a finite state machine M with inputs Σ_I, outputs Σ_O is said to be *compatible* with L if $M_L(i,o) \subset L$; M is said to be a *realization* of L if it is compatible with L and is an implementation. A language is said to be *realizable* if there exists a realization of it; similarly an FSM is realizable if its language is realizable.

It has been empirically observed that finding a state minimal realization of the language defined by a PNDFSM is harder than finding a state minimal realization of an ISFSM even though both the associated decision problems are NP-complete.

Definition 8 A *finite state machine with Muller fairness* is a tuple (M,C) where M is an FSM, and C is a set of subsets of the states of M. Given $F = (M,C)$, an FSM with fairness, F is said to be deterministic (complete) if M is deterministic (complete).

The notion of a corresponding run over infinite input/output sequences is defined analogously to that for ordinary FSMs; a run σ is fair if the infinitary set of states is an element of C. Similarly the language of a finite state machine with fairness is defined to be the language of the corresponding Muller automaton.

3 Synthesizing $*$ languages

In this section we will restrict our attention to $*$-languages; in the next section we will see how our results generalize to ω-languages.

3.1 The E-machine

Consider machines communicating in the configuration shown in figure 3. Suppose x and y are the observable inputs and outputs, and $S(x,y)$ is a specification on them. Also suppose the machine $M(x,u,v,y)$ is fixed.

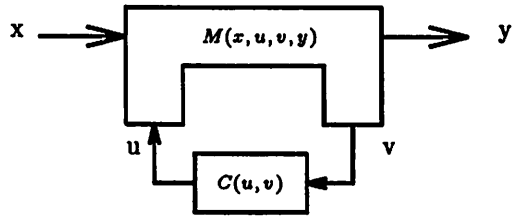

Figure 3: General example of interacting finite state systems

Theorem 3.1 The set of all behaviors on u, v which will yield behavior on the inputs and outputs which is compatible with the specification is given by the following expression:

$$\phi^{C^*}(u,v) \quad = \quad (\forall x,y)\,[\phi^M(x,y,u,v) \rightarrow \phi^S(x,y)] \tag{1}$$

Proof: Equation 1 could infact be taken as begin correct by construction. However we add to its credibility by showing an alternate game theoretic formulation.

View the problem of finding the set of u's which can be generated corresponding to a given v as the problem of finding strategies for a game, given that the player corresponding to the controller knows the systems structure, but can not observe x or y. The object of the controller is to play moves so that the specification is not violated. Then the strategy, based on perfect reasoning is the following: if $\tilde{x}$ was an input to M consistent with u, v being input/output, then all $\tilde{y}$ consistent with $\tilde{x}, u, v$ should be acceptable, and any other $\tilde{y}$ loses the game. Mathematically, this strategy is defined by the following (the reason for the odd notation will become apparent later)

$$\phi^{O_2^C}(v,u) \quad = \quad (\forall \tilde{x})\,[(\exists \tilde{y})\,\phi^M(\tilde{x},u,v,\tilde{y}) \rightarrow (\forall \tilde{y})\,(\phi^M(\tilde{x},u,v,\tilde{y}) \rightarrow \phi^S(\tilde{x},\tilde{y}))]$$

The following lemma proves that the strategy followed in the game yields the E-machine.

Lemma 3.1 $\vdash \phi^{O_2^C}(v,u) \leftrightarrow \phi^{C^*}(v,u)$

Proof:

$$\begin{aligned}
&(\forall \tilde{x})\,[(\exists \tilde{y})\phi^M(\tilde{x},u,v,\tilde{y}) \rightarrow (\forall \tilde{y})\,(\phi^M(\tilde{x},u,v,\tilde{y}) \rightarrow \phi^S(\tilde{x},\tilde{y}))] \\
&\qquad \leftrightarrow (\forall \tilde{x})\,[\neg(\exists \tilde{y})\phi^M(\tilde{x},u,v,\tilde{y}) \ \wedge \ (\forall \tilde{y})\,(\neg\phi^M(\tilde{x},u,v,\tilde{y}) \ \wedge \ \phi^S(\tilde{x},\tilde{y}))] \\
&\qquad \leftrightarrow (\forall \tilde{x})\,[(\forall \tilde{y})\,\phi^M(\tilde{x},u,v,\tilde{y}) \ \wedge \ (\forall \tilde{y})\,(\neg\phi^M(\tilde{x},u,v,\tilde{y}) \ \wedge \ \phi^S(\tilde{x},\tilde{y}))]
\end{aligned}$$

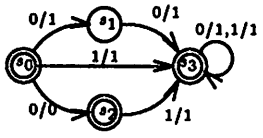

Figure 4: A system for which every input can be controlled, by no realizable controller exists

$$\leftrightarrow (\forall \tilde{x})\,[(\forall \tilde{y})\,(\neg \phi^M(\tilde{x},u,v,\tilde{y}) \ \wedge \ \phi^S(\tilde{x},\tilde{y}))]$$
$$\leftrightarrow (\forall \tilde{x})\,(\forall \tilde{y})[\phi^M(\tilde{x},u,v,\tilde{y}) \rightarrow \phi^S(\tilde{x},\tilde{y})]$$

■

The equivalence of the two completes the proof of the theorem. ■

Equation 1 can be rewritten to give the following expression:

$$\phi^{C^*}(u,v) \quad = \quad \neg\,(\exists x,y)\,[\phi^M(x,y,u,v) \wedge (\neg \phi^S(x,y))] \qquad (2)$$

By the remarks in section 2, $C^*(u,v)$ is regular. Furthermore, the number of states in the DFA derived from the expression is upper bounded by $2^{|S_M|\cdot 2^{|S_S|}}$, where S_S, S_M are the state spaces of the automata defining $S_L(x,y)$ and $M(x,u,v,y)$. In the special case when the automaton defining $S_L(x,y)$ is deterministic, the corresponding upper bound is $2^{|S_M|\cdot|S_S|}$. This is precisely the construction of [22].

The specification automaton could be $M \times C$; this corresponds the *re-synthesis* problem, i.e. suppose we wish to find a replacement for the C block which is optimal (with respect to an appropriate objective function) while preserving the observed behavior. Then the behavior of the replacement must be contained in the behavior $C^*(v,u)$.

In the most general setting M and the specification automaton are non-deterministic and incompletely specified. In this case, simply deciding if an implementation exists for the block C which is compatible with the specification is non-trivial; this motivates the next section.

3.2 Discrete Control

There are obvious applications of the theory developed above to discrete control. Indeed the set $C^*(u,v)$ is the acceptable controller behavior. The set of inputs which can be controlled is defined by the formula

$$\phi^{D^*}(x) \quad = \quad (\exists u,v,y)\,[\phi^M(x,u,v,y) \rightarrow \phi^S(x,y)]$$

Algorithm_Star_Controller_Strategy:(*DFA_type:* **D**)
while (TRUE) {
remove states s from **D** such that
$\neg[(\exists u,t)(T(s,(0,u),t) \wedge$
$(t \in A)) \wedge (\exists u',t')T((s,(0,u'),t') \wedge (t' \in A))]$
if (no states were removed)
break;
}

Figure 5: Algorithm for deciding if a strategy exists

This does not mean that a controller can be realized which will generate acceptable outputs for all inputs in $D^*(x)$; this is due to the usual problem with causality, viz the control input may be forced to guess the future values of the input. Conceptually an implementable controller must be finite state and causal; in [16] it is argued that a necessary and sufficient condition for implementability is that a *strategy tree* must exist for a player observing inputs over V and producing outputs over U while ensuring that the input-output behavior is compatible with the relation $C^*(u,v)$.

Deciding if an implementable controller exists is simply the Church solveability problem [17], which can be settled using algorithms for emptiness of tree automaton.

We are given $C^*(v,u) \subset (\Sigma_V \times \Sigma_U)^*$ and wish to determine the existence of a finite state machine $C(v,u)$ which is compatible with $C^*(v,u)$ in the sense $(\forall v,u)[\phi^{C(v,u)} \rightarrow \phi^{C^*(v,u)}]$. The procedure for doing this is as follows:

1. Determinize the $*$-automaton for $C^*(v,u)$ by the usual power set construction

2. Use the algorithm in figure 5 to check if a strategy tree exists: (For simplicity we assume $\Sigma_V = \{0,1\}$)

Lemma 3.2 An implementable controller exists if and only if the DFA left remaining after this algorithm converges is non empty.

If the DFA is nonempty, the existence of a controller strategy (finite state machine implementing acceptable control) is proved. However we may want to

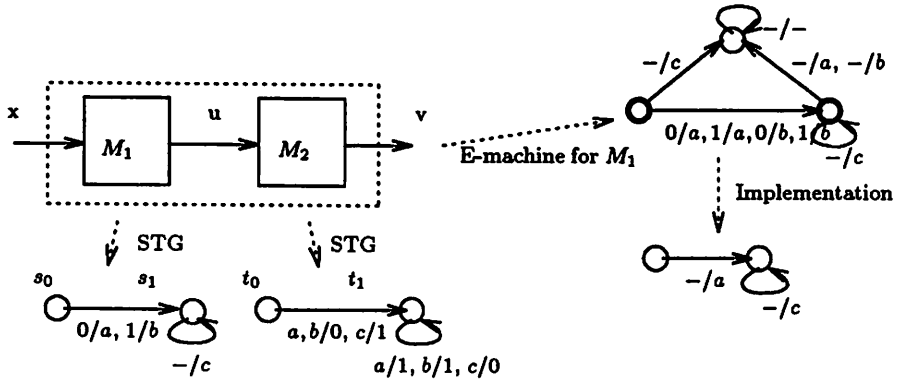

Figure 6: A simple cascade system in which the range of permissible behaviors for the driving machine is not definable by an ISFSM.

come up with state minimal strategies; in the $*$-case, the fixed point DFA of the above algorithm is a PNDFSM. If the tree language is non empty, there is not necessarily one strategy which is maximum, since controlling one input may come at the cost of not controlling another. However again, we believe there are simple greedy strategies for coming up with maximal controller strategies, i.e. controller that control a maximal set of input behaviors.

There are variations in the formulation of the discrete event system control problem. One is the *model matching* problem in which the system is characterized by the finite state machine $M(u, y)$, and the specification is given by $S(x, y)$. The objective is to design a controller which takes as input x, y and outputs u so that the composition of the plant and the controller satisfies the specification. The set of acceptable controller moves is given by

$$\phi^{C^*}(x, y, u) \;=\; \phi^M(u, y) \rightarrow \phi^S(x, y)$$

The algorithm of figure 5 can be applied to $C^*(x, y, u)$ to determine if a controller exists; take the input to be the tuple (x, y), and the output to be u.

3.3 Optimization of Interacting FSMs

We now turn our attention to networks of finite state machines. For such systems, the full range of admissible behavior at a node is described by the E-machine. We now provide more intuitive descriptions of the flexibility at a component.

Let x, y be the input and output of the FSM network. Let $S(x, y)$ be the specified input/output behavior of the network. Consider a component machine

C on inputs v and outputs u. Let $M(x,u,v,y)$ be the rest of the network.

Definition 9 The *strong satisfiability dont care set* for $C(v,u)$ is logically defined by the following formula:

$$\phi^{SDC_0^C}(v) \quad = \quad \neg(\exists \tilde{x},\tilde{u},\tilde{y})\,\phi^M(\tilde{x},\tilde{u},v,\tilde{y})$$

It is precisely the set of sequences over v which can never be generated, no matter what replacement is used for C.

This set gives a certain amount of flexibility in choosing implementations for C; namely any behavior in the machine $C_0(u,v)$ defined below is acceptable.

$$\phi^{C_0}(v,u) \quad = \quad \neg\phi^{SDC_0^C}(v) \rightarrow \phi^C(v,u)$$

The following lemma states that the construction is sound but not complete:

Lemma 3.3 $\phi^{C_0}(v,u) \vdash \phi^{C^*}(u,v)$ but $\phi^{C^*}(u,v) \nvdash \phi^{C_0}(v,u)$

Definition 10 The *weak satisfiability dont care set* for $C(v,u)$ is mathematically given by the following expression:

$$\phi^{SDC_1^C}(v) \quad = \quad \neg(\exists \tilde{x},\tilde{u},\tilde{y})\,[\phi^M(\tilde{x},\tilde{u},v,\tilde{y}) \wedge \phi^C(\tilde{u},v)]$$

It is precisely the set of sequences over v which can never be generated in the product machine $M \times C$, and corresponds to the *input dont care* sequences of [20].

This set gives more flexibility in choosing implementations for C; namely any behavior in the machine $C_1(u,v)$ defined below is acceptable.

$$\phi^{C_1}(v,u) \quad = \quad \neg\phi^{SDC_1^C}(v) \rightarrow \phi^C(v,u)$$

The following lemma states that the construction is sound, yields more flexibility than the previous construction, but is still not complete.

Lemma 3.4 $\phi^{C_1}(v,u) \vdash \phi^{C_0}(u,v)$; but $\phi^{C_0}(v,u) \nvdash \phi^{C_1}(v,u)$. $\phi^{C_1}(v,u) \vdash \phi^{C^*}(u,v)$ but $\phi^{C^*}(u,v) \nvdash \phi^{C_1}(v,u)$

Proof: We provide a formal proof (in the sense of mathematical logic – refer [10]).
TPT

$$(\exists \tilde{x},\tilde{u},\tilde{y})\,[\phi^M(\tilde{x},\tilde{u},v,\tilde{y}) \wedge \phi^C(\tilde{u},v)] \rightarrow \phi^C(u,v) \quad \vdash \quad (\forall x,y)\,[\phi^M(x,y,u,v) \rightarrow \phi^S(x,y)]$$

We show the contrapositive:

$$(\exists x, y)\,[\phi^M(x,y,u,v) \wedge \neg\phi^S(x,y)] \;\vdash\; (\exists \tilde{x}, \tilde{u}, \tilde{y})\,[\phi^M(\tilde{x},\tilde{u},v,\tilde{y}) \wedge \phi^C(\tilde{u},v)] \wedge \neg\phi^C(u,v)$$

First we show

$$(\exists x, y)\,[\phi^M(x,y,u,v) \wedge \neg\phi^S(x,y)] \;\vdash\; \neg\phi^C(u,v) \qquad (3)$$

Since $M \otimes C$ satisfies the original specification, the following is an axiom:

$$\kappa^C \;=\; (\forall \tilde{x}, \tilde{u}, \tilde{v}, \tilde{y})\,[\phi^M(\tilde{x},\tilde{u},\tilde{v},\tilde{y}) \wedge \phi^C(\tilde{u},\tilde{v}) \rightarrow \phi^S(\tilde{x},\tilde{y})]$$

By generalization,

$$\kappa^C \;\vdash\; (\forall \tilde{x}, \tilde{y})\,[\phi^M(\tilde{x},u,v,\tilde{y}) \wedge \phi^C(u,v) \rightarrow \phi^S(\tilde{x},\tilde{y})]$$

applying generalization and tautology again,

$$\kappa^C \;\vdash\; \phi^C(u,v) \rightarrow (\forall \tilde{x}, \tilde{y})[\phi^M(\tilde{x},u,v,\tilde{y}) \rightarrow \phi^S(\tilde{x},\tilde{y})]$$

by the contrapositive,

$$\kappa^C,\, (\exists \tilde{x}, \tilde{y})\,[\phi^M(\tilde{x},u,v,\tilde{y}) \wedge \neg\phi^S(\tilde{x},\tilde{y})] \;\vdash\; \neg\phi^C(u,v)$$

This completes the proof of 3. Now we show

$$(\exists \tilde{x}, \tilde{y})\,[\phi^M(\tilde{x},\tilde{y},u,v) \wedge \phi^S(\tilde{x},\tilde{y})] \;\vdash\; (\exists \tilde{u}, \tilde{x}, \tilde{y})\,[\phi^M(\tilde{x},\tilde{y},\tilde{u},\tilde{v}) \wedge \phi^C(\tilde{u},v)] \qquad (4)$$

By tautology, STS

$$(\exists \tilde{x}, \tilde{y})\,[\phi^M(\tilde{x},\tilde{y},u,v)] \;\vdash\; (\exists \tilde{u}, \tilde{x}, \tilde{y})\,[\phi^M(\tilde{x},\tilde{y},\tilde{u},\tilde{v}) \wedge \phi^C(\tilde{u},v)]$$

By generalization, STS

$$(\forall \tilde{u})\,(\exists \tilde{x}, \tilde{y})\,[\phi^M(\tilde{x},\tilde{y},\tilde{u},v)] \;\vdash\; (\exists \tilde{u})\,[(\exists \tilde{x}, \tilde{y})\,\phi^M(\tilde{x},\tilde{u},v,\tilde{y}) \wedge \phi^C(\tilde{u},v)]$$

But since C is an implementation, we have as an axiom:

$$\lambda^C \;=\; (\forall \tilde{v})(\exists \tilde{u})\,[\phi^C(\tilde{u},\tilde{v})]$$

By generalization,

$$\lambda^C \;\vdash\; (\exists \tilde{u})\,[\phi^C(\tilde{u},v)]$$

Hence,

$$\lambda^C,\ (\forall \tilde{u})\,(\exists \tilde{x}, \tilde{y})\ [\phi^M(\tilde{x}, \tilde{y}, \tilde{u}, v)] \ \vdash\ (\exists \tilde{u})\,[\phi^C(\tilde{u}, v) \wedge \phi^M(\tilde{x}, \tilde{y}, \tilde{u}, v)]$$

proving 4.

The conjunction of the results in 3 and 4 yields the lemma. ∎

Definition 11 The *strong observability equivalence relation* for $C(v, u)$ is mathematically given by the following expression:

$$\phi^{\mathcal{O}_0^C}(v) \ = \ (\forall \tilde{x}, \tilde{y}, \tilde{u})\,[\phi^M(\tilde{x}, \tilde{u}, v, \tilde{y}) \rightarrow \phi^S(\tilde{x}, \tilde{u})]$$

It is precisely the set of sequences over v for which any output sequence can be safely generated. Clearly, for input sequences which are never generated any output is acceptable, so $\phi^{SDC_1^C}(v) \rightarrow \phi^{\mathcal{O}_0^C}(v)$.

This set gives still more flexibility in choosing implementations for C; namely any behavior in the machine $C_2(u, v)$ defined below is acceptable.

$$\phi^{C_2}(v, u) \ = \ \neg\phi^{\mathcal{O}_0^C}(v) \rightarrow \phi^C(u, v)$$

The following lemma states that the construction is sound, yields more flexibility than the previous construction, but is still not complete.

Lemma 3.5 $\phi^{C_2}(v, u) \vdash \phi^{C_1}(u, v)$; but $\phi^{C_1}(v, u) \not\vdash \phi^{C_2}(v, u)$. $\phi^{C_2}(v, u) \vdash \phi^{C^*}(u, v)$ but $\phi^{C^*}(u, v) \not\vdash \phi^{C_2}(v, u)$. In fact $\phi^{\mathcal{O}_0^C}(v) \leftrightarrow (\forall \tilde{u})\,\phi^{C^*}(v, u)$

Definition 12 The *weak observability equivalence relation* for $C(v, u)$ is mathematically given by the following expression:

$$\phi^{\mathcal{O}_1^C}(v, u) \ = \ (\forall \tilde{x})\,[(\exists \tilde{u}, \tilde{y})(\phi^M(\tilde{x}, \tilde{u}, v, \tilde{y}) \rightarrow (\forall \tilde{y})\,(\phi^M(\tilde{x}, u, v, \tilde{y}) \rightarrow \phi^S(\tilde{x}, \tilde{y})))]$$

Sequences v and u are in this relation precisely when the following condition is met for every external input:

- if $\tilde{x}$ was an input to M which was consistent with v being output, then all primary outputs $\tilde{y}$ consistent with $\tilde{x}, u, v$ should be acceptable.

This set gives yet more flexibility in choosing implementations for C; namely any behavior in the machine $C_3(u, v)$ defined below is acceptable.

$$\phi^{C_3}(v, u) \ = \ \phi^{\mathcal{O}_1^C}(v, u)$$

Because this relation requires that for *any* primary input x which could produce v the output must be consistent (and hence ignores the requirement that x must be consistent with u also) it is still incomplete.

Lemma 3.6 $\phi^{C_3}(v,u) \vdash \phi^{C_2}(v,u)$; but $\phi^{C_2}(v,u) \not\vdash \phi^{C_3}(v,u)$. $\phi^{C_3}(v,u) \vdash \phi^{C^*}(u,v)$ but $\phi^{C^*}(u,v) \not\vdash \phi^{C_2}(v,u)$.

Definition 13 The *true observability equivalence relation* for $C(v,u)$ is mathematically given by the following expression:

$$\phi^{\mathcal{O}_2^C}(v,u) \quad = \quad (\forall \tilde{x})\,[(\exists \tilde{y})\,\phi^M(\tilde{x},u,v,\tilde{y}) \rightarrow (\forall \tilde{y})\,(\phi^M(\tilde{x},u,v,\tilde{y}) \rightarrow \phi^S(\tilde{x},\tilde{y}))]$$

As was shown in theorem 3.1, the true observability equivalence relation is equivalent to the E-machine, and hence captures all the flexibility possible for synthesizing C. Let C_4 be the machine defined by $\phi^{\mathcal{O}_2^C}(v,u)$

3.4 Deriving Optimal Implementations

Given the E-machine, one would like to derive an implementation that is optimal. One criterion for optimality is state minimality. State minimization of incompletely specified state machines (ISFMs) is easier than that for pseudo non-deterministic finite state machines (PNDFSMs). Our interest in input dont care sequences and satisfiability don't care sequences stems from the fact that using them for flexibility gives ISFSMs rather than PNDFSMs.

Usually, the E-machine is a PNDFSM, rather than a ISFSM. In the case of cascade machines, as in figure 6, the E-machine for C is an ISFSM. It would seem that a simple topology might be in certain situations guarantee that the E-machine is an ISFSM. However this is not the case. Even in the simple cascade circuit of figure 6, the E-machine for M is not an ISFSM, even though there is some flexibility in choosing an implementation. If it were, then for some state and input, the machine should be allowed to transit to any state while producing any input, which is clearly not the case.

Lemma 3.7 The machines C_0, C_1, C_2 are always ISFMS's while the machines C_3, C_4 are PNDFSM's.

3.5 Combinational Resynthesis of Sequential Hardware

The theory developed in the previous section can also be applied to optimizing sequential designs specified as combinational blocks interacting with sequential elements, i.e. netlists of interconnected gates and flip flops. Given a sequential circuit we will now characterize the changes which can be made to the associated

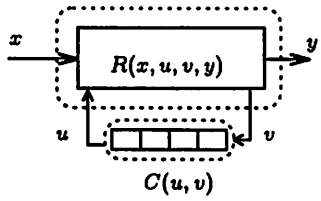

Figure 7: R is the relation defining the combinational portion of the circuit. C is the FSM derived from the latches $l_1, l_2, \ldots, l_L$ in the circuit; let $(i_1 i_2 \ldots i_L)$ be the initial state of the design. Then $u[i+1] = v[i]$ and $u[0] = (i_1 i_2 \ldots i_L)$

combinational logic without any changes to the latches. Pictorially, this is represented in figure 7.

The "E-machine" at R (keeping C fixed) is given by the following

$$\phi^{M^*}(x, u, v, y) \quad = \quad \phi^C(v, u) \rightarrow \phi^S(x, y)$$

where $\phi^S(x, y) = (\exists \tilde{u}, \tilde{v})\,(\phi R(x, \tilde{u}, \tilde{v}, y) \wedge \phi^C(\tilde{u}, \tilde{v}))$ is the formula defining the original machine.

Generally, $\phi^{M^*}(x, u, v, y)$ is a sequential machine. Valid combinational implementations include:

$$\begin{aligned}
\phi^{R_1}(x, u, v, y) &= \phi^R(x, v, u, y) \\
\phi^{R_2}(x, u, v, y) &= (\exists \tilde{v})\,(\phi^{\mathcal{E}}(v, \tilde{v}) \wedge \phi^R(x, \tilde{v}, u, y)) \\
\phi^{R_3}(x, u, v, y) &= \phi^R(x, v, u, y) \vee \neg(\phi^{Rch}(u))
\end{aligned}$$

where $\phi^{\mathcal{E}}(v, v')$ is the relation that states v, v' are input-output equivalent, and $\phi^{Rch}(u)$ is the relation that u is reachable in the original machine. Variants of the above are given in [9].

Definition 14 The *static dont care set* for the design is the set of all combinational functions that map $\Sigma_X \times \Sigma_U$ to $\Sigma_Y \times \Sigma_V$ which are implementations of $\phi^{M^*}(x, u, v, y)$.

The use of the term "dont care" set is of course a misnomer; the class of functions comprising the set is highly discontinuous, and not even definable by a relation let alone a single function with a dont care set. In fact, generally deciding if a given NDFSM has a combinational implementation is NP-complete, as is shown by the following lemma.

Lemma 3.8 Given an NDFSM $\mathcal{N}$ on input Σ_U and outputs Σ_V, it is NP-complete to decide if there exists a combinational implementation compatible

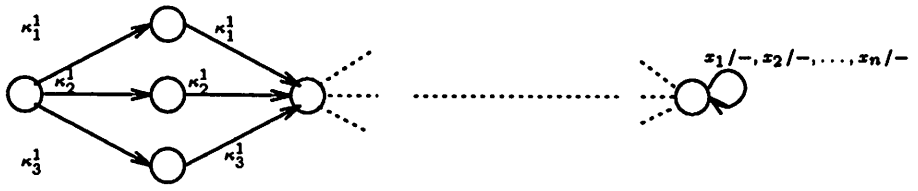

Figure 8: We are taking $C_1 = (\mathbf{x_1} \vee \bar{\mathbf{x_2}} \vee \mathbf{x_3})$ for illustration. $\kappa_1^1 = x_1/1, x_2/-, x_3/-, \ldots, x_n/-$; $\kappa_2^1 = x_1/-, x_2/0, x_3/-, \ldots, x_n/-$; $\kappa_3^1 = x_1/-, x_2/-, x_3/1, \ldots, x_n/-$;

with $\mathcal{N}$, i.e a function $f : \Sigma_U \rightarrow \Sigma_V$ such that $\phi^I \rightarrow \phi^{\mathcal{N}}$ holds.

Proof: Membership in NP is trivial: a combinational implementation can be described in $O(\mid \Sigma_U \mid \cdot \mid \Sigma_V \mid)$ bits, and tested for containment in NDFSM in polytime.

To demonstrate NP-hardness we use a reduction from 3-SAT. Let $\phi = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ be a given 3-CNF form over variables $x_1, x_2, \ldots, x_n$. Construct a PNDFSM over inputs $\{x_1, x_2, \ldots, x_n\}$ and outputs $\{0, 1\}$ as shown in figure 8.

Combinational implementations of the above PNDFSM are maps from the set $\{x_1, x_2, \ldots, x_n\}$ to $\{0, 1\}$, and are in a natural correspondence with truth assignments to the variables in ϕ. It should be clear from the construction that a combinational implementation corresponds to a truth assignment that satisfies ϕ and vice versa. The reduction is in polynomial time, and thus deciding combinational implementability is NP-complete. ∎

4 Synthesizing General Finite State Systems

4.1 Fair Systems

The analysis of systems with *fairness* is of growing importance. This is in part due to the advent of formal verification [2, 15]. In order to verify large systems, they have to be simplified; in practise this is often done by abstraction, i.e. adding behaviors (possibly through non-determinism) that the original system did not have in order to obtain a more compact representation. Verification performed on the abstract system is usually conservative. To get a more accurate representation of the system, fairness constraints, which are restrictions on the infinitary behavior of the system, are added. Fairness also plays an integral role in the *top-down* paradigm for hierarchical design paradigm proposed by [13].

Since fairness is a restriction on the infinitary behavior of the system, the defining relations for languages derived from FSMs with fairness are formula

in S1S rather than WS1S. Thus, the definition for the E-machine continues to give the range of permissible behaviors at a node in the context of ω-languages. The formulae for C_0, C_1, C_2, C_3, C_4 continue to define approximations to the E-machine; any implementation compatible with an approximation is correct.

For the case of infinite strings, i.e. $L^{C^*}(v,u) \subset (\Sigma_V \times \Sigma_U)^\omega$, defined by a non-deterministic Buchi automaton over the alphabet $\Sigma_V \times \Sigma_U$ the procedure for determining if a finite state machine $C(v,u)$ exists which implements $C^*(v,u)$ in the sense $(\forall v)(\forall u)\ [\phi^C(v,u) \rightarrow \phi^{C^*}(v,u)]$ is as follows:

1. Determinize the NB automaton defining $C^*(v,u)$ to obtain a deterministic Rabin automaton (*Complexity* — $O(2^{n \cdot \log(n)})$, where n is the number of states in the NB automaton)

2. In the DR automaton, project the symbols of the alphabet $\Sigma_V \times \Sigma_U$ down to Σ_V. Interpret the new structure as a Rabin automaton on *trees* and check for tree emptiness; An implementable controller exists if and only if the tree emptiness is negative. The algorithm of Rosner [16] for Rabin tree emptiness actually derives an implementation if one exists. (Complexity: Rabin tree emptiness is known to be NP-complete; the algorithm of [16] has complexity polynomial in the number of states and exponential in the number of accepting pairs.)

We would like to maintain some degree of optimality in the controller synthesis process; one criterion for optimality could be state minimality. For $*$-languages we were able to reduce the search for state minimal implementations to finding the minimum state implementation in a PNDFSM. In general, this is not the case for ω-languages.

4.2 Timed Systems

The formal analysis of *real time systems* is an area of active research [1]. The behavior of a timed system is now a map from $\Re$ rather than ω as was the case for discrete time systems. Languages can be defined in terms of sets of maps from $\Re$ to the output, a finite set of scalars. The real time control/synthesis problem is defined in a manner analogous to that for discrete time.

Let $S(x,y)$ a timed automaton whose language describes an acceptable relationship between the input timed trace x and the output timed trace y, and $M(x,y,u,v)$ a timed automaton on inputs x,u and outputs y,v. The game theoretic formulation and derivation of the E-machine continues to hold – the set $C^*(u,v)$ of strategies for a controller which can observe v and control u which yields acceptable behavior is still given by $\neg(\exists \tilde{x}, y)\,[\phi^M(x,y,u,v) \wedge \neg(\phi^S(x,y))]$,

where ϕ^A is a formula defining the language of the timed system A in an appropriate logic.

Different formulations of timed automaton yield different classes of definable timed languages. Unfortunately, none of the non-deterministic timed automaton formulations define classes which are closed under complement. Even if we start with deterministic timed automaton which are closed under complement (eg timed deterministic Muller automaton), the quantification step in the defining relation for C^* will lead to non-deterministic timed automaton; thus even for such simple systems there is no "regular" form for $C*$.

4.3 Stochastic Systems

A formal analysis of definitions and verification of stochastic systems has been carried out in [3]. Let the system to be controlled be a Markov process $M(x, y, u, v)$ where x, u are inputs and y, v are outputs, and $S(x, y) \subset {\Sigma_I}^* \times {\Sigma_o}^*$ be the specification. Given that the controller observes v and controls u, the *stochastic control* problem in this context can be cast as finding the "best" control strategy, i.e. given input v the controller should play v such that the probability that the system fails the specification is minimized. The following two strategies are variants on this theme.

The relation below defines the strategy which seeks minimize the net probability of failure. Formally define C^* as follows:

$$C_1^*(u, v) = (\forall \tilde{v})\,[\,\mu\{x, y \mid M(x, u, \tilde{v}, y) \wedge \neg S(x, y)\} \;\geq\; \mu\{x, y \mid M(x, u, v, y) \wedge \neg S(x, y)\}\,]$$

where μ is the measure function derived from the Markov process.

The following strategy seeks to minimize the maximum probability of a specific failure.

$$C_\infty^*(u, v) \;=\; (\forall \tilde{v})\,[\,\max_{\{x,y \mid \neg S(x,y)\}} \mu(M(x, u, \tilde{v}, y)) \;\geq\; \max_{\{x,y \mid \neg S(x,y)\}} \mu(M(x, u, v, y))\,]$$

5 Conclusion

In report we have described applications of the sequential calculus S1S to problems related to the synthesis and control of interacting finite state systems. We gave a detailed analysis of FSMs on finite strings, and related our results to past work. We also sketched extensions of the theory to more general classes of finite states systems, including fair, timed, and stochastic systems. In the future we plan to study these systems in more detail.

References

[1] R. Alur and D. L. Dill. Automata for Modelling Real Time Systems. In *International Colloquium on Automata, Languages and Programming*, 1990.

[2] A. Aziz, F. Balarin, R. K. Brayton, S.-T. Cheng, R. Hojati, T. Kam, S. C. Krishnan, R. K. Ranjan, A. L. Sangiovanni-Vincentelli, T. R. Shiple, V. Singhal, S. Tasiran, and H.-Y. Wang. HSIS: A BDD-Based Environment for Formal Verification. In *Proc. of the Design Automation Conf.*, June 1994.

[3] A. Aziz, V. Singhal, F. Balarin, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. The Temporal Logic of Stochastic Systems. Submitted for publication, 1995.

[4] A. Aziz, V. Singhal, G. M. Swamy, and R. K. Brayton. Minimizing Interacting Finite State Machines: A Compositional Approach to the Language Containment Problem. In *Proc. Intl. Conf. on Computer Design*, pages 255–261, Oct. 1994.

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

[6] E. M. Clarke and E. A. Emerson. Design and Synthesis of Synchronization Skeletons Using Branching Time Logic. In *Proc. Workshop on Logic of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer-Verlag, 1981.

[7] S. Devadas. Optimizing Interacting Finite State Machines Using Sequential Don't Cares. *IEEE Trans. Comput.-Aided Design Integrated Circuits*, pages 1473–1484, Dec. 1991.

[8] J. Fron, J. C.-Y. Yang, M. Damiani, and G. D. Micheli. A Synthesis Framework Based on Trace and Automata Theory. In *Workshop Notes of Intl. Workshop on Logic Synthesis*, Tahoe City, CA, May 1993.

[9] H. L. Harkness and G. M. Swamy. Minimization of Transition Relation Representations for Complex FSM's. EE290H Class Project Report, Dec. 1992.

[10] H. Henderton. *A Mathematical Introduction to Logic* . Academic Press, 1972.

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

[12] J. Kim and M. M. Newborne. The Simplification of Sequential Machines With Input Restrictions. *IRE Transactions on Electronic Computers*, pages 1440–1443, Dec. 1972.

[13] R. P. KURSHAN. REDUCIBILITY IN ANALYSIS OF COORDINATION. IN *Discrete Event Systems: Models and Applications*, VOLUME 103 OF *LNCIS*, PAGES 19–39. SPRINGER-VERLAG, 1987.

[14] Z. MANNA AND P. WOLPER. SYNTHESIS OF COMMUNICATING PROCESSES FROM TEMPORAL LOGIC SPECIFICATIONS. *ACM Trans. Prog. Lang. Syst.*, 6(1):68–93, 1984.

[15] K. L. MCMILLAN. *Symbolic Model Checking.* KLUWER ACADEMIC PUBLISHERS, 1993.

[16] A. PNEULI AND R. ROSNER. ON THE SYNTHESIS OF A REACTIVE MODULE. IN *Proc. ACM Symposium on Principles of Programming Languages*, PAGES 179–190, 1989.

[17] M. O. RABIN. *Automata on Infinite Objects and Church's Problem*, VOLUME 13 OF *Regional Conf. Series in Mathematics.* AMERICAN MATHEMATICAL SOCIETY, PROVIDENCE, RHODE ISLAND, 1972.

[18] J.-K. RHO, G. HACHTEL, AND F. SOMENZI. DON'T CARE SEQUENCES AND THE OPTIMIZATION OF INTERACTING FINITE STATE MACHINES. IN *Proc. Intl. Conf. on Computer-Aided Design*, PAGES 418–421, NOV. 1991.

[19] W. THOMAS. AUTOMATA ON INFINITE OBJECTS. IN J. VAN LEEUWEN, EDITOR, *Formal Models and Semantics*, VOLUME B OF *Handbook of Theoretical Computer Science*, PAGES 133–191. ELSEVIER SCIENCE, 1990.

[20] H.-Y. WANG AND R. K. BRAYTON. INPUT DON'T CARE SEQUENCES IN FSM NETWORKS. IN *Proc. Intl. Conf. on Computer-Aided Design*, 1993.

[21] H.-Y. WANG AND R. K. BRAYTON. PERMISSIBLE OBSERVABILITY RELATIONS IN FSM NETWORKS. IN *Proc. of the Design Automation Conf.*, JUNE 1994.

[22] Y. WATANABE AND R. K. BRAYTON. THE MAXIMUM SET OF PERMISSIBLE BEHAVIORS FOR FSM NETWORKS. IN *Proc. Intl. Conf. on Computer-Aided Design*, 1993.

[23] H. WONG-TOI AND D. L. DILL. SYNTHESIZING PROCESSES AND SCHEDULERS FROM TEMPORAL SPECIFICATIONS. IN *Proc. of the Second Workshop on Computer-Aided Verification*, 1990.