Control of Discrete Event Systems Modeled as Hierarchical State Machines

Y. Brave and M. Heymann

Abstract—**Discrete event systems (DESs) are systems in which state changes take place in response to events that occur discretely, asynchronously and often nondeterministically. In this paper we consider a class of DESs modeled as hierarchical state machines (HSMs), a special case of the statecharts formalism introduced recently. We provide an efficient algorithm for solving reachability problems in the HSM framework that utilizes the hierarchical structure of HSMs. This efficient solution is used extensively in control applications, where controllers achieving a desired behavior are synthesized on-line.**

I. Introduction

In most modeling frameworks for discrete event systems, state-transitions and their associated events constitute the basic structural fragments of the model. (Finite) state-machines and state transition diagrams are the simplest formal mechanism for collecting such fragments into a whole. These models are conceptually appealing because of their inherent simplicity and the fact that they can be formally described by finite automata and their behavior can be described by formal languages.

In the pioneering work on the control of discrete event systems by Ramadage and Wonham [1]–[3] and in much of the work that followed, discrete event systems were indeed modeled in the state-machine framework. A rich control theory has been developed and recently increasing attention has been focused on computational aspects (e.g., [4], [5]).

In many practical control problems, the discrete event system consists of a large number of components that operate concurrently. Thus, the number of states in the state-machine representation of the composite system grows exponentially with the number of parallel components. This exponential explosion in the number of states constitutes a severe shortcoming of the state-machine modeling framework in view of the fact that most computational algorithms for such systems are of complexity that grows at least linearly in the number of states.

To alleviate the modeling complexity of the state-machine formalism, while preserving many of its appealing features, Harel [6] introduced the statechart modeling framework which extends ordinary (sequential) state-machines by endowing them with natural constructs of orthogonality (parallelism), hierarchy (depth), broadcast synchronization and many other sophisticated features that strengthen their modeling power. Hierarchical State Machines (HSMs) are a simplified version of statecharts that extend state machines by adding only the hierarchy and orthogonality features. Specifically,

1) States are organized in a hierarchy of superstates and substates thereby achieving depth.

2) States are composed orthogonally (in parallel), thereby achieving concurrency.

3) Transitions are allowed to take place at all levels of the hierarchical structure, thereby achieving descriptive economy.

HSMs are well suited for modeling and specification of complex processes such as manufacturing systems, communication networks, resource distribution systems, air traffic control systems, etc.

In [7] Drusinsky showed that statecharts exhibit substantial descriptive economy when compared with the equivalent state-machine description of the process. In particular, he showed that the descriptive complexity is exponentially lower than that of the equivalent state-machine model. The present paper deals with computational aspects of a class of HSMs, called asynchronous HSMs (AHSMs). AHSMs are characterized by sparse interaction between parallel components. That is, there is no synchronization of orthogonal components by means of shared events; all interaction is assumed to be modeled either by high-level transitions or in the control constraints (a precise definition of AHSMs is given in Section II). It is shown that such pivotal issues as computation of reachability can be executed in the AHSM framework with exponential reduction of complexity as compared with the equivalent ordinary state machine representation of the process. In fact, this paper identifies and investigates a class of statecharts for which reachability computations and several control problems (which are known to be hard problems (see, e.g., [7]) are decidable and solvable in polynomial time.

We developed an efficient algorithm for testing reachability, that makes fundamental use of the hierarchical structure of the process, thereby demonstrating the inherent advantage of the AHSM representation. This reachability algorithm is then used for solving the *forbidden configuration* control problem, and an efficient algorithm

Manuscript received June 12, 1992; revised December 30, 1992. Paper recommended by Past Associate Editor, C. G. Cassandras. This work was supported in part by the Fonds National Suisse de la Recherche Scientifique and in part by the Technion fund for the promotion of this research.

Y. Brave is with the Institut d'Automatique, Ecole Polytechnique Fédérale de Lausanne, CH-1015 Lausanne, Suisse.

M. Heymann is with the Department of Computer Science, Technion—Israel Institute of Technology, Haifa 32000, Israel.

IEEE Log Number 9213189.

for on-line control execution is developed. Several other applications of the reachability algorithm are also discussed. In the remainder of this section we shall give an informal description of HSMs, and shall define the notion of canonical HSMs. The formal structure of HSMs is presented in Section II, whereas Section III provides an efficient algorithm for testing reachability in the HSM framework. Several applications of this algorithm are presented in Section IV and V, and their complexity is considered in Section VI. A summary of some parts of the present paper appeared in [8] and [9].

A. Informal Description of HSMs

States are represented by Boxes. Hierarchy is represented by the insideness of boxes, as illustrated in Fig. 1(a) where states a and e are *immediate substates* of the state f, and the states b, c, and d are immediate substates of a.

States b, c, and d are also regarded as substates of f. Fig. 1(b) represents the equivalent state machine of the HSM in Fig. 1(a). The symbols $\alpha - \eta$ stand for events associated with the various *transition-paths* (or edges). The state a is called an *OR-state* which means that being in a is equivalent to being in either b or c or d (but not in more than one state at a time). The edge γ, which leaves the contour of a, applies to b, c, and d [just as in Fig. 1(b)]. *Default-arrows* indicate default states. In Fig. 1(b), state e is selected as the initial state, and not a (a fact represented in Fig. 1(a) by the default arrow attached to e). The arrow attached to state c is the default among b, c, and d if we are already in a, and alleviates the need for continuing the β-arrow beyond a's boundary.

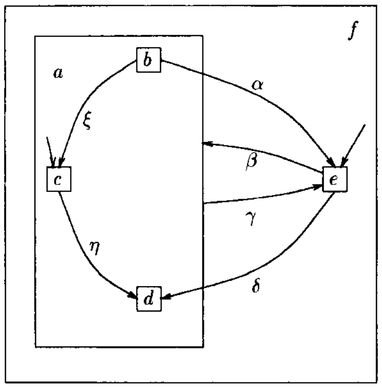
Orthogonality, or concurrency, is the dual of the *OR*-decomposition of states. In Fig. 2(a), state h consists of two *orthogonal components*, f and g, related by *AND*; to be in h is equivalent to being in both f and g, and hence the two default arrows.

The state h is called an *AND-state*. States that have no substates, such as a, b, d, and e, are called *basic*. The tuples $\langle a, g\rangle$, as well as $\langle b, e\rangle$ and $\langle i\rangle$, are *configurations* of the HSM of Fig. 2(a) representing sets of orthogonal states which the HSM can occupy simultaneously. The configuration $\langle b, e\rangle$ is said to be *basic* since it consists only of basic states. The set of all basic configurations of the HSM of Fig. 2(a) is the set of all states in its equivalent 'flat' version of Fig. 2(b).

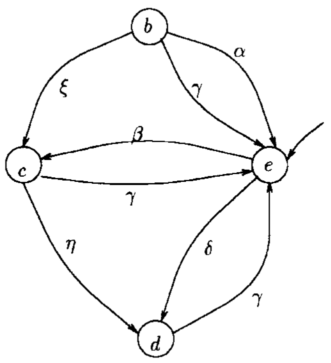
The transition-path labeled λ is represented by the triple $(\langle j\rangle, \lambda, \langle a, d\rangle)$, where $\langle j\rangle$ and $\langle a, d\rangle$ are the *source* and *destination* configuration of this transition-path. Each transition-path t is associated with a unique state of the HSM. This state is the smallest (in terms of its size) OR-state containing t's source and destination configurations, as well as its entire arc. Thus, for instance, the transition-path labeled α belongs to the state f, whereas the transition-path θ belongs to state k. Edges belonging to state f, such as the transition labeled α, do not affect the g component. Thus, if α occurs at $\langle a, d\rangle$, it affects only the f-component, resulting in $\langle b, d\rangle$. The event λ at j causes entrance to the configuration $\langle a, d\rangle$, and μ at $\langle b,e\rangle$ causes transfer to i. The event θ at d means that the *AND*-state $h(= f \times g)$ is left and j is entered, depending only on the fact that the g-component of the configuration is at d. The η-arrow, on the other hand, leaves h unconditionally. By default arrows, event λ at i means entering $\langle b, e\rangle$, whereas ρ means entering $\langle b, d\rangle$. We end this example by remarking that Fig. 3 is a 'zoom-out' of Fig. 2(a) (which suppresses the detailed behavior inside h), whereas Fig. 4 is a graphical representation of the hierarchical structure of the HSM in Fig. 2(a).

Example 1: A hierarchical processing of jobs. Consider the HSM H of Fig. 5. The transition-paths of H represent the following actions.

n_0—take a job for processing.
a_1—partition job for further (hierarchical) processing.
c_1—start hierarchical processing.
b_1—perform 'direct' (nonhierarchical) processing.
d_1—store job.
e_1—continue processing.
f_1—test product.

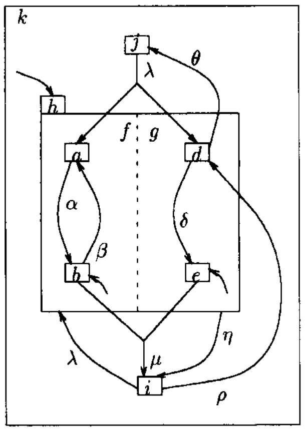


(a)

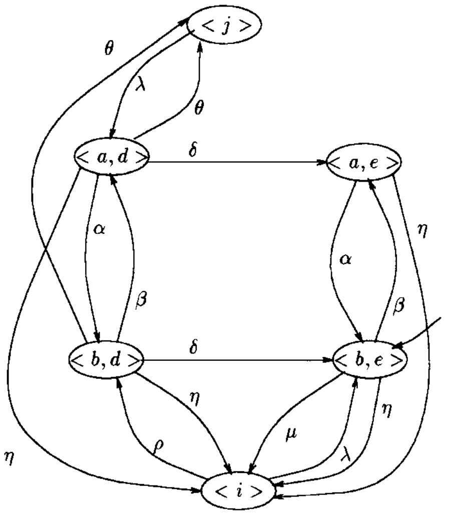


(b)

Fig. 1. (a) An HSM H consisting of OR-states. (b) The equivalent state machine of H.



(a)



(b)

Fig. 2. (a) An HSM H consisting of AND- and OR-states. (b) The equivalent state machine of H.

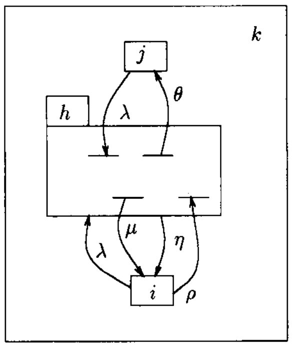


Fig. 3. Zoom-out of Fig. 2(a).

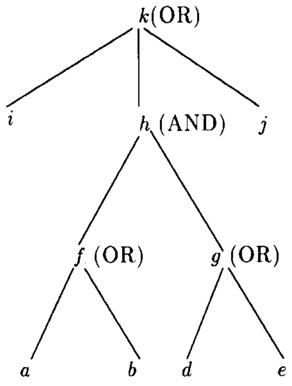


Fig. 4. The hierarchical structure of the HSM in Fig. 2(a).

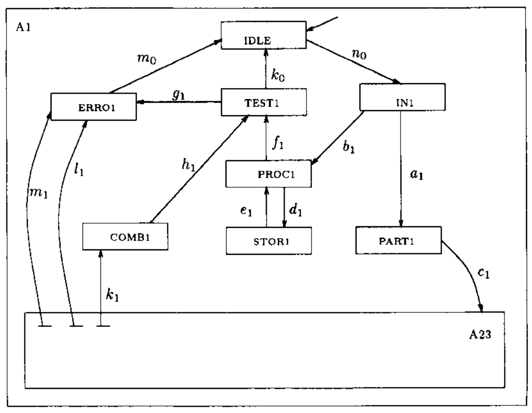


Fig. 5. The HSM of Example 1.

k_1—combine products of higher level of processing.
h_1—test combined product.
g_1, l_1, m_1, m_o—processing failures.
k_0—processing ended successfully.

Now we may 'zoom-in' into state A23 (see Fig. 6). Similarly we may proceed zooming-in into states A45 and A67, and so on. We shall discuss the system in Fig. 7.

Some of the events are uncontrolled (in the sense that they cannot be prevented from occurring by external control) because either i) they model an unpreventable failure—such as the events labeled by g_i, or ii) they correspond to executions or tasks with hard time constraints, and hence should never be disabled (e.g., k_i). In Fig. 7, bars are attached to controlled events. A supervisor (or a controller) S for an HSM H is a device that specifies at each instant a set of controlled events that must be disabled, and thereby restricts the 'behavior' of H. The concurrent run of S and H is denoted S/H.

In this example our objective may be to synthesize a supervisor S for H such that S/H never performs the

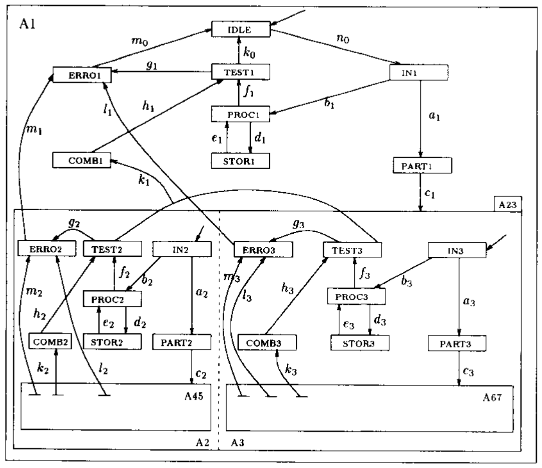


Fig. 6. Zoom-in into state A23 in Fig. 5.

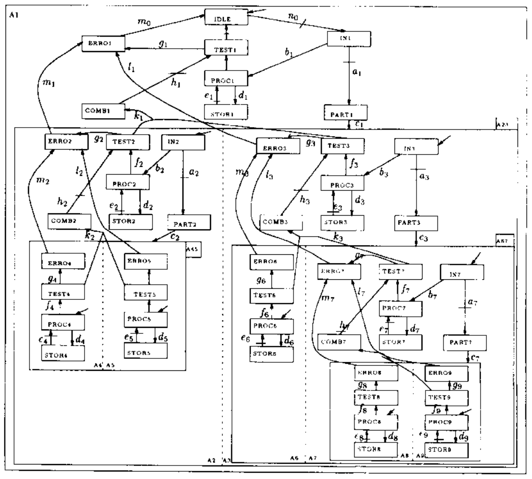


Fig. 7. The full HSM of Example 1.

operations Partition and Combine at the same time. In other words, it may be required that states PARTi and COMBj are never occupied simultaneously. Because of the OR/AND decomposition of states, not all combinations of PARTi and COMBj are relevant. For example, H can never be simultaneously in states PART3 and A67, and hence, neither in PART3 and PART7. Consequently, it would be enough to consider the following forbidden basic configurations: $\langle$PART2, COMB3$\rangle$, $\langle$PART2, COMB7$\rangle$, $\langle$COMB2, PART3$\rangle$ and $\langle$COMB2, PART7$\rangle$.

A synthesis problem of the type described above will be called the *forbidden configuration problem* (FCP), namely the problem of synthesizing a supervisor S for an HSM H such that forbidden configurations are never reached in S/H. In fact, we shall be interested in a supervisor that solves FCP by minimally restricting H's 'behavior.' One possible approach to solving FCP is to construct $M(H)$, the equivalent (flat) state machine of H, and then to synthesize the required supervisor using the algorithm in [3] for computing supremal controllable languages. The complexity of this approach is $O(|Q|)$, where Q is the state set of $M(H)$. Since Q grows exponentially as the number of *AND*-components increases, this approach for solving FCP can be computationally prohibitive for nontrivial examples. In Sections III and IV we propose an alternate approach for solving FCP in the HSMs framework, based on efficient reachability computations and on-line synthesis of minimally-restrictive supervisors. Other works related to the FCP problem are [4] in which a mutual exclusion problem in product systems is considered, and [10], where control policies for discrete event systems modeled as marked graphs are proposed.

A. Canonical Structure

In the sequel, it will be convenient to transform HSMs into a *canonical structure* in which states and transition-paths have a specific standard form, as defined below.

Definition 1: An HSM is said to have an *alternating structure* if the immediate substates of *OR*-states are either *AND*-states or basic states, whereas the immediate substates of *AND*-states are *OR*-states.

Transformation of an HSM into one with alternating structure can be carried out by applying the conversions in Fig. 8. In fact, the third conversion in Fig. 8 also shows how to transform a high-level transition (i.e., a transition whose source configuration includes superstates) into one whose source configuration is basic. In the HSM of Fig. 2(a), which possesses the alternating structure, the source configuration of every transition-path, except for η, is basic.

For explicitly defining the basic destination configuration of a transition-path, we first introduce the notion of default configurations. Recall that each *OR*-state a has a (unique) default arrow which points to a's *default immediate-substate*, denoted $\rho(a)$. We then have the following.

Definition 2: The *default configuration* of a state a, denoted $\hat{\rho}(a)$, is defined as the basic configuration obtained inductively as follows:

1) For an *AND*-state a with immediate substates $a_1,\cdots,a_k$,

$$\hat{\rho}(a) = \langle \hat{\rho}(a_1),\cdots,\hat{\rho}(a_k)\rangle.$$

2) For an *OR*-state a with immediate substates $a_1,\cdots,a_k$,

$$\hat{\rho}(a) = \hat{\rho}(a_i) \quad \text{iff} \quad a_i = \rho(a).$$

3) For a basic state a, $\hat{\rho}(a) = \langle a\rangle$.

In Fig. 2(a), $\hat{\rho}(k) = \hat{\rho}(h) = \langle \hat{\rho}(f), \hat{\rho}(g)\rangle = \langle b, e\rangle$.

Consider now the transition-path $t = (\langle i\rangle, \rho, \langle d\rangle)$ that belongs to state k (see Fig. 2(a)). If the event ρ occurs at $\langle i\rangle$, the HSM enters the AND-state h, and this, in turn,

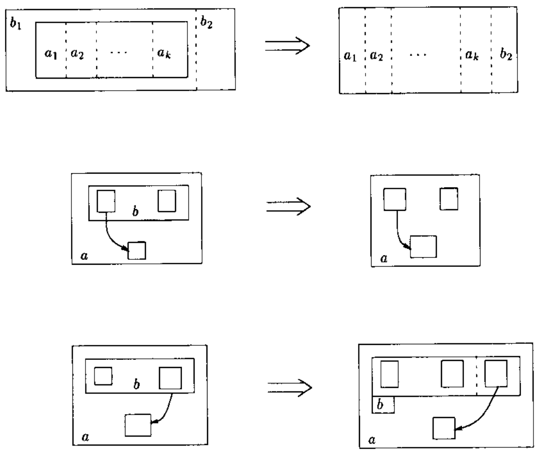


Fig. 8. Conversions required for the AND/OR alternating structure.

means entering states f and g. Since, however, no state in f is specified by the destination configuration of the transition-path t, the default immediate-substate of f is selected. Thus, the *explicit* destination configuration of t is in fact $\langle b, d\rangle$. This procedure for constructing the explicit form is generalized in the following definition.

Definition 3: Let $t = (u, \sigma, v)$ be a transition-path that belongs to state a.

i) By backtracking from the elements of v along the hierarchy tree, mark every state that is a strict superstate of an element in v until the level of state a is reached.

ii) Add to v the default configuration of every unmarked immediate substate of a marked *AND*-state.

iii) The resultant tuple is said to be the *explicit* destination configuration of the transition-path t.

Intuitively, in step i), every state which is occupied by the HSM under configuration u is marked. In fact, if an AND-state c is marked, it means that the destination configuration v specifies a (possibly partial) configuration of c. In step ii), it is then verified whether the destination configuration v specifies for each immediate substate c_i of c a configuration of c_i. If not, the default configuration of every unmarked substate c_i is added to v. Thus, if this procedure is applied to the destination configuration of the transition $t = (\langle i\rangle, \rho, \langle d\rangle)$ in Fig. 2(a), states h and k are marked in step i), and $\langle b\rangle$—the default configuration of state f—is added to $\langle d\rangle$ in step ii). Notice that default configurations, and therefore also explicit destination configurations, are always basic.

Based on the foregoing notation, we introduce the notion of canonical HSMs, as defined below.

Definition 4: An HSM H is said to be in *canonical form* if

1) H has the alternating structure; and

2) For each transition-path $t = (u, \sigma, v)$ of H, the configurations u and v are basic and the destination configuration v is given in its explicit form.

For clarity we shall assume that HSMs are given in their canonical form; thus, henceforth, unless stated otherwise, all configurations are assumed to be basic.

II. Formal Structure of HSMs

An HSM is a structure $H = (A, \vdash, \Sigma, T, \rho)$ where: A is a set of states, $\vdash$ is the hierarchy relation on A, Σ is a set of event symbols, T is a set of transition-paths (or edges) and ρ is a default function. Next we give a detailed description of these elements, as well as some related notions (part of the terminology is adopted from [7] and [11]). First, we shall need the following notations. Let x, y and z be three tuples. We shall write $x \sqsubseteq y$ iff every element of x is an element of y, e.g., $\langle a, b\rangle \sqsubseteq \langle a, c, b, d\rangle$. We shall write $z = x - y$ iff z consists of all elements of x that do not appear in y, e.g., $\langle a, c, b, d\rangle - \langle a, b, e\rangle = \langle c, d\rangle$.

A. States

A is the (finite) set of states of H, consisting of A^+, the subset of *OR*-states, $A^\perp$, the subset of *AND*-states and A^{basic}, the subset of *basic* states. The hierarchical structure of H's states is represented by the binary relation $\vdash$ on A, called the *hierarchy relation* and satisfies the following conditions:

1) There exists a unique state, called the *root state of* H and denoted $r = (r(H))$, such that for no state $a \in A$, $a \vdash r$.

2) For every state $a \in A$, $a \neq r$, there exists a unique state $b \in A$ such that $b \vdash a$. The state b is called the *immediate superstate* of a, whereas a is an *immediate substate* of b.

3) A state $a \in A$ has no immediate substates if and only if a is basic.

4) (Alternating Structure). If $b \vdash a$ then either $b \in A^+ \wedge a \notin A^+$, or $b \in A^\perp \wedge a \in A^+$.

It is clear that the pair $(A, \vdash)$ defines a tree, called the *hierarchy tree* of H. Let l be the depth of the hierarchy tree, and let A_j denote the set of all states at level j, $0 \leq j \leq l$, where the root state is the (unique) state at level 0. The transitive closure of $\vdash$ is denoted $\vdash^+$, and the reflexive and transitive closure of $\vdash$ is denoted $\vdash^*$. Thus $a \vdash^* b$ means that b is a (not necessarily immediate) substate of a, whereas $a \vdash^+ b$ means that b is a substate of a and that $b \neq a$. For a state a with immediate substates $a_1, \cdots, a_k$, we shall sometimes identify a with its set of immediate substates by writing $a \cong a_1 \times \cdots \times a_k$ whenever a is an *AND*-state and $a \cong \bigcup_{i=1}^k a_i$ whenever a is an *OR*-state.

B. Configurations

Let q be a tuple of (disjoint) basic states. (The examples throughout this subsection relate to Fig. 2(a).

- The *restriction* of q to a state a, denoted $q|_a$, is obtained from q by deleting all elements that are not substates of a. For example, $\langle b, e\rangle|_f = \langle b\rangle$.

- A state a is a *superstate* of q if every element of q is a substate of a. For example, h and k are superstates of $\langle b, e\rangle$.
- The *lowest superstate* of q denoted $LS(q)$, is the superstate a of q that satisfies the condition that for each superstate b of q, $b \vdash^* a$. For example, $LS(\langle b, e\rangle) = h$.
- Two states q_1 and q_2 are *orthogonal*, denoted $q_1 \perp q_2$, if either $q_1 = q_2$ or, alternatively, if neither is a superstate of the other and $LS(\langle q_1, q_2\rangle) \in A^{\perp}$. A tuple of states q is *orthogonal* if every pair of states in q is orthogonal. For example, $\langle b, e\rangle$ is orthogonal, whereas $\langle b, a\rangle$ and $\langle b, i\rangle$ are not orthogonal. An orthogonal tuple is also called a *configuration*. Intuitively, a configuration is a tuple of states all of whose elements can be occupied simultaneously when running H.
- Let q be a configuration and let a be a superstate of q. Then q is a *full configuration* of a if it cannot be extended through augmentation with further orthogonal substates of a, i.e., if q satisfies the condition that

$$\forall b \in A, \qquad a \vdash^* b \Rightarrow \langle q, b\rangle \text{ is not orthogonal.} \quad (1)$$

If q is a configuration that does not satisfy (1) then it is a *partial* configuration of a. For example, $\langle b\rangle$ and $\langle b, e\rangle$ are, respectively, a partial and a full configuration of h. The set of all full configurations of a is denoted Q_a (see Lemma A1 for its computation).

- For a basic configuration q of a state a, the *a-span* of q, denoted $\mathscr{S}_a(q)$ is defined as the set of all basic full configurations p of a such that $q \sqsubseteq p$. For example, $\mathscr{S}_k(\langle b\rangle) = \{\langle b, e\rangle, \langle b, d\rangle\}$. For a subset P of basic configurations of a, $\mathscr{S}_a(P) = \bigcup_{p \in P} \mathscr{S}_a(p)$. Lemma A3 summarizes several properties of the span operation.

C. Transition-Paths

Associated with each *OR*-state a is a set T^a of transition-paths. A *transition-path* of a is formally represented by a triple $t = (u, \sigma, v)_a$, where u and v are configurations of a, called, respectively, the source and destination configurations of t, and $\sigma \in \Sigma$ is an event symbol that labels t. (The graphical meaning of the association of a transition path with a specific state has been explained in Section I-A). The set of transition-paths T of the entire HSM is defined as $T = \bigcup_{a \in A^+} T^a$. For each *OR*-state a, we denote by $S_a(D_a)$ the set of all source (respectively, destination) configurations of transition-paths of a.

D. Transition Functions

In the remainder of the paper we shall consider only asynchronous HSMs (AHSMs), that is, HSMs in which no two distinct states have transition-paths labeled by identical event symbols. That is, for every pair of distinct states $a, b \in A^+$

$$(u, \sigma, v) \in T^a \wedge (u', \sigma', v') \in T^b \Rightarrow \sigma \neq \sigma'.$$

We interpret the transition-paths of an AHSM H as follows. Suppose H is at configuration $q \in Q(= Q_r)$. Then a transition labeled $\sigma \in \Sigma$ is *defined at* q iff there exists a transition-path $t = (u, \sigma, v)_a$ such that $u \sqsubseteq q$. Furthermore, the 'next' configuration of H will be p, where p is the configuration obtained from q by replacing (in q) the restriction of q to a with the destination configuration v. Thus, in Fig. 2(a), the transition-path $t = (\langle d\rangle, \theta, \langle j\rangle) \in T^k$ is defined at configuration $\langle b, d\rangle$, as well as at $\langle a, d\rangle$, but it is undefined at $\langle b, e\rangle$. Also, if the AHSM H executes θ at $q = \langle b, d\rangle$ it enters configuration $p = \langle q - q|_k, j\rangle = \langle j\rangle$ (since $q|_k = q$).

The asynchrony assumption of AHSMs excludes the possibility of modeling interaction between orthogonal components through a synchronization formalism in which transition-paths whose labels are identical must occur simultaneously. This type of synchronization, however, can be modeled in AHSMs by high-level transition-paths, as illustrated in Fig. 9. Clearly, the latter approach is only suitable for HSMs with sparse synchronization, for otherwise, the number of high-level transition-paths might increase exponentially as the number of orthogonal components grows.

Formally, we associate with each state $a \in A$ the *transition function* δ_a: $Q_a \times \Sigma \to 2^{Q_a}$ satisfying the condition that for all $q, p \in Q_a$ and $\sigma \in \Sigma$, $p \in \delta_a(q, \sigma)$ iff there exists a transition-path (u, σ, v) of a substate b of a such that

$$u \sqsubseteq q \quad \wedge \quad p = \langle q - q|_b, v\rangle.$$

The transition function of H is defined as $\delta = \delta_r$, where r is the root state. The following observation is an immediate consequence of the definition of δ_a, $a \in A$.

Lemma 1: The transition functions δ_a, $a \in A$ can be computed inductively (up the hierarchy) as follows:

1) For a basic state a, $\delta_a(\langle a\rangle, \sigma) = \varnothing$, $\forall \sigma \in \Sigma$.

2) For an *OR*-state $a \cong \bigcup_{i=1}^{k} a_i$, and for all $q, p \in Q_a$, $\sigma \in \Sigma$

$$\begin{aligned} p \in \delta_a(q, \sigma) \quad \text{iff} \quad & \exists\, i \leq k \text{ s.t. } p \in \delta_{a_i}(q, \sigma) \vee \\ & \exists (u, \sigma, v) \in T^a \text{ s.t. } u \\ & \sqsubseteq q \wedge v = p. \end{aligned} \quad (2)$$

3) For an *AND*-state $a \cong a_1 \times \cdots \times a_k$ and for all $q, p \in Q_a$, $\sigma \in \Sigma$

$$\begin{aligned} p \in \delta_a(q, \sigma) \quad \text{iff} \quad & \exists\, i \leq k \text{ s.t. } p|_{a_i} \in \delta_{a_i}(q|_{a_i}, \sigma) \wedge \\ & \forall j \neq i,\ p|_{a_j} = q|_{a_j}. \end{aligned} \quad (3)$$

We interpret an AHSM $H = (A, \vdash, \Sigma, T, \rho)$ as a device that starts at configuration $q_o = \hat{\rho}(r)$ (= the default configuration of the root state) and executes configuration transitions according to its transition function δ. That is, H can be represented by its equivalent (ordinary) state machine $M(H) = (Q, \Sigma, \delta, q_o)$ whose states consist of all full configurations of H and whose transition function is the transition function δ of (the root of) H.

III. Reachability

In this section, we discuss the problem of testing reachability of a set of (full or partial) configurations from a

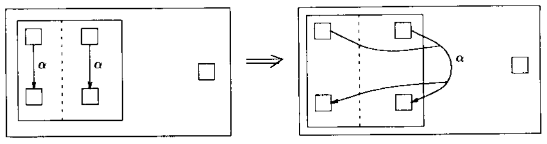


Fig. 9. Conversion of (a) shared-event synchronization into (b) high-level transition-path synchronization.

given full configuration. Let $a \in A$. A *path* in a is a finite sequence $s = q_o, \sigma_1, q_1, \cdots, \sigma_n, q_n$, where the q_i are full configurations of a and the σ_i are symbols in Σ, such that $q_i \in \delta_a(q_{i-1}, \sigma_i)$ for all $i = 1, 2, \cdots, n$. In this case we say that q_n is *a-reachable* from q_o. For a subset P of full configurations of a, define $R_a(H, P)$ to be the set of all full configurations of a that are a-reachable from P. Similarly, define $R_a^{-1}(H, P)$ to be the set of all full configurations of a from which P can be a-reached.

Given a full configuration q of H and a subset P of configurations of H, our objective is to verify whether there exists a full configuration w of H such that for some $p \in P$, $p \sqsubseteq w$, and w is r-reachable from q, i.e., to verify whether

$$q \in R_r^{-1}(H, \mathscr{S}_r(P)), \tag{4}$$

where $\mathscr{S}_r(P)$ is the r-span of P. One way to test (4) is by performing the computation on $M(H)$, the equivalent state machine of H. Such a test has the disadvantage that the number of states of $M(H)$ grows exponentially with the number of orthogonal components of H. We shall next present an alternate algorithm for testing (4), that does not require the construction of $M(H)$ and has a fundamental computational advantage as discussed in detail in Section VI.

Consider the AHSM H depicted in Fig. 10. Let $q = \langle a_1, b_1 \rangle$ and $p = \langle a_3, b_3 \rangle$, and suppose we wish to verify whether p is c-reachable from q. That is, we wish to find out whether there exists a path s that starts at q and ends at p, such that s consists only of transition-paths that belong to substates of c, i.e., transition-paths labeled by ψ, ϕ, θ and ρ. By the asynchrony assumption, this question can be resolved by independent reachability tests in states a and b. Thus, we check whether $\langle a_3 \rangle$ is a-reachable from $\langle a_1 \rangle$, and whether $\langle b_3 \rangle$ is b-reachable from $\langle b_1 \rangle$. Since the answer to both questions is negative, we conclude that $\langle a_3, b_3 \rangle$ is not c-reachable from $\langle a_1, b_1 \rangle$. This test for checking reachability within *AND*-states is formally stated in the following lemma.

Lemma 2: Let $a \cong a_1 \times \cdots \times a_k$ be an *AND*-state, and let $q, p \in Q_a$ be two full configurations. Then p is a-reachable from q iff for all $1 \leq i \leq k$, $p|_{a_i}$ is a_i-reachable from $q|_{a_i}$.

The above lemma can be proved by a repeated application of step 3) in Lemma 1. Now we consider again Fig. 10 and examine the effect of the transition-paths labeled by α, β, γ and δ of state f on the reachability of a configuration p from a configuration q. Specifically, we

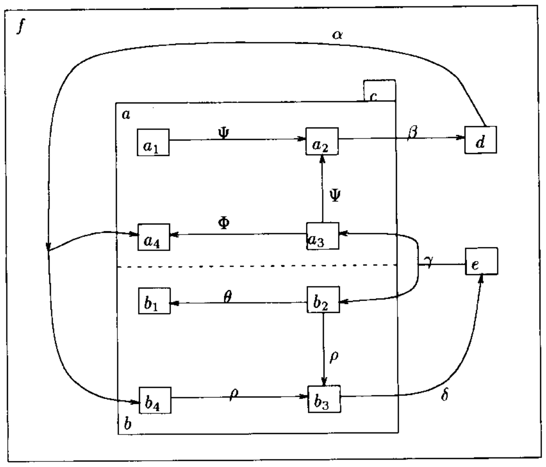


Fig. 10. The AHSM discussed in Section III.

wish to check whether $p = \langle a_3, b_3 \rangle$ is f-reachable from $q = \langle a_1, b_1 \rangle$. Since p is *not* c-reachable from q, we search for a configuration $s \in S_f$ (i.e., a source of a transition-path of f) that is c-reachable from q. Since the source $\langle b_3 \rangle$ of δ is not c-reachable from $q = \langle a_1, b_1 \rangle$, we proceed with β whose source $\langle a_2 \rangle$ is c-reachable from q. Our final destination is $p = \langle a_3, b_3 \rangle$ (which is a configuration of c), and thus we continue with α, thereby entering configuration $\langle a_4, b_4 \rangle$. Now we check whether $p = \langle a_3, b_3 \rangle$ is c-reachable from $\langle a_4, b_4 \rangle$. Since this is not the case, we continue with δ, the only transition-path of f whose source $\langle b_3 \rangle$ is c-reachable from $\langle a_4, b_4 \rangle$, and return to state c through γ. This search terminates successfully since $p = \langle a_3, b_3 \rangle$ is c-reachable from the destination $\langle a_3, b_2 \rangle$ of γ. In summary, $p = \langle a_3, b_3 \rangle$ is f-reachable from $q = \langle a_1, b_1 \rangle$ via the following path: $\langle a_1, b_1 \rangle$, ψ, $\langle a_2, b_1 \rangle$, β, $\langle d \rangle$, α, $\langle a_4, b_4 \rangle$, ρ, $\langle a_4, b_3 \rangle$, δ, $\langle e \rangle$, γ, $\langle a_3, b_2 \rangle$, ρ, $\langle a_3, b_3 \rangle$.

The following lemma is the analog of Lemma 2 for testing reachability within *OR*-states, and it formalizes the idea of the foregoing discussion.

Lemma 3: Let $a \cong \bigcup_{i=1}^{k} a_i$ be an OR-state, let q be a full configuration of state a and let p be a configuration of a. Then there exists a full configuration $w \in Q_a$, with $p \sqsubseteq w$, that is a-reachable from q iff there exists a sequence $a_{i_o}, a_{i_1}, \cdots, a_{i_n}$ of states, a sequence $(u_1, \sigma_1, v_1), \cdots, (u_n, \sigma_n, v_n)$ of transition-paths of a (where for $j = 1, \cdots, n$, u_j is a configuration of $a_{i_{j-1}}$ and v_j is a full configuration of a_{i_j}), and a sequence $w_o, w_1, \cdots, w_n$ of full configurations of a such that the following conditions hold:

1) w_o is a_{i_o}-reachable from q, and $p \sqsubseteq w_n$.

2) For $j = 1, 2, \cdots, n$, $u_j \sqsubseteq w_{j-1}$ and w_j is a_{i_j}-reachable from v_j.

Let us illustrate Lemma 3 w.r.t. the example considered prior to the above lemma. We have shown there that

$p = \langle a_3, b_3 \rangle$ is f-reachable from $q = \langle a_1, b_1 \rangle$. In this case, we have indeed (see Lemma 3) a sequence $a_{i_o}, \cdots, a_{i_4} = c, d, c, e, c$ of states, a sequence $(u_1, \sigma_1, v_1), \cdots, (u_4, \sigma_4, v_4) = (\langle a_2 \rangle, \beta, \langle d \rangle), (\langle d \rangle, \alpha, \langle a_4, b_4 \rangle), (\langle b_3 \rangle, \delta, \langle e \rangle), (\langle e \rangle, \gamma, \langle a_3, b_2 \rangle)$ of transition-paths of f, and a sequence $w_o, \cdots, w_4 = \langle a_2, b_1 \rangle, \langle d \rangle, \langle a_4, b_3 \rangle$ of configurations of f that satisfy conditions 1 and 2 in Lemma 3. Notice that the sequence $\{a_{i_o}, a_{i_1}, \cdots, a_{i_n}\}$ need not consist of distinct elements, as is shown above.

Now that we have established the inductive arguments of Lemmas 2 and 3, we turn to our main goal of testing (4). But first consider again Fig. 10. Recall that during the search performed in the paragraph preceding Lemma 3 for deciding whether the configuration $p = \langle a_3, b_3 \rangle$ is f-reachable from $q = \langle a_1, b_1 \rangle$, we checked whether p is c-reachable from q, from $\langle a_4, b_4 \rangle$ and from $\langle a_3, b_2 \rangle$, where the latter are configurations of c that are in D_f, the set of all destinations of transition-paths of f (see Subsection II-C). In fact, a reachability test from q, $\langle a_4, b_4 \rangle$ and $\langle a_3, b_2 \rangle$ has been carried out also w.r.t. $\langle a_2 \rangle$ and $\langle b_3 \rangle$ that are configurations of c and belong to S_f, the set of all sources of transition-paths of f. Thus we conclude that the only information regarding reachability within state c, that may be required for reachability computations within state f, is the c-reachability of configurations in $S_f \cup \{p\}$ from configurations in $D_f \cup \{q\}$. This observation, as well as Lemmas 2 and 3, are the key points in the development of the algorithm below for reachability computations associated with (4).

Fix a full configuration q of H, and a set P of configurations of H. For each state $a \in A$ we define a set $X_a(q)$ (called the *input* set of a) of full configurations of a, and a set $Y_a(P)$ (called the *output* set of a) of configurations of a, as follows. A configuration x of a is an element in $X_a(q)$ iff either $x = q|_a$, or $x = d|_a$ where d is a destination configuration in D_b for some strict superstate b of a. That is,

$$X_a(q) = \{q|_a\} \cup \{d|_a | d \in D_b \wedge b \vdash^+ a\}. \quad (5)$$

Thus, in Fig. 10, $X_c(\langle a_1, b_1 \rangle) = \{\langle a_1, b_1 \rangle\} \cup \{\langle a_4, b_4 \rangle, \langle a_3, b_2 \rangle\}$. Analogously, a configuration y of state a is an element in $Y_a(P)$ iff either $y = p|_a$ for some $p \in P$, or $y = s|_a$ where s is a source configuration in S_b for some strict superstate b of a. That is

$$Y_a(P) = P|_a \cup \{s|_a | s \in S_b \wedge b \vdash^+ a\}. \quad (6)$$

Thus, in Fig. 10, $Y_c(\langle a_3, b_3 \rangle) = \{\langle a_3, b_3 \rangle\} \cup \{\langle a_2 \rangle, \langle b_3 \rangle\}$.

It should be clear from the examples above that for each state $a \in A$ at a given level in the hierarchy, all the information about reachability that may be required for higher level computations concerns only a-reachability tests between input configurations in $X_a(q)$ and output configurations in $Y_a(P)$. For the computation of the sets $X_a(q)$ and $Y_a(P)$ we need the following lemma.

Lemma 4: Let w be a configuration. The following algorithm computes $w|_a$ for every state $a \in A$ satisfying $w|_a \neq \langle\ \rangle$. It has complexity $O(l \cdot |w|)$, where l is the depth of the hierarchy tree and $|w|$ is the number of elements in w.

Algorithm 1:

1) Let $w|_a = \langle a \rangle$ for every basic state a such that $\langle a \rangle \sqsubseteq w$ and for $j = 0, 1, \cdots, l$ let

$$B_j = \{a | \langle a \rangle \sqsubseteq w \wedge a \in A_j\} \quad (7)$$

where A_j is the set of all states at level j. Also, define $C_l := B_l$.

For $j = l - 1, l - 2, \cdots, 0$ do steps 2)–4).

2) Let

$$C_j = B_j \cup \{a | a \vdash b \text{ for some } b \in C_{j+1}\}. \quad (8)$$

3) For an *OR*-state $a \cong \bigcup_{i=1}^{k} a_i$ such that $a \in C_j$ let $w|_a = w|_{a_i}$, where a_i is the (unique) substate of a in C_{j+1}.

4) For an *AND*-state $a \cong a_1 \times \cdots \times a_k$ such that $a \in C_j$ let $w|_a = \langle w_1, \cdots, w_k \rangle$ where

$$\forall 1 \leq i \leq k, \qquad w_i = \begin{cases} w|_{a_i} & \text{if } a_i \in C_{j+1} \\ \langle\ \rangle & \text{otherwise.} \end{cases}$$

It is worth noting that $w|_a = \langle\ \rangle$ for every $a \in A_j - C_j$ and $j = 0, 1, \cdots, l$. Now, Algorithm 1 can be used for computing the input sets $X_a(q)$ as explained below.

Lemma 5: Given a configuration q, the following algorithm computes the input sets $X_a(q)$ for all $a \in A$.

Algorithm 2:

1) Let $X_a(q) = \varnothing$ for every $a \in A$.

2) Execute Algorithm 1 with $w = q$, and then set

$$\forall j = 0, 1, \cdots, l, \forall a \in C_j, \qquad X_a(q) \leftarrow X_a(q) \cup \{w|_a\}$$

where the sets C_j are computed by Algorithm 1 [see (8)].

3) For every level i, $0 \leq i \leq l$, every *OR*-state $b \in A_i$ and every configuration $d \in D_b$, execute Algorithm 1, with $w = d$, and then set

$$\forall j = l, l - 1, \cdots, i + 1, \forall a \in C_j,$$
$$X_a(q) \leftarrow X_a(q) \cup \{w|_a\}. \quad (9)$$

Given a set P of configuration of H, the output set $Y_a(P)$ can be computed by Algorithm 2 with P replacing q and S_b replacing D_b. The complexity of Algorithm 2, as well as that of all subsequent algorithms, is discussed in Section VI.

As was explained above, we test a-reachability between configurations in $X_a(q)$ and configurations in $Y_a(q)$. These a-reachability tests are carried out by the inductive Lemmas 2 and 3, depending on whether a is an *AND*- or an *OR*-state. The results of these tests are represented by a subset $W_a(q, P) \subseteq X_a(q) \times Y_a(P)$, where for a pair $(x, y) \in X_a(q) \times Y_a(P)$, $(x, y) \in W_a(q, P)$ means that y is a-reachable from x. The computation proceeds inductively (up the hierarchy), and since for the root state r, $X_r(q) = \{q\}$, and $Y_r(P) = P$, the verification of (4) can be accomplished by testing whether $W_r(q, P) = \varnothing$. Formally, we have the following algorithm for testing reachability.

Algorithm 3: Given q and P as above, compute $W_a(q, P) \subseteq X_a(q) \times Y_a(P)$ inductively (up the hierarchy) as follows:

1) For a basic state a, $W_a(q, P) = \emptyset$.

2) For an *OR*-state $a \cong \bigcup_{i=1}^{k} a_i$, and for all $(x, y) \in X_a(q) \times Y_a(P)$, $(x, y) \in W_a(q, P)$ iff y is reachable from x in the digraph $G_a(q, P)$, whose node set is

$$V_a(q, P) = X_a(q) \cup Y_a(P) \cup D_a \cup S_a, \tag{10}$$

and whose edge set is

$$E_a(q, P) = \left[\bigcup_{i=1}^{k} W_{a_i}(q, P)\right] \cup \{(u, v) | \exists \sigma, \quad \text{s.t.} \quad (u, \sigma, v) \in T^a\}. \tag{11}$$

3) For an *AND*-state $a \cong a_1 \times \cdots \times a_k$ and for all

$$(x, y) = (x_1, \cdots, x_k, y_1, \cdots, y_k) \in X_a(q) \times Y_a(P),$$

where $x_i = x|_{a_i}$ and $y_i = y|_{a_i}$,

$$(x, y) \in W_a(q, P) \quad \text{iff} \quad \forall 1 \le i \le k, \quad (x_i, y_i) \in W_{a_i}(q, P) \vee y_i = \langle\ \rangle. \tag{12}$$

Proposition 1: Upon termination of Algorithm 3 (at the root state r), condition (4) holds iff the relation $W_r(q, P)$ is nonempty, that is,

$$q \in R_r^{-1}(H, \mathscr{S}_r(P)) \qquad \text{iff} \qquad W_r(q, P) \neq \emptyset. \tag{13}$$

We end this section with an example illustrating Algorithm 3.

Example 2: Consider the AHSM H of Fig. 10, and suppose we wish to test whether $p = \langle a_3, b_3 \rangle$ is reachable from $q = \langle a_1, b_1 \rangle$. For applying Algorithm 3, we first compute the input and output sets: $X_f(q) = \{q\}$, $Y_f(p) = \{p\}$, $X_c(q) = \{q, \langle a_4, b_4 \rangle, \langle a_3, b_2 \rangle\}$, $Y_c(p) = \{p, \langle a_2 \rangle, \langle b_3 \rangle\}$, $X_a(q) = \{\langle a_1 \rangle, \langle a_3 \rangle, \langle a_4 \rangle\}$, $Y_a(p) = \{\langle a_3 \rangle, \langle a_2 \rangle\}$, $X_b(q) = \{\langle b_1 \rangle, \langle b_2 \rangle, \langle b_4 \rangle\}$ and $Y_b(p) = (\langle b_3 \rangle)$. The digraphs $G_a(q, p)$ and $G_b(q, p)$ (see step 2) in Algorithm 3) consist of the transition-paths of states a and b, respectively. Thus,

$$W_a(q, p) = \{(\langle a_1 \rangle, \langle a_2 \rangle), (\langle a_3 \rangle, \langle a_3 \rangle), (\langle a_3 \rangle, \langle a_2 \rangle)\},$$

and

$$W_b(q, p) = \{(\langle b_2 \rangle, \langle b_3 \rangle), (\langle b_4 \rangle, \langle b_3 \rangle)\}.$$

The digraph $G_f(q, P)$ is given in Fig. 11, where the set $W_c(q, p)$ is the set of all dashed arrows. Since p is reachable from q in $G_f(q, p)$, and therefore, $W_f(q, p) \neq \emptyset$, we conclude that p is reachable from q in H.

IV. Controlled AHSMs

In this section, we consider controlled AHSMs, thereby illustrating an important application of the reachability algorithm proposed in the previous section. In a *controlled* AHSM, the set of events Σ is partitioned into two disjoint subsets Σ_c and Σ_u of *controlled* and *uncontrolled* event

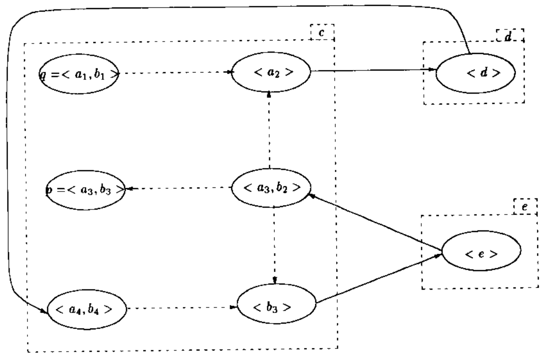


Fig. 11. The diagraph $G_f(q, p)$ of state f in Fig. 10.

sets, respectively. Clearly, this partition induces a similar partition of the transition-path set T to T_c and T_u.

A *configuration feedback supervisor* (or in short *supervisor*) for an AHSM H is a map S: $Q \to 2^{\Sigma_c}$ that specifies at each full configuration of H a set of controlled events that must be disabled. The equivalent state machine of the supervised AHSM, denoted S/H, is given by

$$M(S/H) = (Q, \Sigma, \xi, q_o)$$

where the *controlled* transition function ξ: $Q \times \Sigma \to 2^Q$ is defined as follows: For all $q, p \in Q$ and $\sigma \in \Sigma$

$$p \in \xi(q, \sigma) \qquad \text{iff} \qquad p \in \delta(q, \sigma) \wedge \sigma \notin S(q).$$

That is, a transition (labeled σ) from a configuration q to a configuration p may occur in S/H iff this transition is possible in H and is allowed by the supervisor S.

Given an AHSM $H = (A, \vdash, \Sigma, T, \rho)$ with $\Sigma = \Sigma_u \cup \Sigma_c$ and $T = T_u \cup T_c$, we define H_u to be the AHSM obtained by removing from H all controlled transition-paths. That is, $H_u = (A, \vdash, \Sigma_u, T_u, \rho)$. Furthermore, define S_Σ to be the supervisor that disables all controlled events, i.e., for each $q \in Q$, $S_\Sigma(q) = \Sigma_c$. Clearly, $M(S_\Sigma/H) = M(H_u)$.

We now pose the following control synthesis problem.

Problem 1: Forbidden Configuration Problem (FCP): Let F be a set of configurations of H. Synthesize a supervisor S such that

$$R(S/H, q_o) \subseteq Q - \mathscr{S}(F), \tag{14}$$

where $\mathscr{S}(F)$ is the set of all (forbidden) full configurations of H spanned by F (see Subsection II-B).

Thus, the problem FCP consists of synthesizing (if possible) a supervisor S such that S/H, initialized at the default configuration q_o, never reaches a forbidden configuration $p \in \mathscr{S}(f)$, with $f \in F$. It is clear that if H is at some configuration q and a forbidden full configuration $f \in \mathscr{S}(F)$ is reachable from q via an uncontrolled path (i.e., a path consisting only of uncontrolled events) then no supervisor can prevent H from reaching f. Thus, in fact, the set F of forbidden configurations induces a larger set of forbidden configurations, denoted $\Theta(F)$ and called the *extended* forbidden set, consisting of all full configurations of H from which a full configuration in $\mathscr{S}(F)$ can be reached via an uncontrolled path. That is,

$$\Theta(F) = R^{-1}(H_u, \mathscr{S}(F)). \tag{15}$$

Now the existence of a supervisor S satisfying (14) and thereby solving FCP can be stated as follows.

Observation 1: FCP is solvable iff

$$q_o \notin \Theta(F). \tag{16}$$

An efficient test of (16) is obtained by modifying Algorithm 3 as follows. In analogy to S_a and D_a, define $\hat{S}_a$ and $\hat{D}_a$ to be the set of all source, respectively destination, configurations of *uncontrolled* transition-paths of a. We then have the following definition.

Definition 5: Let $\hat{W}(q, P)$ be defined as the result of Algorithm 3 modified as follows:

1) The sets $\hat{D}_b$ and $\hat{S}_b$ replace the sets D_b and S_b, respectively in (5), (6), and (10).

2) The set T_u^a (the subset of uncontrolled transition-paths of state a) replaces T^a in (11).

The consequence of modifications 1 and 2 in Definition 5 is that $\hat{W}(q, P)$ represents (uncontrolled) reachability with respect to H_u just as $W(q, P)$ represents reachability with respect to H. Thus, following Proposition 1 and Observation 1, we conclude that FCP is solvable iff $\hat{W}(q_o, F) = \emptyset$.

It follows from Observation 1, that whenever FCP is solvable, it can be solved by S_Σ. However, S_Σ may be too restrictive in the sense that it eliminates controlled transitions whose deletion is not necessary for satisfying (14). Thus, we shall say that a supervisor S is a *minimally restrictive* solution of FCP if for every supervisor S' solving FCP

$$R(S'/H, q_o) \subseteq R(S/H, q_o).$$

To see that whenever FCP is solvable, it indeed has a minimally restrictive solution, consider the set $\overline{\Theta}(F)$ of all full configurations in $Q - \Theta(F)$, from which a configuration in $\Theta(F)$ can be reached in one transition. It is clear from (15) that every such transition [that takes H into $\Theta(F)$] is controlled, and must be disabled by every supervisor solving FCP. Moreover, if no other transitions (except the transitions that take H from $\overline{\Theta}(F)$ into $\Theta(F)$) are prevented by a supervisor S, then S is a minimally restrictive solution of FCP. Thus we have the following proposition.

Proposition 2: Assuming FCP is solvable, the following algorithm computes a minimally restrictive supervisor. It has complexity $O(|Q|)$ (we neglect the dependence of the complexity bound on Σ).

Algorithm 4:

1) Compute $\Theta(F)$ [see (15)].

2) For every $q \in Q$, let

$$S(q) = \begin{cases} \{\sigma \in \Sigma_c | \delta(q, \sigma) \cap \Theta(F) \neq \emptyset\} \\ \quad \text{if } q \in Q - \Theta(F) \\ \emptyset \quad \text{otherwise.} \end{cases} \tag{17}$$

The complexity of the 'inverse' reachability computation of step 1) in Algorithm 4, as well as that of the control synthesis in step 2) is $O(|Q|)$. Thus, the overall complexity of Algorithm 4 is $O(|Q|)$. It is worth noting that whenever solvable, FCP need not have a unique minimally restrictive solution. Indeed, in the region $Q - \Theta(F)$, every minimally restrictive solution of FCP, as well as any other solution of FCP must prevent controlled transitions at least as S [the supervisor specified by (17)] does. Yet, a supervisor may disable more controlled transitions than specified by (17) and still be a minimally restrictive solution of FCP. This is due to the fact that a state may, in general, be reached along multiple paths, and the fact that controllers often have "don't care" controls.

Since the Q grows exponentially with the number of orthogonal components in H, the *a priori* ('off-line') synthesis of a minimally restrictive supervisor, as proposed by Algorithm 4, may be intractable. Thus we proceed according to the following *on-line* approach. Instead of constructing the extended forbidden set $\Theta(F)$ explicitly, and then specifying (*a priori*) a control strategy for each configuration in $Q - \Theta(F)$, we synthesize an appropriate control during execution as follows. Whenever H performs a configuration transition, and thereby enters a new configuration q, all controlled events are immediately disabled. Then only controlled events that do not take H to configurations from which a forbidden configuration is reachable via an uncontrolled path, are enabled. These reachability tests are carried out using the modified version of Algorithm 3.

It should be emphasized that in the supervisory control framework (where processes are modeled as ordinary state machines) there is no substantial benefit in preferring one of these approaches (the 'on-line' and the 'off-line') to the other, since their complexities are both polynomial in the size of the state set. In the AHSM framework, however, the on-line approach may exhibit an exponential reduction in complexity when compared with the (traditional) off-line approach. That is, a single off-line computation of exponential complexity is replaced with on-line calculations, one after each event, where each individual calculation is polynomial.

Formally, the on-line synthesis procedure re-executes Algorithm 5 (below) whenever the supervised AHSM enters a new configuration $q \in Q$.

Algorithm 5:

1) Set $S(q) = \Sigma_c$ (i.e., disable all controlled transitions).

2) Using Algorithm 1, compute $q|_a$ for every state $a \in A$ satisfying $q|_a \neq \langle\ \rangle$, and let $A(q)$ be the set of all OR-states a for which $q|_a \neq \langle\ \rangle$. For every OR-state $a \in A(q)$ and every controlled event $\sigma \in \Sigma^a$ (where Σ^a is the set of all event-symbols that label transition-paths of a) perform the following two steps:

3) Compute the set $\delta(q, \sigma)$ consisting of all 'next' configurations $p = \langle q - q|_a, v\rangle$, where v is a destination of a transition-path $(u, \sigma, v) \in T^a$ satisfying $u \sqsubseteq q$.

4) If for every $p \in \delta(q, \sigma)$, $\hat{W}(p, F) = \varnothing$ (where $\hat{W}(p, F)$ is computed by the modified version of Algorithm 3) then set $S(q) \leftarrow S(q) - \{\sigma\}$ (i.e., enable σ at q).

Upon termination of Algorithm 5, the control $S(q)$ is minimally restrictive in the sense that enabling any event $\sigma \in S(q)$ may cause H to reach a forbidden configuration $f \in \mathscr{F}(F)$. The correctness of this algorithm follows from Proposition 1 and the definition of the transition function δ (see Subsection II-D). We remark that:

1) In steps 3) and 4) of Algorithm 5 we consider only OR-states a for which $q|_a \neq \langle\ \rangle$; for if $q|_a = \langle\ \rangle$ then for no transition-path $(u, \sigma, v) \in T^a$, the condition $u \sqsubseteq q$ is satisfied, and therefore $\delta(q, \sigma) = \varnothing$ for every $\sigma \in \Sigma^a$.

2) The decision as to whether a given event should be disabled or enabled does not depend on the decision regarding other events. Thus, the sequential steps 3) and 4) in Algorithm 5 can be carried out simultaneously with regard to all controlled events, thereby reducing the disabling period of events satisfying the condition in step 4) of Algorithm 5.

3) Since step 4) is repeated for every possible 'next' configuration p of H, an additional reduction in the computational effort required for the on-line synthesis can be obtained by pre-computing $\hat{W}_a(\langle\ \rangle, F)$ for all $a \in A$, and then constructing $\hat{W}_a(p, F)$ as follows. For every state a such that $p|_a = \langle\ \rangle$, let $\hat{W}_a(p, F) = \hat{W}_a(\langle\ \rangle, F)$ (since $X_a(p) = X_a(\langle\ \rangle)$; see (5)). If, however, $p|_a \neq \langle\ \rangle$ then perform steps 2) and 3) in (the modified) Algorithm 3 only for pairs $(x, y) \in \{p|_a\} \times Y_a(F)$.

V. Accessibility and Determinism

In this section we define a notion of accessibility that can help in reducing the state set of a given AHSM without changing its reachable set of configurations. We also discuss the conventional notion of accessibility and provide an efficient condition for testing it. We conclude this section with a characterization of deterministic AHSMs.

We begin with a weak type of accessibility that we call *partial accessibility*. Roughly speaking, an AHSM H is partially accessible iff there exists no basic state that is never occupied by H. Formally,

Definition 6: An AHSM H is partially accessible iff

$$\forall a \in A^{basic}, \exists p \in R(H, q_o) \quad \text{s.t.} \quad \langle a \rangle \sqsubseteq p. \quad (18)$$

Clearly, testing partial accessibility by an exhaustive search of $R(H, q_o)$ is unacceptable, since the size of $R(H,q_o)$ is $O(|Q|)$. Thus, (18) can be verified as follows.

Proposition 3: An AHSM H is partially accessible iff

$$\forall a \in A^{basic}, \qquad (q_o, \langle a \rangle) \in W(q_o, P).$$

where $P = \{\langle a \rangle | a \in A^{basic}\}$.

Clearly, if a basic state a is never occupied by H (i.e., H is not partially accessible), the state a, as well as any transition $(u, \sigma, v) \in T$ incident to a (i.e., a transition (u, σ, v) satisfying $\langle a \rangle \sqsubseteq u$ or $\langle a \rangle \sqsubseteq v$), can be eliminated from H. Nevertheless, one may not infer from the latter that every configuration of H is reachable from the initial configuration whenever H is partially accessible. A simple counterexample is given in Fig. 12. Here, the configuration $\langle c, d \rangle$ is not reachable from the initial configuration $q_o = \langle a \rangle$, although H is partially accessible. This observation can be easily verified if the equivalent state machine of H is considered (see Fig. 13).

Thus, partial accessibility is a necessary (but not a sufficient) condition for conventional accessibility.

Definition 7: An AHSM is *accessible* if every full configuration of H is reachable from q_o, i.e., if $R(H, q_o) = Q$.

For testing accessibility, a notion of *connectivity* is defined in terms of the sets $X_a(q_o)$ [see (5)] and the digraphs $G_a(q_o, \langle\ \rangle)$, $a \in A$ (see step 2) in Algorithm 3).

Definition 8: An OR-state $a \cong \bigcup_{i=1}^{k} a_i$ is *connected* if for every configuration $x \in X_a(q_o)$ and for every substate a_i there exists a configuration $y \in X_{a_i}(q_o)$ such that y is reachable from x in the digraph $G_a(q_o, \langle\ \rangle)$.

Consider again the AHSM of Fig. 12. Here, state r is connected whereas the state f is not. For showing the latter, notice first that $q_o = \langle a \rangle$, $X_f(q_o) = \{\langle b \rangle, \langle c \rangle\}$, $X_b(q_o) = \{\langle b \rangle\}$ and $X_c(q_o) = \{\langle c \rangle\}$. Let $x = \langle b \rangle \in X_f(q_o)$. Then no configuration $y \in X_c(q_o)$ is reachable from x in digraph $G_f(q_o, \langle\ \rangle)$. Thus, f is not connected.

Now we have the following proposition.

Proposition 4: An AHSM H is accessible if every OR-state of H is connected.

Next, we turn to the notion of deterministic AHSMs. Roughly speaking, an AHSM H is said to be deterministic if the 'next' configuration of H can be predicted from the 'current' configuration and the event executed by H. Formally, an AHSM H is *deterministic* iff $|\delta(q, \sigma)| \leq 1$ for every $q \in Q$ and $\sigma \in \Sigma$; otherwise it is called nondeterministic. The following proposition characterizes the determinism property in terms of the transition-paths of H.

Proposition 5: An AHSM H is nondeterministic iff there exists an OR-state $a \in A^+$ and two transition-paths $(u_i, \sigma, v_i)_a$, $i = 1, 2$, such that $v_1 \neq v_2$ and the tuple $\langle u_1, u_2 - u_1 \rangle$ is orthogonal.

Following Proposition 5, it can be verified that the AHSM of Fig. 10 is deterministic, whereas that of Fig. 12 is nondeterministic. The latter conclusion, for instance, follows from the fact that transition-paths $(\langle b \rangle, \rho, \langle c \rangle)_r$ and $(\langle e \rangle, \rho, \langle a \rangle)_r$ satisfy the condition for nondeterminism given in Proposition 5, namely, the condition that $\langle c \rangle \neq \langle a \rangle$ and that the tuple $\langle \langle b \rangle - \langle e \rangle, \langle e \rangle \rangle = \langle b, e \rangle$ is orthogonal. Alternatively, one can verify nondeterminism merely by definition, i.e., by examining the transition function of the equivalent state machine (Fig. 13). This approach, however, may be intractable for nontrivial examples, as compared with the former approach (the one suggested in Proposition 5), whose complexity is polynomial in the size of the transition-path set.

VI. Complexity Considerations

In the present section we shall examine aspects of the computational effort required for reachability computa-

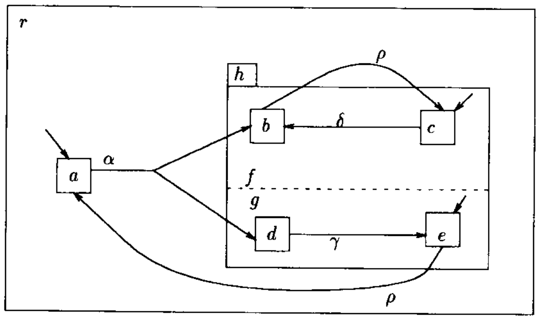


Fig. 12. An AHSM that is partially accessible but not accessible.

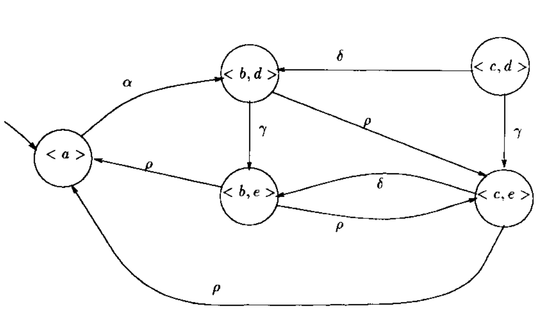


Fig. 13. The equivalent state machine of the AHSM in Fig. 12.

tions (using Algorithm 3) in an AHSM H, as well as for on-line control synthesis (Section IV). Specifically, we wish to compare this computational effort with the effort that would be required if the system were modeled as the equivalent state machine, whose state set coincides with the configuration set Q. To this end, we define the following class of standard AHSMs. A *standard* AHSM has a hierarchy tree of depth $l = 2m$, with *AND*-states at levels $2j$, $0 \leq j \leq m - 1$, *OR*-states at levels $2j + 1$, $0 \leq j \leq m - 1$, and basic states at level l. Thus, an AHSM is standard if A_j, the set of all states at level j, satisfies the condition that $A_o = \{r\}$, $A_l = A^{basic}$ and for $0 \leq j \leq m - 1$, $A_{2j} \subseteq A^{\perp}$ whereas $A_{2j+1} \subseteq A^{+}$. We assume further that each *AND/OR*-state has k immediate substates, and that the number of transition-paths in each *OR*-state is linear in the number of its immediate substates, (i.e., $|T^a| = O(k)$, $a \in A^+$). Some properties of standard AHSMs are given in the following lemma.

Lemma 6: Let H be a standard AHSM with parameters m and k.

i) The number n of basic states is given by $n := |A^{basic}| = k^{2m}$.

ii) The number of *AND*-states is $|A^{\perp}| = \sum_{j=0}^{m-1} k^{2j} = O(k^{2m-1})$.

iii) The number of *OR*-states is $|A^{+}| = \sum_{j=0}^{m-1} k^{2j+1} = O(k^{2m})$.

iv) A lower bound on the size of Q is $k^{k^m} = k^{\sqrt{n}}$.

v) The length (i.e., the number of elements) of any configuration of a state at level $2m - 2j$ or $2m - 2j - 1$, where $0 \leq j \leq m$, is k^j.

Next we consider the complexity of testing reachability by Algorithm 3.1, i.e., the complexity of testing (4) by (13).

Proposition 6: Given a full configuration q and a set P of configurations:

i) For every state $a \in A$, $|X_a(q)| = O(mk)$ and $|Y_a(P)| = O(mk + |P|)$, where $X_a(q)$ and $Y_a(P)$ are given by (5) and (6).

ii) The complexity of computing $X_a(q)$ and $Y_a(P)$ for all $a \in A$ is $O(mkn + m\sqrt{n}|P|)$.

iii) The complexity of computing $W_a(q, P)$ for all $a \in A$ is $O(m^2k^3n + mk^2n|P|)$.

Notice that the complexity of computing $W_r(q, P)$, the set corresponding to the root state, is $O(m^2k^3n + mk^2n|P|)$, since it requires the computation of $X_a(q)$, $Y_a(P)$ and $W_a(q, P)$ for every $a \in A$. Thus we have the following proposition.

Proposition 7: Assuming H is a standard AHSM, the complexity of testing (4) is $O(m^2k^3n + mk^2n|P|)$.

It is worth noting that this polynomial complexity must be compared with the (exponential) size of Q (since a 'direct' computation of (4) may require $|Q|$ computations). We turn now to the problem FCP considered in Section IV. Notice first that Observation 1 and Proposition 6 imply that (using Algorithm 3) the question whether FCP is solvable can be resolved in complexity $O(m^2k^3n + mk^2n|F|)$, where F is the set of forbidden configurations. As regards the 'on-line' synthesis of a minimally restrictive control as suggested by Algorithm 5, we assume that the sets $\hat{W}_a(\langle\ \rangle, F)$, $a \in A$, are pre-computed in accordance with remark 3 below Algorithm 5. Under this assumption we have the following proposition.

Proposition 8: Given a standard AHSM H and a full configuration q of H (where q represents the current configuration occupied by H), a minimally restrictive control $S(q)$ can be computed in complexity $O(m^3k^3n + m^2k^2n|F|)$.

VII. Conclusion

In this paper we examined a class of discrete event systems (DESs) modeled as asynchronous hierarchical state machines (AHSMs). For this class of DESs, we have provided an efficient method (Algorithm 3) for testing reachability which is an essential step in many control synthesis procedures. This method utilizes the asynchronous nature and hierarchical structure of AHSMs thereby illustrating the advantage of the AHSM representation as compared with its equivalent (flat) state machine representation (see Lemma 6-iv) and Proposition 7). An application of the method has been presented in Section IV where we proposed an 'on-line' minimally restrictive solution for the problem of maintaining a controlled AHSM within prescribed legal bounds. The 'on-line' nature of this solution is similar in spirit to the feedback control logic suggested in [10].

This work opens several directions for further research. The first one is extensions to synchronized HSMs, namely

HSMs that allow an explicit modeling of interaction between transition-paths of orthogonal components (e.g., broadcast synchronization [6] or prioritized synchronization [12]). It is noteworthy that for such HSMs, the control strategy proposed by Algorithm 5 solves the forbidden configuration problem (FCP) (Section IV), though it may be too restrictive. Stabilization (in the sense of [13], [14]) of HSMs is another research topic and it is currently under investigation.

Appendix (Proofs)

A. Preliminaries

Lemma 7: The set Q_a, $a \in A$, of all (basic) full configurations of a can be computed inductively (up the hierarchy) as follows:

1) For a basic state a, $Q_a = \{\langle a \rangle\}$.

2) For an *OR*-state $a \cong \bigcup_{i=1}^k a_i$, $Q_a = \bigcup_{i=1}^k Q_{a_i}$.

3) For an *AND*-state $a \cong a_1 \times \cdots \times a_k$, $Q_a = Q_{a_1} \times \cdots \times Q_{a_k}$.

Proof of Lemma 7: Follows from the definition of full configurations [see (1)]. ■

Lemma 8: Let q be a configuration of some state $a \in A$.

i) If $a \cong \bigcup_{i=1}^k a_i$ then there exists $i \le k$ such that $q|_{a_i} = q (\in Q_{a_i})$, whereas for $j \ne i$, $q|_{a_j} = \langle\ \rangle$.

ii) If $a \cong a_1 \times \cdots \times a_k$ then for all $i \le k$, $q|_{a_i} \in Q_{a_i}$.

Proof of Lemma 8: Follows from Lemma 7 and the definitions of restriction and orthogonality. ■

The following lemma summarizes some properties of the span operator.

Lemma 9: Let q be a configuration of a state a. Then

i) $\mathscr{S}_a(q) \subseteq Q_a$.

ii) $\mathscr{S}_a(q) = Q_a$ iff $q = \langle\ \rangle$.

iii) $\mathscr{S}_a(q) = \{q\}$ iff $q \in Q_a$.

iv) If $a \cong \bigcup_{i=1}^k a_i$ then there exists $i \le k$ such that $\mathscr{S}_{a_i}(q|_{a_i}) = \mathscr{S}_{a_i}(q) = \mathscr{S}_a(q)$, and for $j \ne i$, $\mathscr{S}_{a_j}(q|_{a_j}) = Q_{a_j}$.

v) If $a \cong a_1 \times \cdots \times a_k$ then $\mathscr{S}_a(q) = \mathscr{S}_{a_1}(q|_{a_1}) \times \cdots \times \mathscr{S}_{a_k}(q|_{a_k})$.

Proof of Lemma 9: Straightforward. ■

B. Proofs Related to Section III

Proof of Lemma 3:

(IF) First, for $j = 1, 2, \cdots, n$, the existence of the transition-path (u_j, σ_j, v_j) and the fact that $u_j \sqsubseteq w_{j-1}$ imply $v_j \in \delta_a(w_{j-1}, \sigma_j)$. That is, v_j is a-reachable from w_{j-1}. Furthermore, conditions 1) and 2) in this lemma and the fact that (see Lemma 1) reachability within a state implies reachability within its immediate superstate, imply that there exists a path of a that traverses the full configurations $q, w_o, v_1, w_1, \cdots, v_n, w_n$, concluding the 'if' part of this proof.

(ONLY IF) Assuming there exists w, with $p \sqsubseteq w$, such that w is a-reachable from q, there exists a path s of a that starts at q and ends at w. Then the scenario described in this lemma can be constructed from s by deleting (in s) every event σ (and the configuration preceding σ) that labels transition-paths of strict substates of a. ■

Proof of Lemma 4: The complexity of each step in Algorithm 1 is $O(|w|)$. Thus the overall complexity of this algorithm is $O(l \cdot |w|)$.

i) If a is a basic state then by the definition of the restriction operation

$$w|_a = \begin{cases} \langle a \rangle & \text{if } \langle a \rangle \sqsubseteq w \\ \langle\ \rangle & \text{otherwise.} \end{cases} \tag{19}$$

ii) If a is a state whose immediate substates are $a_1, \cdots, a_k$ then for every element v in w

$$\begin{aligned} \langle v \rangle \sqsubseteq w|_a &\Leftrightarrow a \vdash^* v \\ &\Leftrightarrow \exists 1 \le j \le k, a_j \vdash^* v \\ &\Leftrightarrow \exists 1 \le j \le k, \langle v \rangle \sqsubseteq w|_{a_j}. \end{aligned} \tag{20}$$

Thus,

$$w|_a = \langle w|_{a_1}, \cdots, w|_{a_k} \rangle, \tag{21}$$

where some of the restrictions $w|_{a_j}$, $i \le j \le k$, may be the empty tuple. Notice that (7), (8), (19) and (20) imply that for all $j = 0, 1, \cdots, l$ and for every state $a \in A_j$ we have that $w|_a \ne \langle\ \rangle$ iff $a \in C_j$. If a is an *AND*-state, the lemma follows immediately from (21), whereas if a is an *OR*-state then the orthogonality of w implies that there exists a unique substate a_i of a such that $w|_a = w_{a_i}$ and for every other substate a_m of a, $w|_{a_m} = \langle\ \rangle$, concluding the proof. ■

Proof of Lemma 5: Step 2) in Algorithm 2 computes $q|_a$ [and adds it to $X_a(q)$] for every state a such that $q|_a \ne \langle\ \rangle$ (by Lemma 4). In step 3) of Algorithm 2, for every level i, every state $b \in A_i$ and every configuration $d \in D_b$, the restriction $d|_a$ is added to $X_a(q)$ for every *strict* substate a of b (since the index j in (9) is decremented only if $j > i$). Thus, upon termination, for every state $a \in A$ the configurations in $X_a(q)$ are contributions of (destination) configurations of *strict* superstates of a [cf. (5)]. ■

For showing the correctness of Proposition 1 we shall need the following lemmas. The first lemma relates the input and output sets of a given state to the corresponding sets of its substates.

Lemma 10: Suppose $b \vdash a$ for $a, b \in A$. Then for $b \in A^+$

$$\begin{aligned} X_a(q) &= [X_b(q) \cup D_b]|_a \quad \text{and} \\ Y_a(P) &= [Y_b(P) \cup S_b]|_a \end{aligned} \tag{22}$$

whereas for $b \in A^\perp$

$$X_a(q) = [X_b(q)]|_a \quad \text{and} \quad Y_a(P) = [Y_b(P)]|_a. \tag{23}$$

Proof of Lemma 10: (22) is an immediate consequence of (5) and (6), whereas (23) follows from the fact that $D_b = S_b = \varnothing$ for every *AND*-state b. ■

Next we wish to justify the claim that the digraph $G_a(q, P)$, as defined in step 2) of Algorithm 3, is well defined in the sense that its edge set $E_a(q, P)$ is indeed a binary relation on $V_a(q, P)$, the state set of $G_a(q, P)$. Formally, *Lemma 11:* For each *OR*-state $a \in A$,

$$E_a(q,P) \subseteq V_a(q,P) \times V_a(q,P).$$

Proof of Lemma 11: First (see step 2) of Algorithm 3), $(u,v) \in E_a(q,P)$ implies

$$\exists (u,\sigma,v) \in T^a \quad \text{or} \quad \exists i \leq k \text{ s.t. } (u,v) \in W_{a_i}(q,P).$$

By the definition of S_a and D_a (see Subsection II-C) and (10),

$$\begin{aligned}(u,\sigma,v) \in T^a \Rightarrow (u,v) &\in S_a \times D_a \\ &\subseteq V_a(q,P) \times V_a(q,P).\end{aligned}$$

Also, for $1 \leq i \leq k$

$$\begin{aligned} W_{a_i}(q,P) &\subseteq X_{a_i}(q) \times Y_{a_i}(P) && \text{(by definition)} \\ &\subseteq [X_a(q) \cup D_a] \times [Y_a(P) \cup S_a] && \text{[by (22)]} \\ &\subseteq V_a(q,P) \times V_a(q,P) && \text{[by (10)].} \end{aligned} \quad (24)$$

The second inclusion in (24) uses the fact that for every configuration x of *OR*-state a, either $x|_{a_i} = \langle\ \rangle$ or $x|_{a_i} = x|_a$, where $a \vdash a_i$. ■

We proceed with the following inductive lemma.

Lemma 12: Let A be a state whose substates are a_i, $1 \leq i \leq k$, and assume that for all $1 \leq i \leq k$ and for all $(x_i, y_i) \in X_{a_i}(q) \times Y_{a_i}(P)$

$$(x_i,y_i) \in W_{a_i}(q,P) \quad \text{iff} \quad x_i \in R_{a_i}^{-1}\big(H, \mathscr{S}_{a_i}(y_i)\big). \quad (25)$$

Then for all $(x,y) \in X_a(q) \times Y_a(P)$

$$(x,y) \in W_a(q,P) \quad \text{iff} \quad x \in R_a^{-1}(H, \mathscr{S}_a(y)). \quad (26)$$

Proof of Lemma 12:

Case 1) The state a is an *AND*-state. Let $(x,y) \in X_a(q) \times Y_a(P)$, and define for $1 \leq i \leq k$

$$x_i = x|_{a_i}, \qquad y_i = y|_{a_i}.$$

Since $x \in X_a(q)$ is a full configuration of a [see (5)], x_i, its restriction to a_i, is nonempty for all $1 \leq i \leq k$. Moreover, $x_i \in X_{a_i}(q)$ [see (23)]. The configuration y, however, may be a partial configuration of a and, thus, either $y_i \in Y_{a_i}(P)$ or, alternatively, $y_i = \langle\ \rangle$. In the latter case, y_i is trivially a_i-reachable from x_i, i.e.,

$$x_i \in R_{a_i}^{-1}\big(H, \mathscr{S}_{a_i}(\langle\ \rangle)\big), \quad (27)$$

where $\mathscr{S}_{a_i}(\langle\ \rangle) = Q_{a_i}$. Thus, we have

$$(x,y) \in W_a(q,P)$$

$\Leftrightarrow$ [by (12)]

$$\forall 1 \leq i \leq k, \quad (x_i,y_i) \in W_{a_i}(q,P) \ \vee \ y_i = \langle\ \rangle$$

$\Leftrightarrow$ [by (25) and (27)]

$$\forall 1 \leq i \leq k \quad x_i \in R_{a_i}^{-1}\big(H, \mathscr{S}_{a_i}(y_i)\big)$$

$\Leftrightarrow$ [by Lemma 2 and Lemma 9-v)]

$$x \in R_a^{-1}(H, \mathscr{S}_a(y)).$$

Case 2) Suppose a is an *OR*-state, and let $(x,y) \in X_a(q) \times Y_a(P)$. By Lemma 3,

$$x \in R_a^{-1}(H, \mathscr{S}_a(y)) \quad (28)$$

iff There exist a sequence $a_{i_0}, a_{i_1}, \cdots, a_{i_n}$ of substates of a, a sequence $s = (u_1,\sigma_1,v_1),(u_2,\sigma_2,v_2),\cdots,(u_n,\sigma_n,v_n)$ of transition-paths of a, where for $j = 1,\cdots,n$, the tuple u_j is a configuration of $a_{i_{j-1}}$ and v_j is a full configuration of a_{i_j}, such that

$$x \in R_{a_{i_0}}^{-1}\big(H, \mathscr{S}_{a_{i_0}}(u_1)\big) \ \wedge \ v_n \in R_{a_{i_n}}^{-1}\big(H, \mathscr{S}_{a_{i_n}}(y)\big) \quad (29)$$

and

$$\forall 1 \leq j \leq n, \qquad v_j \in R_{a_{i_j}}^{-1}\big(H, \mathscr{S}_{a_{i_j}}(u_{j+1})\big). \quad (30)$$

Notice that a is an *OR*-state and, hence, $\mathscr{S}_{a_{i_n}}(y) = \mathscr{S}_a(y)$ [see Lemma 9-iv)]. It follows from the induction hypothesis (25) that (29) and (30) hold iff

$$(x,u_1) \in W_{a_{i_0}}(q,P) \ \wedge \ (v_n,y) \in W_{a_{i_n}}(q,P) \quad (31)$$

and

$$\forall 1 \leq j < n, \qquad (v_j,u_{j+1}) \in W_{a_{i_j}}(q,P). \quad (32)$$

Furthermore, (11), (31), (32) and the existence of the sequence s of transition-paths imply that the sequence $x, u_1, v_1, \cdots, u_n, v_n, y$ is a path in the digraph $G_a(q,P)$, defined in step 2) of Algorithm 3. Thus, we conclude that (28) holds iff y is reachable from x in digraph $G_a(q,P)$ iff (see step 2) in Algorithm 3) $(x,y) \in W_a(q,P)$. ■

Proof of Proposition 1: We use induction on the level j of states and claim that (26) holds for every $a \in A$ and $(x,y) \in X_a(q) \times Y_a(P)$. For a (basic) state of level $j = l$, the reachability is trivial, whereas for every (*OR*-) state a of level $j = l - 1$, the digraph $G_a(q,P)$ is an equivalent representation of state a and its associated transition-paths. Since Lemma 12 implies the induction step, the claim follows. That is, for every $a \in A$ and $(x,y) \in X_a(q) \times Y_a(P)$

$$(x,y) \in W_a(q,P) \quad \text{iff} \quad x \in R_a^{-1}(H, \mathscr{S}_a(y)). \quad (33)$$

In particular, (33) holds with respect to the root state. Finally, since $X_r(q) = \{q\}$ and $Y_r(P) = P$, Proposition 1 follows. ■

C. Proofs Related to Section V

Proof of Proposition 3: H is partially accessible iff [see (18)]

$$\forall a \in A^{basic} \qquad q_o \in R^{-1}(H, \mathscr{S}(\langle a \rangle))$$

iff [see (33)]

$$\forall a \in A^{basic} \qquad (q_o, \langle a \rangle) \in W(q_o, F).$$

Thus, the proposition follows. ■

Proof of Proposition 4: First we shall show that for every state $a \in A$

$$\forall x \in X_a(q_o), \qquad R_a(H,x) = Q_a.$$

Then the accessibility of H will follow from the fact that for the root state r, $X_r(q_o) = \{q_o\}$ and $Q_r = Q$.

Clearly, for a basic state a we trivially have that $R_a(H, \langle a \rangle) = \{\langle a \rangle\} = Q_a$. Let a be a state whose immediate substates are $a_1, \cdots, a_k$, and suppose for the induction step that

$$\forall 1 \le i \le k, \forall y \in X_{a_i}(q_o), \qquad R_{a_i}(H, y) = Q_{a_i}. \quad (34)$$

Now let $x \in X_a(q_o)$. For showing that indeed $R_a(H, x) = Q_a$, choose an arbitrarily full configuration $w \in Q_a$, and consider the following two cases.

Case 1) Suppose a is an *AND*-state. Then, by (23) and Lemma 8-ii)

$$\forall 1 \le i \le k, \qquad x|_{a_i} \in X_{a_i}(q_o) \text{ and } w|_{a_i} \in Q_{a_i}. \quad (35)$$

It follows from (35) and the induction hypothesis (34) that for all $i = 1, \cdots, k$, $w|_{a_i}$ is a_i-reachable from $x|_{a_i}$. By Lemma 2, w is a-reachable from x, implying that $R_a(H, x) = Q_a$.

Case 2) Suppose a is an *OR*-state, and let a_i be the (unique) substate of a satisfying $w|_{a_i} = w (\in Q_{a_i})$ [see Lemma 8-i)]. Since a is connected there exists a configuration $y \in X_{a_i}(q_o)$ such that y is reachable from x in digraph $G_a(q_o, \langle\ \rangle)$. By an argument similar to the proof of Lemma 12 [Case 2], it follows that y is a-reachable from x. Also, (35) implies that $w (\in Q_{a_i})$ is a_i-reachable from $y \in X_{a_i}(q_o)$. Since a_i-reachability implies a-reachability, we conclude that configuration w is a-reachable from configuration x, implying that $R_a(H, x) = Q_a$. ■

Proof of Proposition 5:

(IF) Let (u_i, σ, v_i), $i = 1, 2$, two transition-paths of a, such that $v_1 \neq v_2$ and $u = \langle u_1, u_2 - u_1 \rangle$ is orthogonal. Clearly, u is a configuration of H (since u is an orthogonal tuple), and let $q \in Q$ be a configuration satisfying $u \sqsubseteq q$. Since for $i = 1, 2$, $u_i \sqsubseteq u \sqsubseteq q$ it follows by the definition of the transition function δ that

$$p_i = \langle q - q|_a, v_i \rangle \in \delta(q, \sigma), \qquad i = 1, 2.$$

Furthermore, $v_1 \neq v_2$ implies $p_1 \neq p_2$. Thus, $|\delta(q, \sigma)| > 1$.

(ONLY IF) Suppose H is nondeterministic, i.e., there exists $q \in Q$, $a \in A$, $\sigma \in \Sigma^a$ and $p_1, p_2 \in Q$ such that $p_1 \neq p_2$ and $p_i \in \delta(q, \sigma)$, $i = 1, 2$. Then, by the definition of δ there exist transition-paths $(u_i, \sigma, v_i) \in T^a$, $i = 1, 2$, such that for $i = 1, 2$

$$u_i \sqsubseteq q \quad \text{and} \quad p_i = \langle q - q|_a, v_i \rangle.$$

Clearly, $p_1 \neq p_2$ implies $v_1 \neq v_2$. Also, $u_i \sqsubseteq q$ implies that $u = \langle u_1, u_2 - u_1 \rangle \sqsubseteq q$. That is, u is a partial configuration of H, and therefore, orthogonal. ■

D. Proofs Related to Section VI

Proof of Lemma 6:

i) At each level j, $0 \le j \le 2m$, there are k^j states, i.e., $|A_j| = k^j$. Thus, $n = |A_{2m}| = k^{2m}$.

ii) and iii) follow from the organization of *AND/OR*-states in the hierarchy tree.

iv) First we claim that for all $0 \le j \le m$

$$|Q_a| = \begin{cases} k\Sigma_{i=0}^{j} k^i - 1 \\ \quad \text{if } a \text{ is an } AND\text{-state of level } (2m - 2j) \\ k\Sigma_{i=0}^{j} k^i \\ \quad \text{if } a \text{ is an } OR\text{-state of level } (2m - 2j - 1). \end{cases} \quad (36)$$

The proof of (36) is as follows (we shall denote the upper and lower part of 36 by (36-up) and (36-low), respectively). For each (basic) state a at level $2m$, $|Q_a| = 1$, and for each *OR*-state a of level $2m - 1$, $|Q_a| = k$ (since each *OR*-state has k immediate substates). Thus, (36) holds for $j = 0$. Assume that (36-up) and (36-low) hold for states at levels $(2m - 2j + 1)$ and $(2m - 2j)$, respectively.

1) Consider an *AND*-state $a \cong a_1 \times \cdots \times a_k$ of level $2m - 2(j + 1) = 2m - 2j - 2$. In this case (see Lemma 1), $Q_a = Q_{a_1} \times \cdots \times Q_{a_k}$, and thus $|Q_a| = \Pi_{h=1}^{k} |Q_{a_h}|$. Since each substate a_h is an *OR*-state of level $2m - 2j - 1$, it follows from the induction hypothesis (36-low) that for all $h = 1, \cdots, k$

$$|Q_{a_h}| = k^{\Sigma_{i=0}^{j} k^i}.$$

Consequently,

$$|Q_a| = \left[k^{\Sigma_{i=0}^{j} k^i} \right]^k = k^{\Sigma_{i=0}^{j+1} k^i - 1}$$
$$= \text{(36-up), with } (j + 1) \text{ replacing } j. \quad (37)$$

2) Consider an *OR*-state $a \cong \bigcup_{h=1}^{k} a_h$ of level $2m - 2(j + 1) - 1 = 2m - 2j - 3$. In this case $Q_a = \bigcup_{h=1}^{k} Q_{a_h}$. Also, each substate a_h is an *AND*-state of level $2m - 2j - 2$, and thus (37) implies

$$|Q_a| = \sum_{h=1}^{k} |Q_{a_h}| = k k^{\Sigma_{i=0}^{j+1} k^i - 1} = k\Sigma_{i=0}^{j+1} k^i$$
$$= \text{(36-low), with } (j + 1) \text{ replacing } j.$$

This completes the proof of (36). Finally, for the root state at level 0 [i.e., $j = m$ in (36-up)], we have

$$|Q| = |Q_r| = k^{\Sigma_{i=0}^{m} k^i - 1} \ge k^{k^m} = k^{\sqrt{n}}.$$

v) A similar proof can be carried out. ■

Proof of Proposition 6:

i) It was assumed in Section VI that each *OR*-state has $O(k)$ transition-paths, and thus (see Subsection II-C) $|D_a| = |S_a| = O(k)$ for every $a \in A$. Furthermore, each state has at most m *OR*-superstates (since the hierarchy tree is of depth $2m$). Consequently, i) follows immediately from (5) and (6).

ii) First we consider the computation of the sets $X_a(q)$, $a \in A$, using Lemma 5 and Algorithm 2. Since $|A| = O(n)$ (see i), ii) and iii) in Lemma 6), and thus the complexity of step 1) in Algorithm 2 is $O(n)$. In step 2) of Algorithm 2 the restrictions $q|_a$, $a \in A$, are computed using Algorithm 1. Since q is a full configuration of the root state (which is at level 0), its length is $|q| = k^m$ [see Lemma 6-v)], and thus (see Lemma 4) the complexity of step 2) in Algorithm 2 is $O(mk^m) = O(m\sqrt{n})$. Regarding step 3) of Algorithm 2, if d is the destination configuration of a transition-path of an OR-state of level $2m - 2j - 1$, then the complexity of executing Algorithm 1 (with d as an input) and performing (9) is $O(mk^j)$ [see Lemmas 4 and 6-v)]. Since there are $k^{2m-2j-1}$ OR-states at level $(2m - 2j - 1)$, each having $O(k)$ transition-paths, the overall computation at level $(2m - 2j - 1)$ is $O(mk^j k k^{2m-2j-1}) = O(mk^{2m-j})$. Finally, the fact that there are m levels of OR-states and $\sum_{j=0}^{m-1} k^{2m-j} \leq k^{2m+1}$ imply that the overall complexity of step 3) in Algorithm 2, as well as that of computing $X_a(q)$ for all $a \in A$, is $O(mk^{2m+1}) = O(mkn)$.

The sets $Y_a(P)$, $a \in A$ are computed in a similar fashion. The computation of $P|_a$ for all $a \in A$ is of complexity $O(mk^m|P|)$, whereas the rest of the computation in (6) is of complexity $O(mk^{2m+1})$ (see the previous paragraph). Thus ii) follows.

iii) For a state $a \in A$ and a pair $(x, y) \in X_a(q) \times Y_a(P)$ the complexity of deciding whether $(x, y) \in W_a(q, P)$ is $O(k)$; for if a is an AND-state, $O(k)$ operations are required for testing (12) (notice that the various restrictions corresponding to x and y have already been computed during the construction of the sets $X_a(q)$ and $Y_a(P), a \in A$), and if a is an OR-state then (see step 2) in algorithm 3), one may test the reachability of y from x in digraph $G_a(q, P)$ within the region of states $Z = \{x\} \cup \{y\} \cup D_a \cup S_a$, where $|Z| = O(k)$ [since $|D_a| = |S_a| = O(k)$]. Thus, for a given state a, $W_a(q, P)$ is computed in complexity $O(k|X_a(q))|\,|Y_a(P)|) = O(m^2k^3 + mk^2|P|)$. Since there are at most k^{2m} AND/OR-states (and exactly k^{2m} basic states), the complexity of computing $W_a(q, P)$ for all $a \in A$ is $O(m^2k^3k^{2m} + mk^2k^{2m}|P|) = O(m^2k^3n + mk^2n|P|)$. ∎

Proof of Proposition 8: The proof of this proposition is based on the following observations (proved below).

i) The complexity of step 2) in Algorithm 5 is $O(mk^m)$ and $|A(q)| = O(mk^m)$.

ii) The overall complexity of verifying for every OR-state $a \in A(q)$ and every transition-path $(u, \sigma, v) \in T^a$ whether $u \sqsubseteq q$ is $O(mk^{m+1})$.

iii) The overall number of configurations p for which $\hat{W}(p, F)$ is computed in step 4) of Algorithm 5 is $O(mk^m + 1)$.

iv) Assuming $\hat{W}(\langle\ \rangle, F)$ is pre-computed, the complexity of computing $\hat{W}(p, F)$ for a given configuration p is $O(m^2k^{m+2} + mk^{m+1}|F|)$.

Now, observations i) and ii) imply that the complexity of steps 2) and 3) in Algorithm 5 is $O(mk^{m+1})$ whereas observations iii) and iv) imply that the complexity of step 4) is $O(mk^{m+1}[m^2k^{m+2} + mk^{m+1}|F|]) = O(m^3k^3n + m^2k^2n|F|])$, and the complexity bound in Proposition 8 follows. The proof of observations i)–iv) is as follows:

i) The bound $O(mk^m)$ follows from the fact that q consists of k^m elements (basic states) and from the complexity of Algorithm 1. The set $A(q)$ is, in fact, the set of all OR-states that are superstates of an element of q. Since each basic state has at most m OR-superstates, $|A(q)| = O(mk^m)$.

ii) Let $a \in A(q)$ be an OR-state of level $2m - 2j - 1$, $0 \leq j \leq m$, and let (u, σ, v) be a transition-path of a. By Lemma 6-v), the length of u is bounded by k^j (since u is a configuration of a), and thus the complexity of verifying whether $u \sqsubseteq q$ is $O(k^j)$. The number of OR-states $a \in A(q)$ at level $2m - 2j - 1$ is k^{m-j} (i.e., this is the number of OR-states at level $2m - 2j - 1$ that are superstates of an element in q), and thus the assumption $|T^a| = O(k)$ implies that the overall computation at level $2m - 2j - 1$ is $O(kk^jk^{m-j}) = O(k^{m+1})$. Since there are m levels consisting of OR-states, the overall complexity is $O(mk^{m+1})$.

iii) Follows from the fact that $|A(q)| = O(mk^m)$ [see observation i)] and the assumption $|T^a| = O(k)$ for every $a \in A$.

iv) Following remark 3 below Algorithm 3 and the assumption that $\hat{W}(\langle\ \rangle, F)$, $a \in A$, are pre-computed, given a state a and a configuration p satisfying $p|_a \neq \langle\ \rangle$, the computational complexity of $W_a(p, F)$ is $O(k|Y_a(F)|$ [see also the proof of proposition 6-iii)]. Notice that if $p|_a = \langle\ \rangle$, $\hat{W}_a(p, F) = \hat{W}_a(\langle\ \rangle, F)$ and no computation is required. Since $|p| \leq k^m$, where p is a configuration specified in step 3) of Algorithm 5, and since the number of states a for which $p|_a \neq \langle\ \rangle$ is $O(k^m)$, the complexity of computing $\hat{W}_a(p, F)$ for every $a \in A$ is $O(mk^mk|Y_a(F)|)$, where $|Y_a(F)| = O(km + |F|)$, [see Proposition 6-i)]. ∎

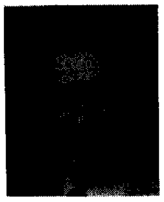
Acknowledgment

The authors would like to thank the reviewers for their constructive comments that helped in clarifying the presentation.

References

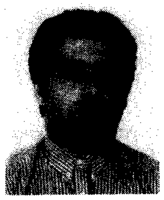
[1] P. J. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete event processes," *SIAM J. Contr. Optimiz.*, vol. 25, pp. 206–230, Jan. 1987.
[2] ——, "Modular Feedback Logic for Discrete Event Systems," *SIAM J. Contr. Optimiz.*, vol. 25, pp. 1202–1218, Sept. 1987.
[3] ——, "On the supremal controllable sublanguage of a given language," *SIAM J. Contr. Optimiz.*, vol. 25, pp. 637–659, May 1987.
[4] P. J. Ramadge, "Some tractable supervisory control problems for discrete-event systems modeled by Büchi automata," *IEEE Trans. Automat. Contr.*, vol. 34, pp. 10–19, Jan. 1989.
[5] J. N. Tsitsiklis, "On the control of discrete event dynamical systems," *Mathematics Contr. Signals Syst.*, vol. 2, pp. 95–107, 1989.
[6] D. Harel, "Statecharts: a visual formalism for complex system," *Sci. Comput. Programming*, vol. 8, pp. 231–274, 1987.
[7] D. Drusinsky, "On synchronized statecharts," Ph.D. dissertation, Weizmann Institute of Science, Rehovot, Apr. 1988.
[8] Y. Brave and M. Heymann, "Reachability in discrete event systems modeled as hierarchical state machines," in *Proc. 17th Convention IEEE Israel*, Tel-Aviv, Israel, May 1991.
[9] ——, "Control of discrete event systems modeled as hierarchical state machines," in *Proc. 30th Conf. Decision Contr.*, Brighton, UK, Dec. 1991.
[10] L. E. Holloway and B. H. Krogh, "Synthesis of feedback control logic for a class of controlled Petri nets," *IEEE Trans. Automat. Contr.*, vol. 35, pp. 514–523, May 1990.
[11] D. Harel, A. Pnueli, J. P. Schmidt, and R. Sherman, "On the formal semantic of statecharts," *IEEE Symp. Logic Comput. Sci.*, 1987.
[12] M. Heymann, "Concurrency and discrete event control," *IEEE Contr. Syst. Mag.*, vol. 10, pp. 103–112, Jan. 1990.
[13] Y. Brave and M. Heymann, "Stabilization of discrete-event processes," *Int. J. Contr.*, vol. 51, pp. 1101–1117, 1990.

[14] ——, "Optimal attraction in discrete event processes," *Inf. Sci.*, 1992.



Michael Heymann received the B.Sc. and M.Sc. degrees from the Technion, Haifa, Israel, in 1960 and 1962, respectively, and the Ph.D. degree from the University of Oklahoma, Norman, in 1965, all in chemical engineering.

From 1965 to 1966 he was on the Faculty of the University of Oklahoma. From 1966 to 1968 he was with Mobil Research and Development Corporation, engaged in research in control and systems theory. From 1968 to 1970 he was with the Ben-Gurion University of the Negev, Beer-Sheva, where he established and headed the Department of Chemical Engineering. Since 1970 he has been with the Technion where he is currently Professor in the Department of Computer Science holding the Carl Fechheimer Chair. He has previously been with the Department of Applied Mathematics of which he was Chairman and the Department of Electrical Engineering. He held visiting positions at various institutes, including the University of Toronto, the University of Florida, the University of Eindhoven, Concordia University, CSIR, Yale University, the University of Bremen, and the University of Newcastle, Australia. During 1983 to 1984, 1988 to 1989, as well as during the summers of 1985, 1986, 1987, 1990, 1991, and 1992, he was at NASA-Ames Research Center as an NRC-Senior Research Associate. His research covered topics in the areas of linear system theory, differential games, optimization and adaptive control. His current interest is chiefly in the area of discrete event systems and the theory of concurrent processes.



Yitzhak Brave received the B.Sc. (Hons.), M.Sc. and D.Sc. degrees in electrical engineering from the Technion—Israel Institute of Technology, Haifa, Israel, in 1984, 1987, and 1991, respectively.

He is currently a Research Associate in the Institut d'Automatique at the Swiss Federal Institute of Technology in Lausanne, Switzerland. His research interests include analysis and control of real-time systems, automation, robotics, Petri net models, VLSI testing and design.

Dr. Brave received the Wolf Foundation Prize, the Israeli Parliament (KNESSET) Prize and the special Gutwirth Prize for outstanding graduate students, in 1990, 1989, and 1988, respectively. In 1984/6/7/9 he was the recipient of the Award of the Distinguished Instructor from the Department of Electrical Engineering, Technion—I.I.T.