Environment Modeling and Language Universality

RICHARD RAIMI
Motorola Corporation
RAMIN HOJATI
HDAC, Inc.
and
KEDAR S. NAMJOSHI
University of Texas at Austin

In this paper we outline a theory for the *environment-modeling problem*, the problem of abstracting component finite state machines (FSMs) bordering a particular FSM of interest within a network of interacting FSMs. The goal is to lay a theoretical foundation for the automatic state reduction of large FSM networks. We feel this is a prerequisite for the efficient use of many verification techniques.

We focus on computing conditions for the *safe removal* of a component FSM in a FSM network, where removal is safe if it preserves a certain well-defined trace equivalence. We present an optimized algorithm for determining *language universality* of a FSM, as well as determining *independence* of a FSM from those of its inputs connected to outputs of neighboring FSMs. These two properties, input independence and language universality, provide the necessary and sufficient conditions for safe removal. In addition, we show how simulation relations can be utilized, both to reduce the cost of computing safe removal and to create an appropriate abstract FSM when safe removal is not possible.

Categories and Subject Descriptors: J.6 [**Computer Applications**]: Computer-Aided Engineering; B.8.1 [**Performance and Reliability**]: Reliability, Testing, and Fault-Tolerance

General Terms: Algorithms, Design, Reliability, Theory, Verification

Additional Key Words and Phrases: Abstraction, language universality, environment modeling, model checking

Authors' addresses: R. Raimi, BOPS, Inc., 7719 Woodhollow, Austin, TX 78731; email: richardr@bops.com or raimi@alumni.utexas.net; R. Hojati, HDAC, Inc., 2417 Mariner Square Loop, Suite 260, Alameda, CA 94501; email: rhojati@hdac.com; K. S. Namjoshi, Bell Laboratories, Lucent Technologies, 600–700 Mountain Avenue, Murray Hill, NJ 07974; email: kedar@research.bell-labs.coms.

1. INTRODUCTION

Consider a network of interacting finite state machines (FSMs), in which a particular component FSM is the *target* of some verification effort. The FSM network can be divided into the *target FSM* and the composition of all the other FSMs in the network, the latter being the *concrete environment*. *Environment modeling* refers to the creation of an abstract FSM that replaces the concrete environment when verification is run. Ideally, this *environment model* should

(1) have a much smaller representation than the concrete environment, and

(2) preserve trace equivalence over the inputs and outputs of the target FSM, when composed with it.

The need for what we term *environment modeling* arises often in VLSI design. While it is not always profitable to model sequential circuits as FSMs, in many cases it is. In such cases one often needs to find an abstraction for a concrete FSM network. Examples are the use of abstract models of circuits in behavioral simulation, the assumption of constraints on circuit inputs when doing Boolean-equivalence checking, fault-test generation, or static-timing analysis. Environment modeling is of vital importance in the formal-verification technique of model-checking as well [Clarke et al. 1986]. Without an accurate environment model, unreachable states and unrealizable state transitions may be wrongly incorporated into the verification, or conversely, reachable states and realizable transitions deleted. Both can lead to false model-checking results. The recent work of Martin et al. [1998] discusses this problem, although their focus is on efficient methods for representing abstracted trace behavior, while our focus here is on automatically computing an accurate abstraction.

In this paper, we formalize the environment-modeling problem in terms of language equivalence between concrete and abstract FSMs. We focus on the subproblem of determining when the *safe removal* of a FSM is possible from a FSM network. Safe removal means the substitution, preserving the desired language equivalence for environment modeling, of a set of free inputs driving a FSM's fanout for that FSM. Safe removal is possible only if a FSM is a *universal* FSM (UFSM), a FSM that can realize all possible sequences of all possible output valuations. We offer an algorithm for UFSM identification and describe how it can be used to determine FSM *input-independence*. When FSMs are independent of those of their inputs connected to outputs of neighboring FSMs, and are UFSMs, then their safe removal becomes possible. We also discuss the use of *simulation relations* for reducing the state space of a FSM. Such reduction, performed prior to UFSM checking, lowers its computational cost and also creates a smaller FSM which can serve as an environment model if safe removal is not possible.

At first, it may seem unlikely that UFSMs would appear very often in real designs. A UFSM, however, is not a machine that realizes all output

sequences regardless of input. Such a machine would seldom serve a useful purpose. Rather, a UFSM is a machine that can realize all output sequences, given appropriate input. Often, a trace accurate model for a component FSM in a network, from the "viewpoint" of a neighboring FSM, might be a trivial, single-state UFSM able to nondeterministically assert a signal, or not, on every time step, while executing a self-loop in its initial state. This would be the case if the signal assertion were independent of any feedback that may exist between the two FSMs, and if it could occur on any time step. Our work lays the theoretical foundation for computing such an abstraction by examining a real, concrete FSM network. At present, in VLSI design, such abstractions are made by hand, in a time-consuming and error-prone manner. It is clear that automation would be very useful and most welcome, and this is our goal.

UFSMs have been cited previously in the literature, but only as artifacts for computing worst-case complexities of certain problems in automata theory [Sistla et al. 1987; Meyer and Stockmeyer 1972]. We believe we are the first to give algorithms for their identification and to claim that their identification is of practical value. The problem of UFSM identification is shown to be PSPACE-complete in Meyer and Stockmeyer [1972]. A slightly different version of the UFSM-detection algorithm of Section 4.4 is given by two of the present authors in Raimi and Hojati [1997]. Our algorithm is based on the classical subset construction algorithm described in Hopcroft and Ullman [1979] for determinizing a nondeterministic finite automaton (NFA). We, however, add an optimization that ensures that the full powerset of a set of states will never be manipulated.

Simulation relations, which we also utilize, were introduced in Milner [1971]. Our use of these for state reduction follows that of Bouaijani et al. [1990]; Fernandez et al. [1993]; Hojati [1996], and others. The theory behind this part of our work is not original; but the application is. In addition, the work of Watanabe and Brayton [1994] and Wang and Brayton [1993] bears a strong resemblance to ours, in that constraints on the trace behaviors of FSMs in FSM networks are computed. However, the goal of their research is to synthesize a concrete FSM rather than to abstract it. In addition, the methods we use, simulation relations combined with UFSM detection, are quite different from those they propose.

This paper is organized as follows. In Section 2, we outline a methodology for automated environment modeling. In Sections 3 and 4, we explain our basic formalisms, define safe FSM removal, and describe a UFSM-detection algorithm needed for its determination. In Section 5, we prove the correctness of the algorithm. In Section 6, we discuss state reduction using simulation relations. In Section 7, we prove that if a concrete FSM network is deadlock-free, our abstraction techniques do not introduce deadlock. We conclude, in Section 8, with future research goals.

2. METHODOLOGY FOR ENVIRONMENT MODELING

A set of interconnected sequential circuits may be represented as a network of interacting FSMs. Given such a network, the methodology we propose for environment modeling is to

(1) identify the *target* FSM, about which we desire to prove properties, identifying, thereby, its concrete environment;

(2) partition the environment into small, component FSMs;

(3) for each component FSM,
   —compute its reachable state set and then reduce it using simulation relations;
   —remove the FSM, if possible, after checking language universality.

Algorithms for partitioning are part of our future research effort, which we discuss in Section 8.

In practice, some abstraction would likely be performed on a FSM network before the techniques we describe here would be applied. Digital hardware can be thought of as comprising three types of circuitry: control, datapath, and memory. Our abstraction methods are geared towards control logic: the sequential circuitry that determines when datapath operations are carried out, upon which operands, and where and when the results are stored in memory. Datapath and memory circuitry require very different abstraction techniques, and we assume such abstractions are carried out beforehand. Various semiautomated [Kurshan 1989; Long 1993] or automated [Hojati 1996] techniques have been described in the literature. The problems of datapath and memory abstraction are by no means solved; but abstraction of control logic, the focus of our efforts, is also very much needed. We feel the general environment-modeling problem will ultimately be solved, or alleviated, by bringing together hybrid approaches, each applicable in different areas.

Portions of real environments may, at times, also be modeled with hand-created finite state models of protocols they implement, rather than with FSM models of specific circuits. For example, a bus arbiter in a multiprocessor system might be modeled by its protocol rather than by a specific circuit, thereby making a verification more general. Such a protocol model could, once incorporated into the general environment, then be further reduced by the techniques we outline here.

3. DEFINITIONS

We use a formalism for a finite state machine (FSM) which we feel is appropriate for modeling digital hardware.

Definition 1 (FSM). A FSM is a 5-tuple:

$$E = \langle S, S^0, I, O, T \rangle$$

where S is a finite set of states, $S^0 \subseteq S$ is a set of initial states, I and O are finite disjoint sets of binary input and output variables, and $T(s, i, o, t) \subseteq S \times \Sigma_I \times \Sigma_O \times S$ is the transition relation. $\Sigma_I = \{0, 1\}^{|I|}$ and $\Sigma_O = \{0, 1\}^{|O|}$ are the input and output alphabets. We often write either $T(s, i, o, t)$ or $s \xrightarrow{i/o}_T t$ in place of $(s, i, o, t) \in T$.

Definition 2 (Deadlock). A *deadlock-free* FSM is a FSM where every reachable state has a defined successor state and output valuation on each input $i \in \Sigma_I$.

Note that this corresponds to the behavior of digital sequential circuits, which always *do* respond to their inputs.[1]

FSM interactions are modeled by set intersection, which we term an *overlap* among sets of input and output variables. Thus, a connection between two FSMs, A and B, such that (some of) the outputs of the A drive and (some of) the inputs of B are denoted $O_A \cap I_B \neq \emptyset$, where O_A and I_B are the outputs of A and the inputs of B, respectively.

3.1 FSM Composition

Consider FSMs $E = \langle S, S^0, I, O, T\rangle$ and $M = \langle S', S'^0, I', O', T'\rangle$. The composition of E and M, denoted $E \parallel M$, can be defined by a partition $I = I_0 \cup I_1$ of the inputs of the first machine E and a partition $O' = O'_0 \cup O'_1$ of the outputs of the second machine M, such that $I_1 = O'_1$ and $O = I'$. Additionally, I_0 must be disjoint from O'. The new FSM, $E \parallel M$, can be defined as follows:

—set of states, $S \times S'$;

—input variables, I_0,

—output variables $O \cup O'$,

—initial states, $S^0 \times S'^0$,

—transition relation $T_{\parallel}$, where

$$(s, u) \xrightarrow{i/(k, m, j)}_{T_{\parallel}} (t, v) \equiv s \xrightarrow{(i, j)/k}_T t \wedge u \xrightarrow{k/(m, j)}_T v,$$

where a tuple, for example, (m, j), indicates the partition of M's outputs into $O'_0 \cup O'_1$; see Figure 1 for an illustration.

The above definition does not refer to inputs of M that do not overlap with outputs of E or to outputs of E that do not overlap with inputs of M.

[1]In actuality, digital hardware is deadlock-free even in unreachable states. However, we restrict our attention to the reachable states because this simplifies some of our proofs, such as those in Section 7.

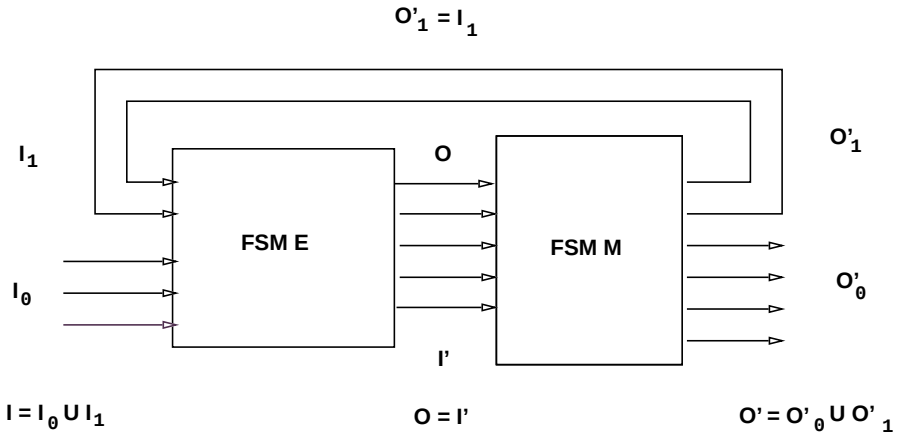

Fig. 1. FSM composition $E \parallel M$.

These were deleted for brevity. However, when they are included, the definition of composition becomes symmetrical.

We assume that the concrete FSM networks we abstract are deadlock-free. This is in no way a limitation, as this accurately models digital hardware. There is a problem, though, in that the class of deadlock-free FSMs is not closed under composition, i.e., it is possible that the composition of two deadlock-free FSMs may exhibit deadlock.[2] Therefore, we prove that our abstraction methods do not introduce deadlock (in Section 7, Theorems 5 and 6).

3.2 Free and Constrained Inputs

Definition 3 (Input types). Given a network of interacting FSMs, i.e., a composite FSM:

(1) A *free input* of a component FSM is one that has no overlap with any output of any other component FSM.

(2) A *constrained input* is one that does have such overlap.

Herein, we assume that the set of inputs I of a FSM E is divided into disjoint sets of free and constrained inputs $I = I_c \cup I_f$, such that $I_c \cap I_f = \emptyset$. I_c and I_f are, respectively, the constrained and free inputs.

---

[2]As an example, consider the FSMs M and N defined below:

(1) M has a single state s, single input and single output variables, and transitions $s \xrightarrow{0/0}_M s$, $s \xrightarrow{1/1}_M s$;

(2) N has a single state t, single input and single output variables, and transitions $t \xrightarrow{0/1}_N t$, $t \xrightarrow{1/0}_N t$.

It is easy to check that M and N are deadlock-free. In the composition $M \parallel N$, however, where the input of M is the output of N and vice-versa, there is no transition out of the initial state (s, t), since the input-output pairs do not match. Thus, $M \parallel N$ exhibits deadlock.

We also assume that FSM networks are partitioned such that the outputs and free inputs of each component FSM are disjoint from like sets of other component FSMs. This restriction may be removed in practice; however, in this paper it allows us to state theorems without resort to arguments about special cases.

3.3 FSM Languages

Definition 4 (Path). A *path* through a FSM E is a finite or infinite sequence of states $s_0, s_1, \cdots, s_j, \cdots$, such that $\exists i_j \exists o_j [T(s_j, i_j, o_j, s_{j+1})]$ holds for all j.

Definition 5 (Run). Given a finite or infinite sequence τ of pairs (i_j, o_j) from the input and output alphabets, there is a *run* of τ in E iff there is a path s in E such that $s_0 \in S^0$, and for all j, $T(s_j, i_j, o_j, s_{j+1})$ holds.

Definition 6 (Projection). Given a finite or infinite sequence τ, defined over valuations of a set of variables A, the *projection* of τ onto $A' \subset A$, written $\tau_{|A'}$, is the sequence formed by omitting the valuation of variables not in A' along τ.

We say a finite or infinite sequence τ is *realized* by a FSM E when τ either has a *run* in E or is a *projection* of a sequence that has a run. We stipulate when sequences are finite; otherwise, they are assumed to be infinite.

Given a set of variables A and an associated alphabet $\Sigma_A = \{0, 1\}^{|A|}$, Σ_A^ω denotes the set of all possible infinite sequences of elements in Σ_A, and $\Sigma_A^\star$, the set of all possible finite sequences.

Definition 7 (FSM language). The *language* of a FSM E, denoted $\mathcal{L}(E)$, is the set of all infinite input-output sequences that have a run in E.

We use the term *trace* to mean an input-output sequence of a FSM, and use the terms *trace equivalence* and *language equivalence* synonymously.

Definition 8 (Output language). The *output language*, $\mathcal{L}_{|O}(E)$, of a FSM E is the projection of $\mathcal{L}(E)$ onto the set of E's output variables O.

Definition 9 (UFSM). A FSM E is a *universal finite state machine* (UFSM) iff its output language is Σ_O^ω.

3.4 The Environment-Modeling Problem

Given the composition of two FSMs, $E \parallel M$, the *environment modeling problem* is that of finding a FSM E_M much smaller than E in terms of number of states, such that

$$\mathcal{L}_{|I' \times O'}(E \parallel M) = \mathcal{L}_{|I' \times O'}(E_M \parallel M)$$

where I' and O' are the input and output variables of M, respectively.

Intuitively, consider M the *target* FSM, E its concrete environment, and E_M its environment model. All three may, themselves, be composite FSMs.

4. SAFE REMOVAL AND UFSMS

4.1 FSM Removal

Definition 10 (FSM removal). A FSM E is *removed* from a network of interacting FSMs as follows:

(1) for every input i of a neighboring FSM M, if i overlaps with an output of E, replace i with a newly created free input i', and substitute i' for i in the transition relation of M;

(2) remove any output o' from any neighboring FSM M' if o' overlaps with inputs of E, and with no other FSM in the network;

(3) remove any FSM M'' by the same process, if removal of E results in M'' have an empty set of outputs.

Definition 11 (Safe removal). Given a network of interacting FSMs N and a set of input or output variables Q of component FSMs in the network, component FSM E is *safe to remove*, relative to Q, if removal does not change $\mathcal{L}_{|_Q}(N)$, the language of the network, projected onto Q.

The language of a FSM network, $\mathcal{L}(N)$, is the language of the composition of all FSMs in N.

We assume that any FSM being removed is not the *target* FSM of the verification. Thus, the recursive removal process outlined in step 3 would halt if the FSM being removed were the target FSM.

4.2 Input Independence

Safe removal of a FSM is predicated upon *input independence* from its constrained inputs, if any. Intuitively, a FSM has *input independence* if each sequence in its output language is realizable regardless of the valuation of its constrained inputs.

Definition 12 (Input independence). A FSM E has the property of *input independence* from a subset of its inputs I_c, iff for all $\tau_o \in \mathcal{L}_{|_O}(E)$ and for all $\tau_{ic} \in \Sigma^{\omega}_{I_c}$, where $\tau_o = o_0, o_1, \cdots, o_j, o_{j+1}, \cdots$ and $\tau_{ic} = ic_0, ic_1, \cdots, ic_j, ic_{j+1}, \cdots$, the combined sequence $\tau = (ic_0, o_0), (ic_1, o_1), \cdots, (ic_j, o_j), (ic_{j+1}, o_{j+1}), \cdots$, is in $\mathcal{L}_{|_{I_c \times O}}(E)$.

Input independence can be framed as a language-containment problem. Consider FSM E with disjoint constrained and free inputs, I_c and I_f. Consider FSMs E_1 and E_2, derived from E, as follows:

$$T_1(s, i_c, o, t) = \exists i_f [T(s, (i_c, i_f), o, t)]$$

$$T_2(s, i_c, o, t) = \exists i_f \exists i_c [T(s, (i_c, i_f), o, t)]$$

$$E_1 = \langle S, S^0, I_c, O, T_1 \rangle$$

$$E_2 = \langle S, S^0, I_c, O, T_2 \rangle$$

where $i_c \in I_c$ and $i_f \in I_f$. Note that the languages of E_1 and E_2 are defined over $\Sigma_{I_c \times O}$.

THEOREM 1. *E has input independence from I_c iff $\mathcal{L}(E_1) \supseteq \mathcal{L}(E_2)$.*

PROOF. $\mathcal{L}(E_1)$ is easily seen to be $\mathcal{L}_{|_{I_c \times O}}(E)$. The input-independence criterion can now be stated as $\Sigma^{\omega}_{I_c} \times \mathcal{L}_O(E) \subseteq \mathcal{L}_{|_{I_c \times O}}(E)$. Hence, we need to show that $\mathcal{L}(E_2)$ is $\Sigma^{\omega}_{I_c} \times \mathcal{L}_O(E)$ in order to complete the proof.

Consider any pair (σ, δ) in $\Sigma^{\omega}_{I_c} \times \mathcal{L}_O(E)$. By definition, δ is a realizable output sequence for E. Thus, there is an input sequence γ such that (γ, δ) has a run in E. The definition of T_2 adds an edge for each element of Σ_{I_c} between any consecutive states in this run. Hence, (σ, δ) has a run in E_2.

In the other direction, consider any pair (σ, δ) that has a run in E_2. From the definition of T_2, the quantified-out inputs on each transition of this run define an input sequence γ such that (γ, δ) has a run in E. Hence, δ is in $\mathcal{L}_O(E)$, as desired. □

The intuition behind the proof is that E_1's traces are restricted by its constrained inputs, while E_2's are not. If, in spite of this, we have $\mathcal{L}(E_1) \supseteq \mathcal{L}(E_2)$, then E_1's output traces must be independent of its constrained inputs.

4.3 CUFSMs: "Controlled" UFSMs

Component FSMs in a FSM network will, in general, be in complicated feedback and feedforward relationships to each other. Safe removal is possible for such FSMs if they are *CUFSMs*.

Definition 13 (CUFSM). A FSM E is a *controlled universal finite state machine (CUFSM)* iff (a) E has input independence from its constrained inputs I_c, and (b) E is a UFSM.

A CUFSM, then, is "controlled" by its free inputs.

THEOREM 2. *If E is a component in a network of interacting FSMs N, and is a CUFSM, then E is safe to remove from N.*

PROOF. Safe FSM removal is relative to a set Q of input or output variables of component FSMs in a FSM network. Let us choose Q to be the set union of the inputs and outputs of E. By Definition 11, removal of E may entail (a) substituting free inputs for inputs of other FSMs that overlap with its outputs; and (b) eliminating overlap between outputs of

neighboring FSMs and inputs of E. With respect to (a), the set of possible traces over inputs of neighboring FSMs driven by outputs of E can only increase, not decrease, when E is removed. Since E is assumed to be a UFSM, however, no increase is possible.

Considering case (b), the overlap of neighboring FSM outputs with inputs of E can only constrain E's inputs. Since E is assumed to be a CUFSM, E can realize all pairings of constrained input sequences and output sequences. Therefore, E's removal could not change sequences over its outputs, and thus not over Q.

We have shown that trace equivalence is preserved over any inputs of neighboring FSMs with E's removal. By the definition of a FSM, the trace set at its inputs determines the trace set it can realize at its outputs. Thus, given that removal of E cannot alter sequences over Q when Q is the set of input and output variables of E, removal of E cannot alter sequences over the union of Q with any other input or output variables of any component FSM in the network. □

THEOREM 3. *E is a CUFSM iff it is universal over the alphabet $\Sigma_{I_c \times O}$.*

PROOF. From the definition, E is a CUFSM iff (a) it has the property of input independence from its constrained inputs, which condition is equivalent to $\Sigma_{I_c}^{\omega} \times \mathcal{L}_O(E) \subseteq \mathcal{L}_{|_{I_c \times O}}(E)$; and (b) it is a UFSM, i.e., $\mathcal{L}_O(E) = \Sigma_O^{\omega}$.

Hence, if E is a CUFSM, then $\mathcal{L}_{|_{I_c \times O}}(E) = \Sigma_{I_c}^{\omega} \times \Sigma_O^{\omega}$, so E is universal over $\Sigma_{I_c \times O}$. The proof in the other direction is trivial. □

4.4 UFSM Detection

We present our UFSM-identification algorithm in Figure 2. In the algorithm, the underlying FSM E is as defined in Section 3. The algorithm is based on the classical subset construction algorithm described in Hopcroft and Ullman [1979], for creating a DFA (deterministic finite automaton) from a NFA (nondeterministic finite automaton). Out algorithm differs, however, in that a set of states is discarded if it is a superset of another set already encountered. In this lies our optimization, as this insures we will never enumerate the entire powerset of E, something which could happen in the worst case in the classical subset construction.

As the algorithm runs, *DFA_MetaStates* is populated with all incomparable sets of E's states from which partial images have been taken, while *NFA_Statesets* consists of sets of states held for future partial image generation. The *partial image under output valuation* $o \in \Sigma_O$ of a set of states $\mathcal{Q} \in S$ is

$$Image(t) = \exists s \exists i [T(s, i, o, t) \wedge \mathcal{Q}(s)]$$

where o is a fixed valuation.

Figure 2 (lower part) illustrates the algorithm. On the left, the state transition graphs of two FSMs are depicted, each having a single, binary

```
1  NFA_Statesets := {S^O};
2  DFA_MetaStates := ∅;
3  While (NFA_Statesets ≠ ∅)
     begin
4      Pick any Q ∈ NFA_Statesets;
5      Remove Q from NFA_Statesets;
6      Add Q to DFA_MetaStates;
7      Foreach o ∈ Σ_O:
         begin  // Compute Partial Image of Q under o;
8          Image_o(t) = ∃ s ∃ i [T(s, i, o, t) ∧ Q(s)];
9          If (Image_o = ∅)
10           Exit; (Not a UFSM)
11         If ( Forall X ∈ DFA_MetaStates ∪ NFA_Statesets, Image_o ⊇ X )
             begin
12             Add Image_o to NFA_Statesets;
13             Remove any X' ∈ NFA_Statesets ∪ DFA_MetaStates such that X' ⊃ Image_o;
             end
         end
     end
```

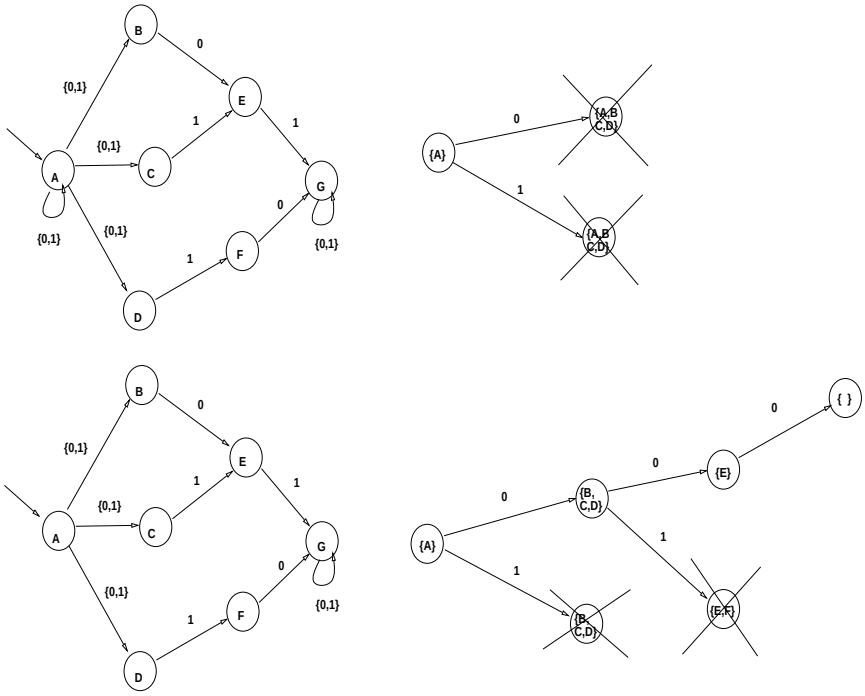

Fig. 2. UFSM Identification: Algorithm and illustration.

output. They differ only in the existence of a self-loop in the initial state. The upper machine is a UFSM, the lower one is not. To the right of each FSM's state graph, the workings of the algorithm are depicted as another graph. Nodes are labeled with the metastates (i.e., sets of states encountered as the algorithm runs) and arcs are labeled with the output symbols by which these metastates are reached. A cross through a node indicates a metastate that is discarded as a superset of another. The algorithm ends with success for the upper FSM after taking images from just the initial state, since each partial image is a superset of the singleton initial state

set. The algorithm ends with rejection for the lower FSM when an empty partial image is encountered.

The algorithm in Figure 2 can be altered to check if E is universal over $\Sigma_{I_c \times O}$, and thus, by Theorem 3, is a CUFSM. The loop beginning on line 7 would iterate over all i_c and o in $\Sigma_{I_c \times O}$, and the *partial image* operation on line 8 would then become

$$Image(t) = \exists s \exists i_f [T(s, (i_c, i_f), o, t) \wedge \mathcal{Q}(s)]$$

where $o \in \Sigma_O$ and $i_c \in \Sigma_{I_c}$ are fixed and $i_f \in \Sigma_{I_f}$ is a variable.

Theorem 1 of Section 4.2 provides a means of determining input independence via language containment. Simulation relations, discussed in Section 6, provide a polynomial time approximation to language containment. If by using simulation relations one can determine that a FSM has input independence from its constrained inputs, then one can run the UFSM-detection algorithm of Figure 2 over the outputs of the FSM only, as opposed to outputs plus constrained inputs. This could considerably reduce the computational resources needed.

5. CORRECTNESS PROOF FOR UFSM-IDENTIFICATION ALGORITHM

We prove here that the UFSM-identification algorithm of Figure 2 correctly detects whether a FSM is a UFSM or not. For brevity, we omit a necessary proof that the procedure terminates. Informally, the termination condition is that *NFA_Statesets* must eventually become empty, since the powerset of a FSM is finite.

Below, we consider a FSM E, defined over an alphabet of action symbols Σ_O, where $\Sigma_O^\star$ signifies the set of all finite sequences over Σ_O and Σ_O^ω the set of all infinite sequences over Σ_O. We first describe how the UFSM-detection algorithm of Figure 2 builds, implicitly, a DFA derived from E.

The FSM E will, in general, become a NFA in the algorithm, since on line 8 all inputs are quantified away. Let $\mathcal{D}_{\mathcal{E}}$ be the implicit DFA derived from E by the algorithm. States of $\mathcal{D}_{\mathcal{E}}$ are sets of states of E, and we refer to these as *metastates*, to differentiate them from single states of E. The initial metastate of $\mathcal{D}_{\mathcal{E}}$ is the set of initial states of E, which, on line 1 of the algorithm, is put into *NFA_Statesets*. As the algorithm runs, the metastates of $\mathcal{D}_{\mathcal{E}}$ are all the members *NFA_Statesets* $\cup$ *DFA_MetaStates*, which set union, for convenience, we call S_D. New metastates and transitions are added to S_D in the loop starting at line 7, in which partial images are taken. Consider an arbitrary partial image Q', generated from metastate Q on action symbol o, on line 8. If Q' is not a superset of any metastate in S_D, the algorithm behaves as the classical subset construction algorithm described in Hopcroft and Ullman [1979]. Q' becomes a metastate in $\mathcal{D}_{\mathcal{E}}$ (it is put into *NFA_Statesets*), and a transition is added to $\mathcal{D}_{\mathcal{E}}$ from Q to Q' on action symbol o. Generation of Q', however,

may result in discarding of a set of states, which is a departure from the classical algorithm. Q' itself may be discarded if it is a superset of an existing metastate $D \in S_D$. In this case, a transition is created in E from Q to D, on action symbol o. It is also possible that some existing $D' \in S_D$ is discarded (line 13) if $D' \supset Q'$. In this case, Q' is kept, a transition from Q to Q' on action symbol o is added, and, additionally, existing transitions going into D' are routed into Q'. The loop beginning at line 7 then has four possible outcomes for each partial image Q' that is generated:

(1) Q' is kept, nothing is discarded,

(2) Q' is kept, another metastate D' is discarded,

(3) Q' is discarded, or

(4) Q' is empty, and the algorithm exits on line 10.

In the last case, Q will not have a transition on at least one $o \in \Sigma_O$, and all metastates that may have been in *NFA_Statesets* at the time of exit will be dead-end metastates, i.e., they will have no outgoing transitions at all.

Let $dfa(E)$ be the DFA derived from E by the classical determination construction. We prove, below, that $dfa(E)$ can realize all finite sequences that $\mathcal{D}_{\mathcal{E}}$ can realize, and therefore that the language of $dfa(E)$ is a superset of that of $\mathcal{D}_{\mathcal{E}}$. We prove that when the algorithm exits on line 3 the language of $\mathcal{D}_{\mathcal{E}}$ is $\Sigma_O^\star$. We exploit the known fact, proven in Hopcroft and Ullman [1979] and elsewhere, that the languages of $dfa(E)$ and E are the same, to infer that the language of E is $\Sigma_O^\star$. We then extend this to a proof that E's language is Σ_O^ω, and E is, therefore, a UFSM.

We also prove that if the algorithm is exited on line 10, E is not a UFSM. We cannot use a language containment argument in this case, since the language of $dfa(E)$ could still be $\Sigma_O^\star$ even when the language of $\mathcal{D}_{\mathcal{E}}$ is not. We, therefore, prove directly that when the algorithm is exited on line 10, $dfa(E)$'s language is not $\Sigma_O^\star$, and thus E cannot be a UFSM. Exiting on lines 3 or 10 are the only two possibilities for the algorithm.

PROOF. We construct the proof using the following lemmas.

LEMMA 1. *All metastates in $\mathcal{D}_{\mathcal{E}}$ have outgoing transitions on all $o \in \Sigma_O$ iff the algorithm is exited on line 3.*

PROOF. *NFA_Statesets* is initialized on line 1 to hold the initial state set of E. The algorithm is exited on line 3 if *NFA_Statesets* is empty. There must, then, be at least one pass through the main while loop starting at line 3. *DFA_MetaStates* is initialized as empty (line 2).

By the construction of $\mathcal{D}_{\mathcal{E}}$ recounted above, metastates in *NFA_Statesets* have no outgoing transitions at all. In each pass through the main while loop of line 3, a new member Q of *NFA_Statesets* is chosen

for partial image generation. By the construction of $\mathcal{D}_{\mathcal{E}}$, each iteration through the main while loop either discards such a Q or adds outgoing edges to it for all $o \in \Sigma_O$ and transfers it to *DFA_MetaStates*, from which it can never return to *NFA_Statesets* (this can be verified by inspection). It is the termination condition of the algorithm that *NFA_Statesets* must eventually become empty, since the powerset of E is finite. When it does become empty, all former members of *NFA_Statesets* will either have been discarded or have had outgoing transitions added on all $o \in \Sigma_O$ and be in *DFA_MetaStates*. By inspection of the algorithm, it can also be verified that metastates come into *DFA_MetaStates* only by transfer from *NFA_Statesets*.

An existing metastate may be discarded from *DFA_MetaStates* (line 13) on each pass through the main while loop; but, as detailed above, no outgoing transitions of any remaining metastates are deleted, but are simply rerouted. Thus, if the algorithm is exited on line 3, all metastates in $\mathcal{D}_{\mathcal{E}}$ have transitions on all $o \in \Sigma_O$.

For the proof in the other direction, if the algorithm is not exited on line 3 it must be exited on line 10. This occurs only when there is no transition on some $o \in \Sigma_O$ from the metastate Q chosen for partial image generation. □

LEMMA 2. *For every finite sequence $\tau_n \in \Sigma_O^\star$, if τ_n is realized in $\mathcal{D}_{\mathcal{E}}$ it is realized in $dfa(E)$, and the terminal metastate X of the state sequence realizing it in $\mathcal{D}_{\mathcal{E}}$ is a subset of the terminal metastate R of the state sequence realizing it in $dfa(E)$.*

PROOF. This is trivially true for sequences of length 0, since the initial states of $\mathcal{D}_{\mathcal{E}}$ and $dfa(E)$ are the same.

Assume the claim is true for all sequences $\tau_n \in \Sigma_O^\star$ of length n. Let us consider an arbitrary τ_n that $\mathcal{D}_{\mathcal{E}}$ can realize. Assume the terminal metastates of the state sequences realizing it in $\mathcal{D}_{\mathcal{E}}$ and $dfa(E)$ are X and R, respectively. Consider any sequence $\tau_{n+1} = \tau_n;o$ for any $o \in \Sigma_O$, and assume $\mathcal{D}_{\mathcal{E}}$ can realize τ_{n+1}. Then there is a metastate X' that is a successor to X on o in $\mathcal{D}_{\mathcal{E}}$. Now there are three ways in which X' could have become the successor to X on o when $\mathcal{D}_{\mathcal{E}}$ was constructed. Below, let $Image_o(Q)$ denote the partial image under o of set of states Q.

(1) $X' = Image_o(X)$, i.e., X' was generated on line 8, and the transition X to X' on o was added without any set of states being discarded;

(2) $Image_o(X)$ was discarded on line 11 of the algorithm, and X', an existing metastate of $\mathcal{D}_{\mathcal{E}}$, became the successor of X on o;

(3) on some iteration of the algorithm, X was in *DFA_MetaStates* with successor Z on action symbol o, X' was generated as the partial image of some metastate Y on some action symbol (it doesn't matter which), Z

was discarded (line 13) as a superset of X', and the transition X to X' on o was added.

In cases 2 and 3, by the construction of $\mathcal{D}_{\mathcal{E}}$, detailed above, $X' \subseteq Image_o(X)$. Since by the induction hypothesis, $X \subseteq R$, $Image_o(X) \subseteq Image_o(R)$, and thus $X' \subseteq Image_o(R)$. In the classical subset construction algorithm, $R' = Image_o(R)$, and so $X' \subseteq R'$. Since X' is not empty, neither is $R' = Image(R)$, and therefore R', by the classical subset construction, is a metastate in $dfa(E)$ and $dfa(E)$ can realize τ_{n+1}. □

LEMMA 3. *A DFA defined over a set of action symbols Σ_O can realize all sequences in $\Sigma_O^\star$ iff in every state there is a transition on every $o \in \Sigma_O$.*

PROOF. Let us assume the above is false for some DFA D. Then D can either realize all of $\Sigma_O^\star$, but cannot transition in some state s on some $o \in \Sigma_O$, or D can transition in every state s on every $o \in \Sigma_O$, but cannot realize some sequence in $\Sigma_O^\star$. Considering the first case, assume D can realize finite sequence τ. Being a DFA, there is only one sequence of states realizing τ ending in, say, state s. But if D cannot transition on o from s, then $\tau;o$ cannot be generated, and so D cannot realize all of $\Sigma_O^\star$. Regarding the second case, assume D cannot realize the sequence of length n, τ_n. There must be some longest finite prefix τ_j where $0 \leq j < n$ that D can realize. Being a DFA, there is only one sequence of states that realizes τ_j, and one state s terminating this sequence. Since s can transition on each $o \in \Sigma_O$, the DFA must be able to realize each $\tau_j;o$, and thus τ_j cannot be the longest finite prefix of τ_n. □

We need the lemma below to extend our proofs on finite sequences to infinite ones.

LEMMA 4. *For any FSM E, if E can produce all sequences in $\Sigma_O^\star$, then E can produce all sequences in Σ_O^ω.*

PROOF. Let E be a FSM that can produce all sequences in $\Sigma_O^\star$.

Let τ be an infinite sequence from Σ_O^ω. By the hypothesis, E can produce all finite prefixes of τ. For each integer i there is a sequence of alternating states and transitions through E that produces the finite prefix of τ of length i, τ_i. We can arrange all such sequences for all τ_i in the form of a directed acyclic graph, where the nodes are states and the arcs are transitions labeled with symbols from Σ_O. This graph is rooted at the possibly multiple initial states of E from which the infinite sequence τ, can be realized. Counting the root as the 1^{st} level, nodes at the i^{th} level of the graph represent states reachable on τ_{i-1}, the finite prefix of τ of length $i - 1$. From each of these states, we construct the graph as follows. We extend arcs labeled with the i^{th} symbol in τ, o_i, from each such state s to states reachable from s on o_i. We create two such arcs for each transition.

From one of the (duplicate) end nodes on these arcs, we repeat this pattern and extend the graph further, and from the other we do not, leaving it as a dead-end node. All such dead-end nodes at the i^{th} level of the graph form the termination points for all sequences of length $i - 1$ that realize the finite prefix of τ, τ_{i-1}.

E is finite-state, and, as such, each level of the graph will have finitely many states. Thus, this graph has finite branching. Since successively longer prefixes of τ are produced, the graph is infinite. By König's Lemma [König 1936], the graph must contain an infinite path τ_ω. However, by construction, the path τ_ω agrees with every finite prefix of τ, and hence must equal τ. Thus, τ is produced by E as well. □

LEMMA 5. *If the algorithm exits on line 10, the language of $dfa(E)$ is a proper subset of $\Sigma_O^\star$.*

PROOF. The classical subset construction algorithm, described in Hopcroft and Ullman [1979] and elsewhere, creates a metastate for each subset of the reachable states of the underlying NFA. By the construction of $\mathcal{D}_{\mathcal{E}}$ described above, any metastate Q from which partial images are generated on line 8 is a subset of the reachable states of E and is, therefore, in both $\mathcal{D}_{\mathcal{E}}$ and $dfa(E)$. If the algorithm exits on line 10, Q does not have a transition on some $o \in \Sigma_O$, and by Lemma 3, the language of $dfa(E)$ must be a proper subset of $\Sigma_O^\star$. □

We can now easily form our proof. Recall that lines 3 and 10 are the only two exit lines in the algorithm.

By Lemma 1, if the algorithm exits on line 3, the implicit DFA, $\mathcal{D}_{\mathcal{E}}$, has a transition on each $o \in \Sigma_O$ in each of its metastates. By Lemma 3, $\mathcal{D}_{\mathcal{E}}$ can realize all sequences in $\Sigma_O^\star$ and, by Lemma 2, so can $dfa(E)$. By the known fact that the languages of $dfa(E)$ and E are the same, so can E. By Lemma 4, E is therefore a UFSM.

If the algorithm exits on line 10, by Lemma 5 there is at least one sequence in $\Sigma_O^\star$ which $dfa(E)$ cannot realize, and by Lemma 4, E is therefore not a UFSM. □

6. SIMULATION RELATIONS

A simulation relation is a relation over a set of states and a set of *action symbols* associated with the states or with state transitions. When a pair of states (s, t) are related, the meaning is that all sequences of action symbols realizable from state s are realizable from state t, and additional sequences may be realizable from t as well. We wish to compute simulation relations over the states of a FSM, E, with the set of action symbols being the alphabet $\Sigma_{I_c \times O}$, as defined above. We then create a new abstract FSM E' by choosing representative states from the state pairs related by this simulation relation. This type of state reduction can be both a prelude to

UFSM detection, which will be easier to carry out on a reduced state space, or an end in itself. That is, if E is not a UFSM, then E' may still be a reduced replacement for it. Computation of a simulation relation is polynomial in the number of states, and the UFSM-detection algorithm in Section 4.4 is exponential in the number of states. This greatly motivates our use of simulation relations.

6.1 Simulation, Simulation Equivalence, and Bisimulation

A simulation relation R may be computed over the state set S of a FSM E, as below, where $T(s, i, o, t)$ is the transition relation of E:

$$R^0(s, t) := S \times S$$

$$R^{i+1}(s, t) := R^i(s, t) \wedge \forall i \forall o \forall s' [T(s, i, o, s') \rightarrow (\exists t' T(t, i, o, t') \wedge R^i(s', t'))]$$

A *simulation equivalence relation* R_{se} may then be formed from R,

$$R_{se}(s, t) := R(s, t) \wedge R(t, s)$$

and a state reduced transition relation, T', computed:

$$T'(s, i, o, t) := Rep(s) \wedge Rep(t) \wedge \exists u, v [R_{se}(u, s) \wedge R_{se}(v, t) \wedge T(u, i, o, v)] \quad (1)$$

In practice, the above computations would be performed after reachability analysis, as convergence is likely to occur much quicker when the computations are confined to reachable states.

The simulation equivalence relation R_{se} forms a symmetrical relation by deleting, after convergence, all $(s, t) \in R$ such that $(t, s) \notin R$. We could delete such assymetrical pairs on every iteration of the algorithm, in which case R would become a *bisimulation relation*. In both cases (bisimulation and simulation equivalence), related pairs are trace-equivalent; however, we choose to utilize the simulation-equivalence relation because it is weaker, and may, therefore, include more trace-equivalent state pairs.

6.2 State Reduction

Simulation equivalence defines a partition of a state set into equivalence classes. We may choose lexicographically least elements from each equivalence class to represent the class in a state-reduced FSM. This may be done using ROBDDs (reduced ordered binary decision diagrams), as per Lin and Newton [1991]. Let $R_{se}(s, t)$ be the characteristic function of the simulation equivalence relation, and $Rep(t)$ the characteristic function of the set of representative states. We may then form a new transition relation T' from the original transition relation E, as per formula 1 of Section 6.1. In T', transitions are only among representative states. It is a theorem that this computation preserves trace-equivalence. In stating and proving this theorem, we refer to state reduction via computation of a simulation relation as *SRSR*.

THEOREM 4. *If a FSM E' is derived from another FSM E, via SRSR computed with respect to a set of action symbols A, then $\mathcal{L}_{|_A}(E) = \mathcal{L}_{|_A}(E')$.*

PROOF. E can realize some set of infinite sequences, $\Sigma_E^\omega \subseteq \Sigma_A^\omega$. To make the proof most relevant, we write an element of a sequence as (i_c, o), where i_c is a valuation of the constrained inputs of E. By Lemma 4 of Section 5, if E' can realize all finite prefixes of each $\tau \in \Sigma_E^\omega$, then E' can realize all $\tau \in \Sigma_E^\omega$. We show, by induction, that if E' is derived from E via SRSR, this must be the case.

The induction hypothesis is that E' can realize all finite prefixes of length n of each $\tau \in \Sigma_E^\omega$. For proving the base case, assume E' cannot realize some finite prefix of length 1, (i_c, o), but E can. We show that this is impossible. If E can realize (i_c, o), there must be an initial state E, u, such that u transitions on (i_c, o) to some state v. States u and v have representatives, call them s and t, and therefore $Rep(s)$ and $Rep(t)$ hold. Note that it is a property of the simulation-equivalence relation that each member of the set of states over which the relation is computed has a representative.

If there is a transition from u to v on (i_c, o), then there is some i_f, i.e., some valuation of free inputs, that effects this transition, and $T(u, i, o, v)$ holds, where i is a valuation over both i_c and i_f. Thus, formula 1 of Section 6.1 must hold, and there is a transition in E' from initial state s to state t on (i_c, o), which contradicts our original assumption.

For the induction step, let us choose some finite prefix of length $n + 1$, τ_{n+1}, which E realizes, and let us assume E' cannot realize it. The sequence τ_{n+1} can be decomposed into a sequence of length n, τ_n, and a next action symbol $(i_c, o)_{n+1}$. By the induction hypothesis, E' *can* realize τ_n. Therefore, it must be the case that E' cannot add the next action symbol, $(i_c, o)_{n+1}$, onto τ_n, while E can.

Pick any state u in E that is reachable on τ_n. It must have a representative s. We show that s must be reachable in E' on τ_n. The sequence τ_n can be decomposed into sequence τ_{n-1} and next action symbol $(i_c, o)_n$. Let us assume that s is not reachable in E' because from all states in E' reachable on τ_{n-1}, there is no transition to s on $(i_c, o)_n$. However, there is some state u' that is a predecessor of u in E such that there is a transition to u from u' on action symbol $(i_c, o)_n$ in E. Thus, $T(u', (i, o)_n, u)$ holds in E, where the input component i, of $(i, o)_n$ consists of the constrained input valuation i_c, in $(i_c, o)_n$, coupled with any appropriate free input valuation i_f, which will effect the transition. There is a representative for u' in E', s'. Thus, $Rep(s')$ holds. Since s is also a representative state, $Rep(s)$ holds. The following form of formula 1 of Section 6.1 must, therefore, hold:

$$T'(s', (i, o)_n, s) := Rep(s) \wedge Rep(s') \wedge \exists u', u[R_{se}(u', s') \wedge R_{se}(u, s) \wedge T(u', (i, o)_n, u)]$$

Thus, contrary to our assumption, there must be a transition to s on $(i_c, o)_n$ from state s', in E'. We can make a similar argument for any two successive states reached along τ_n, back to initial states.

To summarize, we have proven that for any state u reachable on τ_n in E, the representative of u in E', s must likewise be reachable on τ_n in E'. Given this, we can argue, as we did for the base case of the induction, that if u can transition on action symbol $(i_c, o)_{n+1}$, so can s, and therefore E' realizes τ_{n+1}. □

7. DEADLOCK

In this section we show that our abstraction techniques do not introduce any deadlock. As discussed in Section 3, it is possible for the composition of deadlock-free FSMs to deadlock.

7.1 Freedom from Deadlock

THEOREM 5. *Let E and M be machines such that $E \parallel M$ is deadlock-free. If E is removed via CUFSM removal, then M will not deadlock.*

PROOF. CUFSM removal results in replacement of M's inputs by a set of free primary inputs. As $E \parallel M$ is deadlock-free, there cannot be a deadlock state in M. □

Next we show that the simulation-based state reduction of Section 6 cannot introduce deadlock. Note that the definition of deadlock (see Section 3) applies only to the reachable states of a FSM. This is not an important restriction, since we assume SRSR would be performed after reachability analysis.

THEOREM 6. *Let E and M be FSMs such that $E \parallel M$ is deadlock-free. If E' is derived from E via SRSR, then $E' \parallel M$ is also deadlock-free.*

PROOF. Let S and S' be the state sets of E and E', respectively. $S' \subseteq S$ if E' is derived from E via SRSR. Assume there is a deadlock state (s', u) in $E' \parallel M$, where s' is an internal state of E' and u is an internal state of M. Since $E \parallel M$ is assumed deadlock-free (it is a concrete machine), then (s', u) cannot be a state in $E \parallel M$. Thus, in order for (s', u) to be a deadlock state, it must have been created when SRSR was applied to E.

When SRSR was applied to E, s', a state that existed in E became a representative state. It cannot represent only itself because in that case (s', u), a deadlock state, would be in $E \parallel M$. So s' must represent at least one other state in E, say, s, such that (s, u) is a state in $E \parallel M$. To see this, recall that O, the output of both E and E', overlaps the inputs of M. The state s is reached in E upon some set of finite sequences of states, corresponding to a set of finite sequences over $\Sigma_{I_c \times O}$. This defines a set of input sequences to M, at least one of which leads to state u. If s' represents

s after SRSR, then every sequence of action symbols leading into s in E leads into s' in E'. This is not because each such sequence led into s' in E, but because in the reduced transition relation, T' of E', all transitions going into s are routed into s' (see Section 6.1). To summarize, at this point we know

(1) there is a state (s, u) in $E \parallel M$ that does not deadlock;

(2) there is a state (s', u) in $E' \parallel M$ that we assume does deadlock; and

(3) the same sequences of action symbols leading into (s, u) in $E \parallel M$ lead into (s', u) in $E' \parallel M$.

The possible input valuations to M, while $E \parallel M$ is in state (s, u), is some set $val_m = \{i_0, i_1, \cdots, i_n\}$, where each i_j denotes a value for all the input variables of M. This set of input valuations, val_m, must also be the set of input valuations possible in state (s', u) in $E' \parallel M$. This is because the inputs of M all overlap with outputs of E, and s' and s are trace-equivalent over these outputs (otherwise s' would not be a representative state for s). If the set of inputs and the internal state of a FSM, such as M, are specified, its outputs are likewise specified. These, for M, determine the valuation on the constrained inputs of E' in state (s', u) of $E' \parallel M$. These valuations over constrained inputs must be the same as those of E in $E \parallel M$ in state (s, u), since the constrained inputs of E and E' are the same. Thus, a mismatch in input-output pairs leading to deadlock is impossible in $E' \parallel M$ in state (s', u), since otherwise it would lead to deadlock in $E \parallel M$ in state (s, u). □

8. FUTURE WORK

This work is very new. We have formulated the theory, but have not yet implemented the algorithms described here. Implementing them, and refining the implementations on examples, is our first goal.

We know UFSM detection will not be tractable for large FSMs, since it is a PSPACE-complete problem. The limitation will likely be storing and manipulating the exponential number of sets of states that may be generated. However, if the UFSM-detection algorithm in Figure 2 can be run on sufficiently small FSMs, this may not be a barrier. Thus its utility will depend upon formulating heuristics for partitioning, so that large FSM networks can be divided into many small FSMs, some of which can then be proven to be CUFSMs and safely removed. Developing structural analysis techniques, at the sequential circuit level, to determine good partition points, will be a high priority.

In this paper we have not discussed FSMs that exhibit universality over their outputs for certain sequences of their constrained inputs only or that exhibit universality over a proper subset of their output alphabets. We intend to find suitable abstractions, which we expect will be a common occurance, for these cases. We also intend to expand our use of simulation relations, specifically by utilizing nontrace-equivalent state pairs. Under some circumstances, it is possible for a state to represent other states to

which it is not trace-equivalent, and yet to have trace-equivalence preserved between the original and the reduced machines. We are developing algorithms to find and utilize such representative states.

ACKNOWLEDGMENTS

We thank Panagiotis Manolios, E. Allen Emerson, and Richard Trefler for helpful comments on an earlier draft.

REFERENCES

BOUAIJANI, A., FERNANDEZ, J., AND HALBWACHS, N. 1990. Minimal model generation. In *Proceedings of the 2nd International Conference on Computer Aided Verification* (CAV, New Brunswick, NJ, June), Springer Lecture Notes in Computer Science, vol. 531. Springer-Verlag, Berlin, Germany.

CLARKE, E. M., EMERSON, E. A., AND SISTLA, A. P. 1986. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst. 8*, 2 (Apr. 1986), 244–263.

FERNANDEZ, J. C., KERBRAT, A., AND MOUNIER, L. 1993. Symbolic equivalence checking. In *Proceedings of the Conference on Computer Aided Verification* (CAV '93), C. Courcoubetis, Ed. Springer Lecture Notes in Computer Science, vol. 697. Springer-Verlag, Vienna, Austria.

HOJATI, R. 1996. A BDD-based environment for formal verification of hardware systems. Ph.D. Dissertation. Department of Electrical Engineering and Computer Science, Univ. of Carlifornia at Berkeley, Berkeley, CA.

HOPCROFT, J. AND ULLMAN, J. 1979. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, MA.

KOENIG, D. 1990. *Theory of Finite and Infinite Graphs*. Birkhäuser Boston Inc., Cambridge, MA.

KURSHAN, R. P. 1990. Analysis of discrete event coordination. In *Proceedings of the REX Workshop on Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness* (Mook, The Netherlands, May 29–June 2), J. W. de Bakker, W. P. de Roever, and G. Rozenberg, Eds. Springer Lecture Notes in Computer Science Springer-Verlag, New York, NY, 414–453.

LIN, B. AND NEWTON, A. R. 1991. Efficient symbolic manipulation of equivalence relations and classes. In *Proceedings of the International Workshop on Formal Methods in VLSI Design* (Jan. 1991),

LONG, D. E. 1993. Model checking, abstraction, and compositional verification. Ph.D. Dissertation. Computer Science Department, Carnegie Mellon University, Pittsburgh, PA.

MARTIN, A., KAUFMAN, M., AND PIXLEY, C. 1998. Design constraints in symbolic model checking. In *Proceedings of the Conference on Computer Aided Verification*,

MEYER, A. AND STOCKMEYER, L. 1972. The equivalence problem for regular expressions with squaring requires exponential time. In *Proceedings of the 13th Annual Symposium on Switching and Automata Theory*,

MILNER, R. 1971. An algebraic definition of simulation between programs. In *Proceedings of the Second International Joint Conference on Artificial Intelligence*, British Computer Society, Swinton, UK, 481–489.

RAIMI, R. AND HOJATI, R. 1997. Universal finite state machines and environment modeling. In *Proceedings of the International Workshop on Logic Synthesis*,

PRASAD SISTLA, A., VARDI, M. Y., AND WOLPER, P. 1987. The complementation problem for Büchi automata with applications to temporal logic. *Theor. Comput. Sci. 49*, 2,3 (Mar. 1987), 217–237.

WANG, H. AND BRAYTON, R. 1993. Input don't care sequences in fsm networks. In *Proceedings of the International Conference on Computer Aided Design*,

WATANABE, Y. AND BRAYTON, R. 1994. State minimization of pseudo non-deterministic fsms. In *Proceedings on European Design and Test Conference 94* (Paris, France, Feb. 1994),

Received: January 1998; revised: September 1998; accepted: April 1999