Proceedings of the 30th Conference
on Decision and Control
Brighton, England • December 1991

CONTROL OF DISCRETE EVENT SYSTEMS MODELED AS HIERARCHICAL STATE MACHINES

Y. Brave†
Department of Electrical Engineering
Technion - Israel Institute of Technology
Haifa 32000 , Israel.

M. Heymann‡
Department of Computer Science
Technion - Israel Institute of Technology
Haifa 32000 , Israel.

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.

1. 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. Theses models are conceptually appealing because of their inherent simplicity and the fact that they can be formally described by finite automata and their behavior by formal languages.

In many practical control problems, the discrete event system consists of a large number 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 [H87] 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.

In [D88] 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 then that of the equivalent state-machine model. The present paper deals with computational aspects of HSMs and focuses on asynchronous HSMs (AHSMs) where it is assumed that no interaction exists between parallel components. (All interaction and synchronization is assumed to be modeled in the control constraints.) It is shown that such pivotal issues as computation of reachability can be executed in the AHSM framework in exponential reduction of complexity as compared with the equivalent ordinary state machine representation of the process. We develop 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 for on-line control execution is developed. In the remainder of this section we shall give an informal description of HSMs. The formal structure of HSMs is presented in section 2, whereas section 3 provides an efficient algorithm for testing reachability in the HSM framework. An application of this algorithm is presented in section 4. A summary of sections 1-3 of the present paper appeared in [BH91].

States are represented by boxes. Hierarchy is represented by the insideness of boxes, as illustrated in Figure 1(a): where states a and e are *substates* of the state f and the states b, c and d are substates of a. Figure 1(b) represents the equivalent state machine of the HSM in Figure 1(a). The symbols $\alpha\eta$ stand for events associated with the various edges (transition-paths). 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 Figure 1(b)). *Default-arrows* indicate default states. In Figure 1(b), state e is selected as the initial state, and not a (a fact represented in 1a 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.

---

†Current address: EPFL Institut d'Automatique, 1015 Lausanne, Switzerland. E-mail: brave@elia.epfl.ch . Supported in part by the Wolf Foundation and the Fonds National Suisse de la Recherche Scientifique.

‡ Completed in part while the author was on leave as a NRC-Senior Research Associate at NASA-Ames Research Center, Moffett Field, CA 94035. Also supported in part by the Technion fund for promotion of research.

Orthogonality or concurrency is the dual of the *OR*-decomposition of states. In Figure 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. Edges *internal* to f, such as the transition labeled α, do not affect the g component. Thus, if α occurs at $<a,d>$, it affects only the f-component, resulting in $<b,d>$. An event λ at j causes entrance to combination $<a,d>$, and μ at $<b,e>$ causes transfer to i. The event θ from d states that the *AND*-state $h(=f \times g)$ is left and j entered, depending only on the fact that the g-component is actually at d. The η-arrow, on the other hand, leaves h unconditionally. By default arrows, event λ at i means entering $<b,e>$, whereas ρ means entering $<b,d>$. States that have no substates, such as a, b, d and e, are called *basic*. The tuple $<a,d>$, as well as $<b,e>$ and $<i>$, is a (basic) *configuration* of the HSM of Figure 2(a) and represents a set of orthogonal states which the HSM can occupy simultaneously. The set of all configurations of the HSM of Figure 2(a) is the set of all states in its equivalent 'flat' version of Figure 2(b). **Example 1.1:** A hierarchical processing of jobs. Consider the HSM H of Figure 3. The transition-paths of H represent the following actions.

n_o - take a job for processing.
a_i - partition job for further (hierarchical) processing.
c_i - start hierarchical processing.
b_i - perform 'direct' (non-hierarchical) processing.
d_i - store job.
e_i - continue processing.
f_i - test product.
k_i - combine products of higher level of processing.
h_i - test combined product.
g_i, l_i, m_i - processing failures.
k_0 - processing ended successfully.

Some of the events are uncontrolled in the sense that they cannot be prevented from occurring by external control. In Figure 3, 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 operations Partition and Combine at the same time. In other words, it may be required that states PARTi and COMBj are never occupied simultaneously. 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 an algorithm 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 3 and 4 we propose an alternative approach for solving FCP in the framework of AHSMs, based on efficient reachability computations and on-line synthesis of minimally restrictive supervisors. Other works related to the FCP problem are [R89] that considered a mutual exclusion problem in product systems, and [HK89] that proposed control policies for discrete event systems modeled as marked graphs.

2. 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 [D88]). First, we shall need the following notations. Let x, y and z be three tuples. We shall write $x \subseteq y$ iff every element of x is an element of y, e.g., $<a,b> \subseteq <a,c,b,d>$. We shall write $z = x - y$ iff z consists of all elements of x that do not appear in y, e.g., $<a,c,b,d> - <a,b,e> = <c,d>$.

2.1 states

A is the (finite) set of states of H, consisting of A^+, the subset of *OR*-states, A, 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 a.
3. A state $a \in A$ has no immediate substates if and only if a is basic.
4. If $b \vdash a$ then either $b \in A^+$ and $a \notin A^+$, or $b \in A$ and $a \in A^+$.

Assumption 4, *the alternating structure assumption*, means that the immediate substates of *OR*-states are either *AND*-states of basic states, whereas the immediate substates of *AND*-states are *OR*-states. 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.

2.2. Configurations

Let q be a tuple of (disjoint) states. (The examples throughout this subsection relate to Figure 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. E.g., $<b,e>|_f = <b>$.
- A state a is a *superstate* of q if every element of q is a substate of a. E.g., h and k are superstates of $<b,e>$.
- 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, $b \vdash^* a$. E.g., $LS(<b,e>) = h$.
- Two states q_1 and q_2 are *orthogonal*, denoted $q_1 q_2$, if either $q_1 = q_2$ or, alternatively, if neither is a superstate of the other and $LS(<q_1,q_2>) \in A$. A tuple of states q is *orthogonal* if every pair of states in q is orthogonal. E.g., $<b,e>$ is orthogonal, whereas $<b,h>$ and $<b,i>$ 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

$b \in A, \quad a \vdash^* b \Rightarrow \langle q, b\rangle$ is not orthogonal.

If q is a configuration that does not satisfy (2.1) then it is a *partial* configuration of a. E.g., $\langle b\rangle$ and $\langle b, g\rangle$ are, respectively, a partial and a full configuration of h. The configuration q is *basic* if all its elements are basic states. The set of all basic full configurations of a is denoted Q_a.

- Let q be a configuration of a state a and let be p be a full configuration of a such that $q \subseteq p$. The configuration p is called an *a-completion* of q. An a-completion p of q is *maximal* if for every state b such that $\langle b\rangle \subseteq p - q$, $d \vdash b$ implies that $\langle q, d\rangle$ is not orthogonal. E.g., $\langle b, g\rangle$ is a maximal k-completion of $\langle b\rangle$, whereas the k-completion $\langle b, e\rangle$ of $\langle b\rangle$ is not maximal. It can be shown that each configuration q of a state a has a unique maximal a-completion, denoted $c_a(q)$.
- For a basic configuration q of a state a, the *a-span* of q, denoted $C_a(q)$ is defined as the set of all basic full configurations p of a such that $q \subseteq p$. E.g., $C_k(\langle b\rangle) = \{\langle b, e\rangle, \langle b, d\rangle\}$. For a subset P of basic configurations of a, $C_a(P) = \bigcup_{p \in P} C_a(p)$.

2.3 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 configuration of a, called, respectively, the source and destination configurations of t, and $\sigma \in \Sigma$ is an event symbol that labels t. The association of t with T^a implies that in the associated HSM graph, a is the lowest OR-state containing t's source and destination configurations, as well as its entire arc. Thus, in Figure 2(a), the transition-path labeled α belongs to state f, whereas the θ-transition-path belongs to state k. A transition-path $(u, \sigma, v)_a$ is *canonical* if its source configuration is a basic configuration of a and its destination configuration is basic and full. 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, let S_a (D_a) denote the set of all source (respectively, destination) configurations of transition-paths of a. It was shown in [D88] that it is possible to transform an HSM to one in which every source configuration of a transition-path is basic. Henceforth, we shall assume that all HSMs have been thus transformed.

2.4 Default configurations

- The *default function* $\rho : A^+ \to A$ specifies for each OR-state an immediate substate, called its *default*.
- The *default configuration function* $\hat{\rho}$ specifies inductively for each state a a unique basic full configuration, called its *default configuration*, as follows:
  1. For an AND-state a with immediate substates $a_1, \ldots, a_k$,
     $$\hat{\rho}(a) = \langle \hat{\rho}(a_1), \ldots, \hat{\rho}(a_k)\rangle .$$
  2. For an OR-state a with immediate substates $a_1, \ldots, a_k$,
     $$\hat{\rho}(a) = \hat{\rho}(a_i) \quad \text{iff} \quad a_i = \rho(a) .$$
  3. For a basic state a, $\hat{\rho}(a) = \langle a\rangle$.

  In Figure 2(a), $\hat{\rho}(k) = \hat{\rho}(h) = \langle \hat{\rho}(f), \hat{\rho}(g)\rangle = \langle b, e\rangle$.
- Let q be a configuration of a state a, an(2.1b) $c_a(q) = = \langle c_1, \ldots, c_l\rangle$ be its maximal a-completion. The *default a-completion* of q, denoted $d_a(q)$, is then defined as the basic full configuration $\langle \hat{\rho}(c_1), \ldots, \hat{\rho}(c_l)\rangle$ of a (where $\hat{\rho}(c_i)$ is the default configuration of c_i as defined above).

2.5 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 $a, b \in A^+$

$$(u, \sigma, v) \in T^a \text{ and } (u', \sigma', v') \in T^b \quad \Rightarrow \quad \sigma \neq \sigma' .$$

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 $u \subseteq q$. 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 Figure 2(a), the transition-path $t = (\langle d\rangle, \theta, \langle j\rangle) \in T^k$ is defined at configuration $q = \langle b, d\rangle$, and if the AHSM H executes θ at q it enters configuration $p = \langle q - q|_k, j\rangle = \langle j\rangle$ (since $q|_k = q$). Formally, we associate with each state $a \in A$ the *transition function* $\delta_a : Q_a \times \Sigma \to 2^{Q_a}$ 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

$$u \subseteq q \quad \text{and} \quad p = \langle q - q|_b, d_b(v)\rangle .$$

where $d_b(v)$ is the default b-completion of v. The transition function of H is defined as $\delta = \delta_r$.

We interpret an AHSM $H = (A, \vdash, \Sigma, T, \rho)$ as a device that starts at configuration $q_o = \hat{\rho}(r)$ 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. For clarity we shall assume that transition-paths are given in their canonical form (see subsection 2.3); thus, henceforth, unless stated otherwise, all configurations are assumed to be basic.

3. Reachability

In this section we discuss the problem of testing reachability of a set of (full or partial) configurations from a given full configuration. Let $a \in A$. A *path* in a is a finite sequence $s = q_o, \sigma_1, q_1, \ldots, \sigma_n, q_n$, 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 $i = 1, 2, \ldots, n$. 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 $p \in P$, $p \subseteq w$, and w is r-reachable from q, i.e., to verify whether

$$q \in R_r^{-1}(H, C_r(P)) , \tag{3.1}$$

where $C_r(P)$ is the r-span of P. Next, we shall present an algorithm for testing (3.1) that does not require the construction of the equivalent state machine $M(H)$ whose state set is exponential in the number of orthogonal components in H. Let us begin with a simple example.

Example 3.1: Consider the AHSM H depicted in Figure 4. Let $q = <a_1,b_1>$ and $p = <a_2,b_2>$, 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 $<a_2>$ is a-reachable from $<a_1>$, and whether $<b_2>$ is b-reachable from $<b_1>$. Since the answer to the latter question is negative, we conclude that $<a_2,b_2>$ is not c-reachable from $<a_1,b_1>$.

Next we examine the effect of the transition-paths labeled by α, β, γ and δ of state f. Specifically, we wish to discover whether $p = <a_3,b_3>$ is f-reachable from $q = <a_1,b_1>$. 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 $<b_3>$ of δ is not c-reachable from $q = <a_1,b_1>$, we proceed with β whose source $<a_2>$ is c-reachable from q. Our final destination is $p = <a_3,b_3>$ (which is a configuration of c), and thus we continue with α, thereby entering configuration $<a_4,b_4>$. Now we check whether $p = <a_3,b_3>$ is c-reachable from $<a_4,b_4>$. Since this is not the case, we continue with δ, the only transition-path of f whose source $<b_3>$ is c-reachable from $<a_4,b_4>$, and return to state c through γ. This search terminates successfully since $p = <a_3,b_3>$ is c-reachable from the destination $<a_3,b_2>$ of γ. In summary, $p = <a_3,b_3>$ is f-reachable from $q = <a_1,b_1>$ via the following path: $<a_1,b_1>$, ψ, $<a_2,b_1>$, β, $<d>$, α, $<a_4,b_4>$, ρ, $<a_4,b_3>$, δ, $<e>$, γ, $<a_3,b_2>$, ρ, $<a_3,b_3>$.

We turn to our main goal of testing (3.1), but first consider again Figure 4. Recall that during the search performed in the paragraph preceding Lemma 3.3 for deciding whether the configuration $p = <a_3,b_3>$ is f-reachable from $q = <a_1,b_1>$, we checked whether p is c-reachable from q, from $<a_4,b_4>$ and from $<a_3,b_2>$, where the latter are configurations of c that are in D_f, the set of all destinations of transition-paths of f. In fact, a reachability test from $q, <a_4,b_4>$ and $<a_3,b_2>$ has been carried out also w.r.t. $<a_2>$ and $<b_3>$ 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 is a key point in the development of the algorithm below for reachability computations associated with (3.1).

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,

$$X_a(q) = \{ q|_a \} \cup \{ d|_a \mid d \in D_b \text{ and } b \vdash^+ a \} \quad (3.2)$$

Thus, in Figure 4, $X_c(<a_1,b_1>) = \{ <a_1,b_1> \} \cup \{ <a_4,b_4>, <a_3,b_2> \}$.
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

$$Y_a(P) = P|_a \cup \{ s|_a \mid s \in S_b \text{ and } b \vdash^+ a \} \quad (3.3)$$

Thus, in Figure 4, $Y_c(\{ <a_3,b_3> \}) = \{ <a_3,b_3> \} \cup \{ <a_2>, <b_3> \}$.

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 $Y_a(P)$. If a is an *AND*-state, these a-reachability tests are carried out separately and independently in each substate of a. If, however, a is an *OR*-state, we test a-reachability in the digraph $G_a(q,P)$ whose edge set consists of all transition-paths of a, and edges representing reachability within the substates of a. Figure 5 shows the digraph $G_f(q,p)$, where f is the root state of the AHSM in Figure 4, $q = <a_1,b_1>$ and $p = <a_3,b_3>$. 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 (3.1) can be accomplished by testing whether $W_r(q,P) = \varnothing$. Formally, we have the following algorithm for testing reachability.

Algorithm 1: 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, $W_a(q,P) = \varnothing$.

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

$$V_a(q,P) = X_a(q) \cup Y_a(P) \cup D_a \cup S_a ,$$

and whose edge set is

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

(3) For an *AND*-state a with immediate substates $a_1, \ldots, a_k$, and for all $(x,y) \in X_a(q) \times Y_a(P)$:
$(x,y) \in W_a(q,P)$ iff for each substate a_i either $(x|_{a_i}, y|_{a_i}) \in W_{a_i}(q,P)$ or $x|_{a_i} = < >$.

(4) Upon termination (at the root state r),

$$q \in R_r^{-1}(H, S_r(P)) \quad \text{iff} \quad W_r(q,P) \neq \varnothing .$$

Example 3.2: Consider the AHSM H of Figure 4, and suppose we wish to test whether $p = <a_3,b_3>$ is reachable from $q = <a_1,b_1>$. For applying the Algorithm above, we first compute the input and output sets: $X_f(q) = \{ q \}$, $Y_f(p) = \{ p \}$, $X_c(q) = \{ q, <a_4,b_4>, <a_3,b_2> \}$, $Y_c(p) = \{ p, <a_2>, <b_3> \}$, $X_a(q) = \{ <a_1>, <a_3>, <a_4> \}$, $Y_a(p) = \{ <a_3>, <a_2> \}$, $X_b(q) = \{ <b_1>, <b_2>, <b_4> \}$ and $Y_b(p) = (<b_1>, <b_3>)$ The digraphs $G_a(q,p)$ and $G_b(q,p)$ (see step (2) in the algorithm) consist of the transition-paths of states a and b, respectively. Thus

$$W_a(q,p) = \{ (<a_1>, <a_2>), (<a_3>, <a_3>), (<a_3>, <a_2>) \} .$$

and

$$W_b(q,p) = \{ (<b_1>, <b_1>), (<b_2>, <b_1>), (<b_2>, <b_3>), (<b_4>, <b_3>) \}$$

The digraph $G_f(q,P)$ is given in Figure 5, 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.

4. 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 sets, respectively. A *configuration feedback supervisor* (or in short *supervisor)* for an AHSM H is a map $S : Q \rightarrow 2^{\Sigma_c}$ that specifies for 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 $\xi : Q \times \Sigma \rightarrow 2^Q$ is defined as follows: For all $q, p \in Q$ and $\sigma \in \Sigma$

$$p \in \xi(q,\sigma) \quad \text{iff} \quad p \in \delta(q,\sigma) \text{ and } \sigma \notin S(q) \quad (4.1)$$

We now pose the following control synthesis problem.

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

$$R(S/H, q_o) \subseteq Q - C(F) \;, \qquad (4.2)$$

where $C(F)$ is the set of all (forbidden) full configurations of H spanned by F (see section 2.2).

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 $p \in C(f)$, with $f \in F$. It is clear that if H is at some configuration q and a forbidden full configuration $f \in C(F)$ 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 $C(F)$ can be reached via an uncontrolled path. It is clear that FCP is solvable iff $q_o \notin \Theta(F)$. An efficient test of the latter condition is obtained by modifying Algorithm 1 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:

Definition 4.1 *Let $\hat{W}(q,P)$ be defined as the result of Algorithm 1 modified as follows:*

(1) *The sets $\hat{D}_b$ and $\hat{S}_b$ replace the sets D_b and S_b, respectively in (3.2), (3.3) and step (2) of Algorithm 1.*

(2) The set T_u^a *(the subset of uncontrolled transition-paths of state a) replaces T^a in step (2) of Algorithm 1.*

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

Clearly, whenever FCP is solvable, it can be solved by S_Σ, the supervisor that disables all controlled events. However, S_Σ may be too restrictive in the sense that it eliminates controlled transitions whose deletion is not necessary for satisfying (4.2). Thus, we shall say that a supervisor S is a *minimally restrictive* solution of FCP if for every supervisor S' solving FCP, $R(S'/H, q_o) \subseteq R(S/H, q_o)$. Since the size of Q grows exponentially with the number of orthogonal components in H, an apriori ('off-line') synthesis of a minimally restrictive supervisor may be intractable. Thus we proceed according to the following *on-line* approach. 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 1.

5. 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 1) 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. An application of the method has been presented in section 4 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 [HK90].

This work opens several directions for further research. The first one is extensions to synchronized HSMs, namely HSMs that allow interaction between transition-paths of orthogonal components (e.g., broadcast synchronization [H87] or prioritized synchronization [He89]). Stabilization (in the sense of [BH90a, BH90b]) of HSMs is another research topic and it is currently under investigation.

References

[BH90a] Y. Brave and M. Heymann, stabilization of discrete event processes, *Int. J. Control*, vol. 51, no. 5, pp. 1101-1117, 1990.

[BH90b] Y. Brave and M. Heymann, On optimal attraction in discrete event processes, *Proc. European Control Conference*, July 1991.

[BH91] Y. Brave and M. Heymann, Reachability in discrete event systems modeled as hierarchical state machines, *Proc. 17th Convention of IEEE in Israel*, March 1991.

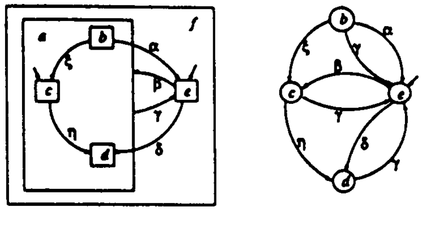
[D88] D., Drusinsky, On synchronized statecharts, *Ph. D. Dissertation, Weizmann Institute of Science*, Rehovot, April 1988.

[H87] D., Harel, Statecharts: A visual formalism for complex systems, *Science of Computer Programming*, 8 , pp. 231-274,1987.

[He89] M. Heymann, Concurrency and discrete event control, *IEEE Control Systems Magazine*, vol. 10, no. 4, pp. 103-112, 1989.

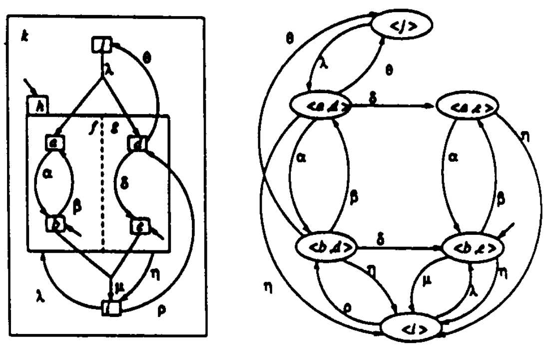
[HK90] L.E. Holloway and B.H. Krogh, Synthesis of feedback control logic for a class of controlled petri nets *IEEE Trans. on Automatic Control*, vol. 35, no. 5, May 1990.

[RW87] P.J. Ramadge and W. M. Wonham, Supervisory control of a class of discrete event processes, *SIAM J. on Control and Optimization*, 25(1), pp. 206-230, January 1987.



(a) (b)

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



(a) (b)

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

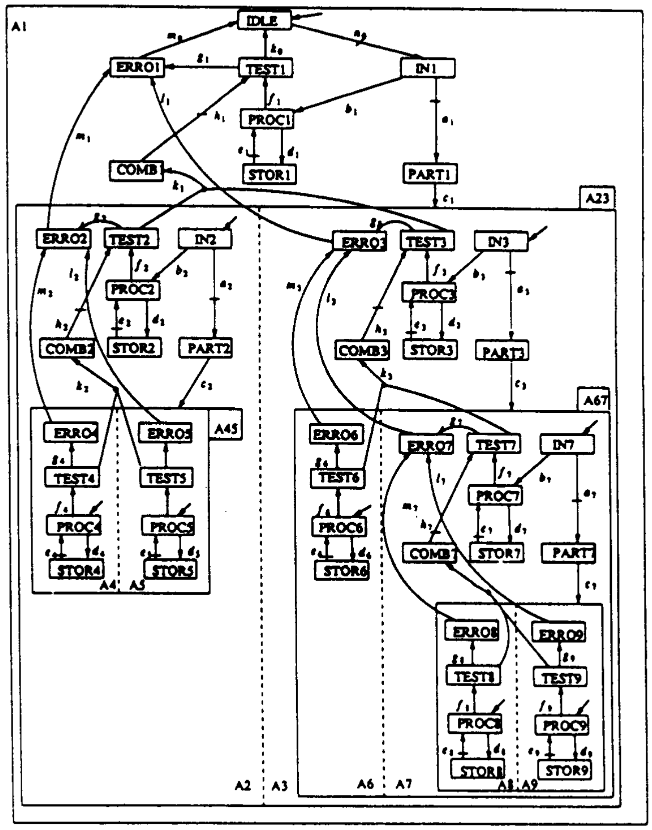


Figure 3: The full HSM of Example 1.1.

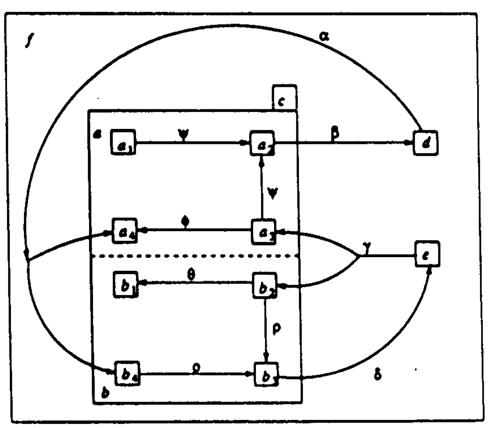


Figure 4. The HSM of Example 3.1.

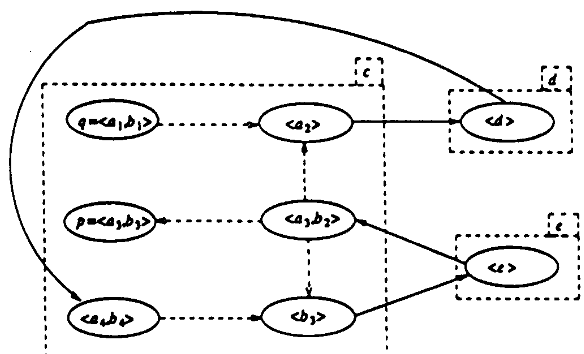


Figure 5: The digraph $G_f(q,p)$ of state f in Figure 4.