Computing Approximating Automata for a Class of Hybrid Systems

Alongkrit Chutinan
ac4c@andrew.cmu.edu

Bruce H. Krogh
krogh@ece.cmu.edu

Department of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA 15213-3890

Abstract

This paper presents a new algorithm for generating finite-state automata models of hybrid systems in which the continuous-state dynamics can be switched when the continuous-state trajectory generates threshold events. The automata state transitions correspond to threshold events and the automata states correspond to portions of the threshold surfaces in the continuous state space. The hybrid system dynamics are approximated by the automata models in the sense that the languages of threshold event sequences generated by the automata contain the threshold event language for the hybrid system. Properties of the algorithm for constructing and refining the approximating automata are demonstrated and the application of approximating automata for system verification is illustrated for a switching controller for an inverted pendulum. Relationships to other approaches to hybrid system synthesis and verification are also discussed.

1 Introduction

Although hybrid systems are extremely complex and difficult to analyze and characterize in general, it is often the case that the important features of the system behavior can be captured entirely in terms of sequences of events generated by the hybrid dynamics. For example, programmable logic controllers (PLC) typically observe the outputs of discrete (often binary) sensors to generate commands to actuators, such as motors, valves, and other devices that are operated in only a discrete set of modes. From the perspective of the PLC, the process being controlled generates discrete events in response to discrete input signals. Consequently, it is useful to eliminate the details of the continuous dynamics to synthesize the PLC program or verify the behavior of the closed-loop system for a given program implementation.

This paper concerns the generation of purely discrete models (finite automata) that generate the sequential behaviors of a class of hybrid systems. We propose an iterative method for constructing *approximating automata* for these systems. The events in the approximating automata correspond to threshold events in the hybrid system, that is, instants at which the continuous state encounters a hypersurface signaling a possible change in the continuous dynamics. The states in the approximating automata are associated with portions of the hypersurfaces that generate the threshold events. In this paper we focus on the sequences of threshold events. Given a hybrid system, we construct an automaton whose language contains all possible sequences of events that can be generated by the hybrid system. Properties of the approximating automaton can then be verified against some specification to deduce certain properties of the hybrid system using tools such as SMV [4] or COSPAN [7]. As such verification may fail due to the coarseness of the approximation, the resulting automaton can be further improved to give a tighter approximation of the behavior of the hybrid system. Alternatively, the approximating automaton can be used as a plant model for synthesizing a supervisor for the hybrid system [12, 5].

An algorithm for generating approximating automata for hybrid systems has been proposed and illustrated in [5, 9]. There are two problems with this algorithm that limit its usefulness for real problems. First, the algorithm relies on the existence of mappings that completely characterize the forward and backward evolution of families of continuous-state trajectories. Second, given an automaton based on the forward mappings of trajectories through sequences of length $N > 2$, the computation of a refinement of the approximation requires an extension of the forward and backward mappings through $2N-1$ events [10]. This paper addresses the second shortcoming of the algorithm in [5, 9]. In particular, a new class of approximating automata is proposed for which an approximation of order $N+1$ is obtained immediately from the approximation of order N through a refinement procedure.

The paper is organized as follows. The class of hybrid systems considered herein is defined in Sect. 2. Section 3 defines notations and mappings used in the algorithm for generating approximating automata. The algorithm is presented in Sect. 4, and the properties of the algorithm are demonstrated in Sect. 5. Section 6 presents an example application of the approximating automata approach to analyzing the properties of a switching controller for an inverted pendulum. The concluding section discusses recent related results directions for further research.

2 Problem Formulation

In this paper we consider the class of hybrid systems illustrated by the block diagram in Fig. 1 [5, 9]. The signal $u(\cdot)$ is a right-continuous piecewise constant *condition signal* taking on discrete values in the finite set U. This signal is generated as the state of a condition/event (C/E) [13] system whose state transitions are forced by the *event signal* $v(\cdot)$, a pointwise-nonzero signal taking on values in $\{0,1\}^{n_v}$. The initial state of the C/E system is given as $u(0^-) = u_0 \in U$. The C/E system state-transition function h is nondeterministic to include the possibility of alternative choices for $u(t)$, perhaps to be determined later by a supervisory controller. The set of events that drive transitions in the C/E system are the nonzero values of the event signal $v(\cdot)$, that is, values in the *threshold event set* $V = \{0,1\}^{n_v} - \{0\}^{n_v}$.

The continuous state trajectory $x(\cdot)$, evolves in $\in R^{n_x}$ according to the differential inclusion $\dot{x}(t) \in f_{u(t)}(x(t))$, where the condition signal $u(\cdot)$ selects the differential inclusion for the continuous dynamics at each instant. The *threshold events* which constitute the non-zero values of $v(\cdot)$ are generated as follows. Each output function g_i defines an event surface M_i so that $g_i(x) = 0$ if and only if $x \in M_i$. The vector function $g(\cdot)$ is applied to the signal $x(\cdot)$ to produce the output signal $y(\cdot)$. The resulting signal $y(\cdot) = g(x(\cdot))$ is in turn the input to a *zero detector* which generates the event signal $v(\cdot)$ according to

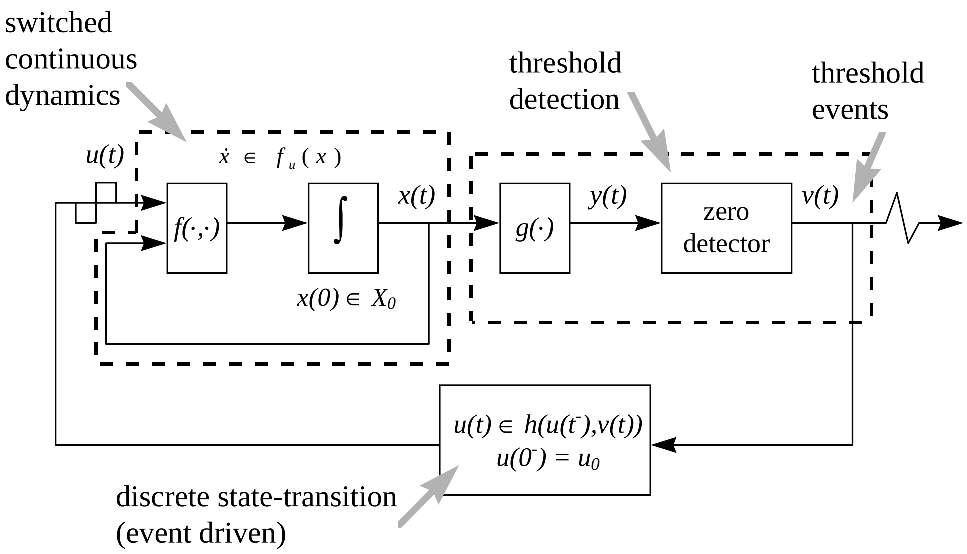


Figure 1: Block diagram illustrating the class of hybrid systems being considered

$$v_i(t) = \begin{cases} 1 & \text{if } y_i(t) = 0 \text{ and } \exists\triangle > 0 \text{ s.t. } y_i(t-\delta) < 0, \text{ for all } \delta \in (0, \triangle] \\ 0 & \text{otherwise} \end{cases}$$

Note that the zero detector is directional: an event is generated only when a component of $y(t)$ approaches zero from below. Also note that a distinct event is generated when more than one threshold is hit at a particular instant. This reflects the fact that when multiple thresholds are encountered simultaneously, the information about the location of the continuous system state is known more precisely than when any of the thresholds are encountered individually.

In general, the analysis of the behavior of a hybrid system as described above is virtually intractable since the input signal $u(\cdot)$ can switch at any instant of time. The scenario of interest is when the input condition signal $u(\cdot)$ changes values in response to threshold events observed in $v(\cdot)$. Thus, a signal pair $(u(\cdot), v(\cdot))$ is admissible only if discontinuities in $u(\cdot)$ occur only when $v(\cdot)$ is non-zero. This could represent a situation, for example, where the system is operating under closed-loop control and the signal $u(\cdot)$ is being generated by a discrete-event system (a C/E system) in which the transitions in $u(\cdot)$ are driven by the observed threshold events. To assure $u(\cdot)$ is piecewise constant with only a finite number of switches on any finite interval of time, we assume that $v(\cdot)$ is *nonexplosive* (the term used in the Markov chain literature) or “non-zeno”.

Following the development in [9], the discrete sequential behavior of a hybrid system H of the form described above is characterized by the language of the admissible sequences of threshold events. In general this language, denoted $L(H)$, contains strings of finite length and sequences of infinite length. Our interest is in creating an automaton that represents the sequential behavior of the hybrid system described above by accepting (or generating) the same language of threshold events. Such an automaton is useful either for verification of system properties, or for discrete controller synthesis, where the sequence of input condition values for $u(\cdot)$ is left unspecified and the language of the hybrid system is defined in terms of the pairs input-output values as described in [5].

3 Mappings of Families of Continuous Trajectories

Threshold events are generated by a set of n_v *threshold surfaces* denoted $M_1, \ldots, M_{n_v}$, where threshold surface M_i is defined by i^{th} component of the output function $g(\cdot)$; that is,

$$M_i = \{x \mid g_i(x) = 0\}.$$

The computations to create the approximating automata rely on the ability to compute the mappings of continuous state trajectories between threshold surfaces, both forward and backward in time.

Given a continuous state $x_0 \in R^{n_x}$ and a discrete state $u \in U$, we define the set of admissible continuous-state trajectories starting from x_0 under u as

$$\mathcal{X}(x_0, u) = \{x(\cdot) \mid x(0) = x_0 \text{ and } \dot{x}(t) \in f_u(x(t)) \text{ for all } t \geq 0\},$$

and let $ZD(y(\cdot))$ denote the resulting signal obtained when the (directional) zero detection operation is applied to the signal $y(\cdot)$.

The *forward dynamic mapping* is defined in terms of an initial continuous state x_0, a discrete state u, a set $T \subset R^{n_x}$, and an event vector $\hat{v} \in V$. The *forward dynamic mapping* from x_0 to T triggering $\hat{v}$ under u, denoted $\mathcal{M}_u(x_o \to T \uparrow \hat{v})$, is defined as the set of states in T that are reached and, when reached, generates the event $\hat{v}$. Formerly, $\mathcal{M}_u(x_o \to T \uparrow \hat{v})$ is defined as

$$\begin{aligned}\mathcal{M}_u(x_0 \to T \uparrow \hat{v}) = \{ \quad & x_f \in T \mid \exists x(\cdot) \in \mathcal{X}(x_0, u) \text{ and } t_f > 0 \text{ s.t. } x_f = x(t_f), \text{ and for} \\ & v(\cdot) = ZD(g(x(\cdot))), v(t_f) = \hat{v}, \text{ and } v(t) = 0, \ \forall 0 \leq t < t_f \ \}.\end{aligned}$$

We generalize the above definition to a set to obtain the forward dynamic mapping from a set of continuous states $S \in R^{n_x}$ to T triggering $\hat{v}$ under u as

$$\mathcal{M}_u(S \to T \uparrow \hat{v}) = \bigcup_{x_0 \in S} \mathcal{M}_u(x_0 \to T \uparrow \hat{v}).$$

The *backward dynamic mapping* from T to S under u triggering $\hat{v}$ is defined by

$$\mathcal{B}_u(T \uparrow \hat{v} \to S) = \{x_0 \in S \mid \mathcal{M}_u(x_0 \to T \uparrow \hat{v}) \neq \emptyset\}.$$

In words, $\mathcal{B}_u(T \uparrow \hat{v} \to S)$ identifies the part of S from which T can be reached first under u, triggering $\hat{v}$.

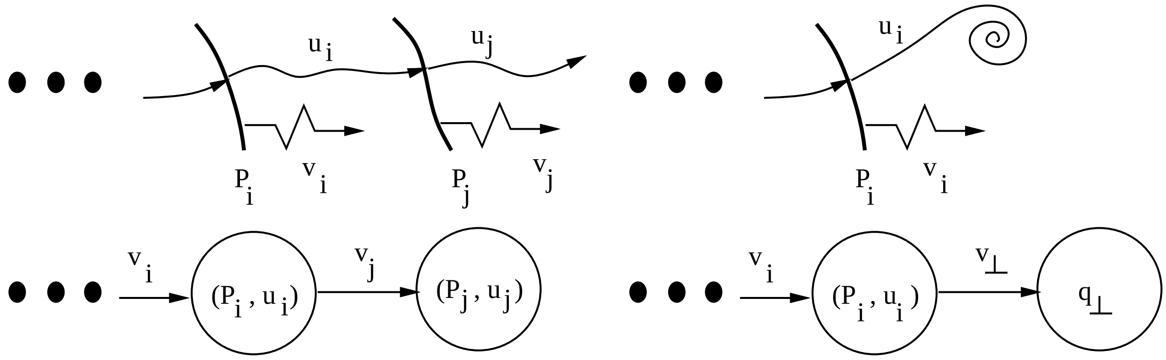
Finally, we define the *null event states* as the set of states from which there exists a trajectory which never triggers any event for a given discrete state u. Formally this set is given as

$$\mathcal{N}_u = \{x_0 \in R^{n_x} \mid \exists x(\cdot) \in \mathcal{X}(x_0, u) \text{ s.t. } v(\cdot) = ZD(g(x(\cdot))) \text{ and } v(t) = 0, \forall t \geq 0\}$$

4 Approximating Automata

Before proceeding to the precise definition of an approximating automaton, we first discuss the basic ideas of our approach. We maintain separate partitions of the threshold surfaces for each discrete state u. The partition for each u is denoted by $\mathcal{P}(u)$. Given such partitions, we define an approximating automaton A as follows. Except for states labeled (X_0, u_0) and $q_\perp$, each state of A is labeled by a pair (P, u) where $P \in \mathcal{P}(u)$. There are two types of transitions in the automaton, the *regular event* transitions, each labeled by $v \in V$, and the *null event* transitions, each labeled by $v_\perp$. As illustrated in Fig. 2a, a regular event transition v_j from (P_i, u_i) to (P_j, u_j) indicates there is possibly a continuous trajectory from a state in P_i to another state in P_j under u_i that triggers the event v_j, causing the discrete state to switch from u_i to u_j. Figure 2b illustrates a null event transition $v_\perp$ from (P_i, u_i) to $q_\perp$ indicating there is possibly a continuous trajectory under u_i from a state in P_i that never triggers any event. The *null state* $q_\perp$ is a special state that we use to represent the case where no more threshold events occur. (X_0, u_0) is the initial state.

To ensure that any subset S of a piece of an event surface P_i can generate any event that is possible for P_i, each partition $\mathcal{P}(u)$ needs to be *unambiguous*, which is defined as follows.



(a) regular event transition (b) null event transition

Figure 2: Two types of transition in an approximation automaton

Definition 1 Let $M = \bigcup_{i=1,\ldots,n_v} M_i$. A partition $\mathcal{P}$= $\{P_1, \ldots, P_s\}$ of M is *unambiguous* if

$$\forall i \in \{1, ..., s\} \text{ and } \forall k \in \{1, ..., n_v\}, \text{ either } P_i \subseteq M_k \text{ or } P_i \cap M_k = \emptyset$$

Definition 2 Given a set of unambiguous partitions $\mathcal{P}(u)$ of M for $u \in U$, we define the approximating automaton A as

$$A = (Q, \Sigma, \delta, (X_0, u_0))$$

where

$Q = \{(X_0, u_0), q_\perp\} \bigcup_{u \in U} \{(P, u) \mid P \in \mathcal{P}(u)\}$, the set of *states*;

$\Sigma = V \cup \{v_\perp\}$, the set of *output symbols*;

δ, the nondeterministic *transition function* is defined by:

1. $(P_j, u_j) \in \delta((X_0, u_0), v_j)$ if $\mathcal{M}_{u_0}(X_0 \to P_j \uparrow v_j) \neq \emptyset$ and $u_j \in h(u_0, v_j)$
2. $q_\perp \in \delta((X_0, u_0), v_\perp)$ if $\mathcal{N}_{u_0} \cap X_0 \neq \emptyset$
3. $(P_j, u_j) \in \delta((P_i, u_i), v_j)$ if $\mathcal{M}_{u_i}(P_i \to P_j \uparrow v_j) \neq \emptyset$ and $u_j \in h(u_i, v_j)$
4. $q_\perp \in \delta((P_i, u_i), v_\perp)$ if $\mathcal{N}_{u_i} \cap P_i \neq \emptyset$

and (X_0, u_0) is the *initial state.*

Letting $L(A)$ denote the collection of all possible sequences of events that can be generated by the automaton A, we show in Sect. 5 that $L(H) \subseteq L(A)$. Hence we call A an *outer approximation* of H. The number of strings in $L(A)$ that are not in $L(H)$ depends on the granularity of the partitions $\mathcal{P}(u)$, $u \in U$. The more refined the partitions are, the closer $L(A)$ approximates $L(H)$. We now present a procedure to obtain finer partitions $\mathcal{P}_{N+1}(u)$ from given partitions $\mathcal{P}_N(u)$, $u \in U$.

Let A_N be the automaton defined from the current partitions $\mathcal{P}_N(u)$, $u \in U$. We consider one partition at a time, that is, we consider $\mathcal{P}_N(u)$ for a particular u. Let $\mathcal{P}_N(u) = \{P_1, \ldots, P_s\}$. For each P_i, we identify all the successor states of (P_i, u) in A_N. We think of the transition to each successor state as a distinct behavior from (P_i, u) and try to isolate the subset of P_i that corresponds to each behavior. We use the backward mapping for this purpose. Suppose there is a transition v from (P_i, u) to (P', u') (note that P' belongs to $\mathcal{P}_N(u')$ which can be a different partition). In this case we are looking for the subset of P_i for which the system can transition under u to P', triggering the event v. The backward mapping $\mathcal{B}_u(P' \uparrow v \to P_i)$ gives us exactly this subset. Similarly, if there is a null event transition from (P_i, u), the subset of P_i corresponding to this behavior is simply $\mathcal{N}_u \cap P_i$.

By computing the backward mapping and the null event set from all successors of (P_i, u), we have a collection of subsets of P_i which exhibit different behaviors under u. We then make a subpartition of P_i itself by computing all intersections of the subsets (see the routine `disjoint` shown in Fig. 3). After we have proceeded through all $P_i \in \mathcal{P}_N(u)$, all subpartitions of the P_i form the new partition $\mathcal{P}_{N+1}(u)$. One can easily see that $\mathcal{P}_{N+1}(u)$ is at least refined as $\mathcal{P}_N(u)$.

We proceed in a similar manner to obtain $\mathcal{P}_{N+1}(u)$ for all u. The pseudocode for the refinement procedure is shown in Fig. 4. With the new set of partitions, we can define a new automaton A_{N+1}.

5 Analytical Results

In this section several properties of approximating automata and their refinements are demonstrated. In particular, we show: (1) the approximating automata generate languages that contain the language generated by the hybrid system; (2) the refinement algorithm leads to a reduction in the language of the approximating automaton; and (3) if a fixed point is reached in the refinement procedure using *exact* forward and backward mappings, the language of the approximating automaton then equals the threshold event language of the hybrid system.

```
Π' = disjoint(Π = {π_1, ..., π_r})
{
   while (∃π_i, π_j ∈ Π, i ≠ j, s.t.  π_i ∩ π_j ≠ ∅) {
     π_k = π_i ∩ π_j
     Π = Π ∪ {π_k}
     π_i = π_i − π_k
     /* if there are empty sets, remove them */
     if π_i = ∅
       Π = Π − {π_i}
     π_j = π_j − π_k
     if π_j = ∅
       Π = Π − {π_j}
   }
   return Π
}
```

Figure 3: Pseudocode for making given sets disjoint

5.1 Language Containment

In this section, we show that our scheme for constructing an approximating automaton A yields an outer approximation of the hybrid system H. We start by defining the languages generated by H and A and develop a series of lemmas leading to the final result that $L(H) \subseteq L(A)$.

Definition 3 Given a hybrid system H, we define the *H-transition system* A_H as

$$A_H = (S, V \cup \{v_\perp\}, \delta_H, S_0)$$

where

$S :=$ the state space $X = (R^{n_x} \times U) \cup \{q_\perp\}$

$V \cup \{v_\perp\} :=$ the output alphabet

$\delta_H :=$ the nondeterministic transition function δ_H for A_H is defined according to the following rules

1. For $v_j \in V$, $\delta_H((x_i, u_i), v_j)$ contains (x_j, u_j) if $\mathcal{M}_{u_i}(x_i \to \{x_j\} \uparrow v_j) \neq \emptyset$ and $u_j \in h(u_i, v_j)$
2. $\delta_H((x_i, u_i), v_\perp)$ contains $q_\perp$ if $x_i \in \mathcal{N}_{u_i}$
3. For all $v \in V \cup \{v_\perp\}$, $\delta_H(q_\perp, v) = \emptyset$

δ_H is extended for an input string $\sigma \in V^*\{v_\perp\}$ in the natural way.

$S_0 :=$ the set of initial states $S_0 = \{(x_0, u_0) \mid x_0 \in X_0\}$

```
𝒫_{N+1}(u) = refine(𝒫_N(u) = {P_1, ..., P_s} | 𝒫_N(u) is a partition of M)
{
  s = |𝒫_N(u)|
  𝒫' = ∅
  for i = 1 to s {
    k = 0
    /* check regular event transitions */
    for each v ∈ V
      for each (P', u') ∈ δ_N((P_i, u), v) {
        S = ℬ_u(P' ↑ v → P_i)
        /* add subset of P_i that goes to (P', u') */
        if (S ≠ ∅) {
          k = k + 1
          P'_ik = S
        }
      }
    /* check null event transition */
    S = 𝒩_u ∩ P_i
    /* add subset of P_i that goes to q_⊥ */
    if (S ≠ ∅) {
      k = k + 1
      P'_ik = S
    }
    /* call disjoint to compute subpartition of P_i
       and form 𝒫_{N+1}(u) from subpartitions of P_i's */
    𝒫' = 𝒫' ∪ disjoint({P'_i1, ..., P'_ik})
  }
  return 𝒫'
}
```

Figure 4: Pseudocode for the partition refinement procedure

Definition 4 The language generated by the hybrid system H, denoted $L(H)$, is defined as

$$L(H) = \{\sigma \in V^\omega \cup V^*\{v_\perp\} \mid \exists x_0 \in X_0 \text{ s.t. } \forall w \in pre(\sigma),\ \delta_H((x_0, u_0), w) \neq \emptyset\}$$

, where $pre(\sigma)$ is the set of all finite length prefixes of σ (including σ itself if σ is finite length).

Throughout this section we let $A = (Q_A, \Sigma, \delta_A, (X_0, u_0))$ be an approximating automaton defined from a set of unambiguous partitions $\mathcal{P}(u)$, $u \in U$. The language for an approximating automaton is defined similarly to the language for the hybrid system.

Definition 5 The *language* generated by an approximating automaton A, denoted $L(A)$, is defined as

$$L(A) = \{\sigma \in V^\omega \cup V^*\{v_\perp\} \mid \forall w \in pre(\sigma),\ \delta_A((x_0, u_0), w) \neq \emptyset\}$$

.

Lemma 1 Given $\sigma \in V^*$ and $x_0 \in X_0$, if $(x_j, u_j) \in \delta_H((x_0, u_0), \sigma)$, then there exists $(P_j, u_j) \in Q_A$ such that $(P_j, u_j) \in \delta_A((X_0, u_0), \sigma)$ and $x_j \in P_j$.

All proofs are given in the Appendix.

In words, Lemma 1 states that a state reached in the H-transition system corresponding to a particular event sequence will necessarily be in a state reached in the approximating automaton for the same event sequence. The following lemma states that when a terminal state is reached in the H-transition system, it will also be reachable in the approximating automaton.

Lemma 2 Given $\sigma \in V^*\{v_\perp\}$ and $x_0 \in X_0$, if $q_\perp \in \delta_H((x_0, u_0), \sigma)$, then $q_\perp \in \delta_A((X_0, u_0), \sigma)$.

These lemmas lead to the following result that states the approximating automata is indeed an outer approximation for the hybrid system in the sense of language containment.

Proposition 1 $L(H) \subseteq L(A)$.

5.2 Refinement

In this section, we consider the relationship between the an approximating automaton, A_N, defined from partitions $\mathcal{P}_N(u)$, and the automaton A_{N+1} defined from partitions $\mathcal{P}_{N+1}(u)$, where $\mathcal{P}_{N+1}(u)$ is obtained by applying the procedure `refine` to $\mathcal{P}_N(u)$. The goal of this section is to develop the result that $L(A_{N+1}) \subseteq L(A_N)$. Throughout this section, we let

$$\begin{aligned} A_N &= (Q_N, \Sigma, \delta_N, (X_0, u_0)) \\ A_{N+1} &= (Q_{N+1}, \Sigma, \delta_{N+1}, (X_0, u_0)) \end{aligned}$$

Note that the output alphabet and the initial state are the same for both automata. The proofs of the following lemmas and proposition closely parallel the development in Sect. 5.1.

Lemma 3 Given $\sigma \in V^*$, if $(P, u_j) \in \delta_{N+1}((X_0, u_0), \sigma)$ for some $(P, u_j) \in Q_{N+1}$, then there exists $(P', u_j) \in Q_N$ such that $(P', u_j) \in \delta_N((X_0, u_0), \sigma)$ and $P \subseteq P'$.

Lemma 4 Given $\sigma \in V^*\{v_\perp\}$, if $q_\perp \in \delta_{N+1}((X_0, u_0), \sigma)$, then $q_\perp \in \delta_N((X_0, u_0), \sigma)$.

Proposition 2 $L(A_{N+1}) \subseteq L(A_N)$.

This proposition shows that the refinement procedure always leads to a new approximating automaton that is at least as good as the previous approximation in the sense of language containment.

5.3 Fixed Point

In the previous section, we found that the automaton obtained after each refinement generates a more restrictive language, providing a tighter approximation to the discrete behavior of the hybrid system. This section discusses the implication when the refinement procedure does not yield any further refinement in the partitions of event surfaces. We show by the following lemmas and proposition that if the refinement procedure reaches a fixed point, then the language of the corresponding automaton is exactly the language of the hybrid system.

Lemma 5 Suppose $\mathcal{P}_{N+1}(u) = \mathcal{P}_N(u), \forall u \in U$ and let $A = (Q_A, \Sigma, \delta_A, (X_0, u_0))$ be the automaton defined from $\mathcal{P}_N(u)$, then $\forall\ (P_i, u_i) \in Q_A$,

1. if $(P_j, u_j) \in \delta_A((P_i, u_i), v)$, for some $v \in V$, then $\mathcal{B}_{u_i}(P_j \uparrow v \to P_i) = P_i$.
2. if $q_\perp \in \delta_A((P_i, u_i), v_\perp)$, then $\mathcal{N}_{u_i} \cap P_i = P_i$.

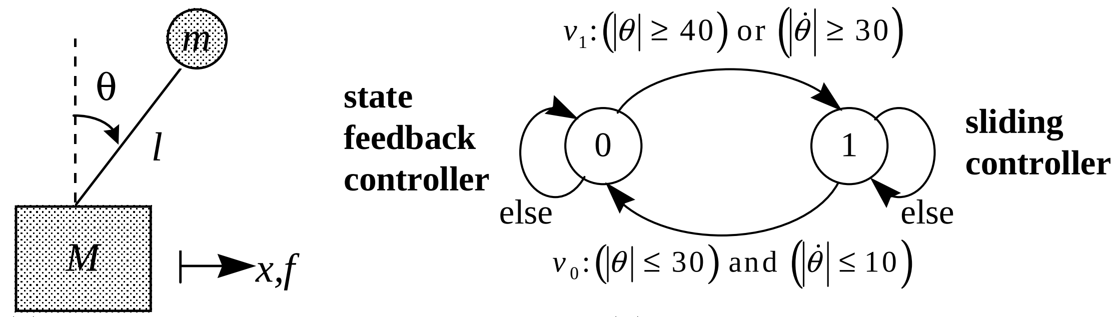
Lemma 6 Suppose $\mathcal{P}_{N+1}(u) = \mathcal{P}_N(u), \forall u \in U$ and let $A = (Q_A, \Sigma, \delta_A, (X_0, u_0))$ be an automaton defined from $\mathcal{P}_N(u)$. Given $\sigma \in V^*$, if $(P_j, u_j) \in \delta_A((X_0, u_0), \sigma)$ for some $(P_j, u_j) \in Q_A$, then there exists $(x_j, u_j) \in \delta_H((x_0, u_0), \sigma)$ for some $x_0 \in X_0$ such that $x_j \in P_j$.

Lemma 7 Suppose $\mathcal{P}_{N+1}(u) = \mathcal{P}_N(u), \forall u \in U$ and let $A = (Q_A, \Sigma, \delta_A, (X_0, u_0))$ be an automaton defined from $\mathcal{P}_N(u)$. Given $\sigma \in V^*\{v_\perp\}$, if $q_\perp \in \delta_A((X_0, u_0), \sigma)$, then $q_\perp \in \delta_H((x_0, u_0), \sigma)$ for some $x_0 \in X_0$.

Proposition 3 If $\mathcal{P}_{N+1}(u) = \mathcal{P}_N(u), \forall u \in U$, then $L(A_N) = L(H)$.

6 Example: Switching Control for the Inverted Pendulum

We consider the inverted pendulum system depicted in Fig. 5a. The system has three state variables, namely the pendulum angle θ, the angular velocity $\dot{\theta}$, and the speed of the cart $\dot{x}$. The system equations are shown below, where f is the control input, M is the mass of the cart, m is the mass of the pendulum, and l is the length of the pendulum.



(a) inverted pendulum (b) switching rule

Figure 5: Inverted pendulum system and switching rule

$$\begin{aligned}
\ddot{\theta} &= \frac{(M+m)gsin\theta - mlsin\theta cos\theta\dot{\theta}^2}{(M+msin^2\theta)l} - \frac{cos\theta}{(M+msin^2\theta)l}f \\
\ddot{x} &= \frac{mlsin\theta\dot{\theta}^2 - mgsin\theta cos\theta}{M+msin^2\theta} + \frac{1}{M+msin^2\theta}f
\end{aligned}$$

As it is difficult to design a single controller that can stabilize all state variables to zero from a large initial angle, two controllers are utilized. The first controller is a state feedback controller, which we will call *Controller 0*. It is designed for the linearized system and is effective when θ, $\dot{\theta}$, and $\dot{x}$ are small. The control law for this controller is shown below.

$$f_0 = -\left[k_1\theta + k_2\dot{\theta} + k_3 sat(\dot{x})\right],$$

where k_1, k_2, and k_3 are constant gains. As $\dot{x}$ may still be large when θ and $\dot{\theta}$ are small, we limit the contribution from $\dot{x}$ to a maximum magnitude of $\dot{x}_{max}$ with the saturation (sat) function.

The second controller is a sliding mode controller, which will later be referred to as *Controller 1*. Its purpose is to stabilize θ and $\dot{\theta}$ from a large initial angle θ_0. It ignores the cart velocity $\dot{x}$. The control law, which is a function of only θ and $\dot{\theta}$, is

$$\begin{aligned}
F(\theta,\dot{\theta}) &= \frac{(M_{max}+m_{max})g}{M_{min}l_{min}} + \frac{m_{max}\dot{\theta}^2}{M_{min}} \\
H(\theta,\dot{\theta}) &= \frac{(M_{max}+m_{max})l_{max}}{cos\theta} \\
f_1 &= H(\theta,\dot{\theta})\left[|F(\theta,\dot{\theta})| + |\lambda\dot{\theta}| + \eta\right] sgn(\dot{\theta}+\lambda\theta)
\end{aligned}$$

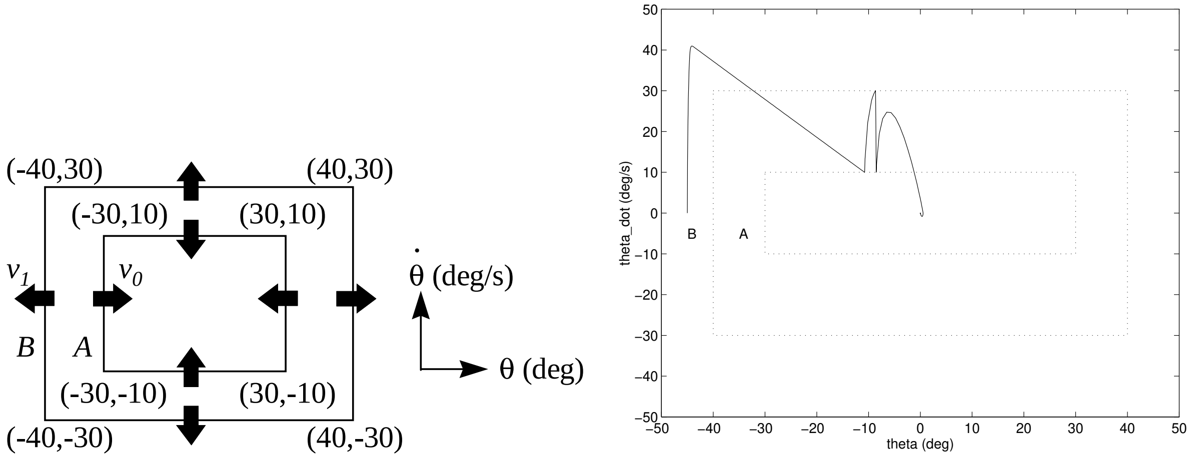
where λ and η are constants. The subscripts *min* and *max* denote the maximum bound and minimum bound on each of the system parameters, respectively. These bounds are introduced because a sliding mode controller is usually designed by assuming uncertainty in system parameters. In the subsequent simulation results, however, only the actual parameter values listed in Table 1 are used. The sign function $sgn(a)$ is defined to be -1 when $a < 0$, 1 when $a > 0$, and 0 when $a = 0$.

parameter	actual	min	max
M (kg)	4.8	4.5	5.5
m (kg)	0.52	0.45	0.55
l (m)	0.33	0.25	0.35
k_1	-135.70	-	-
k_2	-22.40	-	-
k_3	-15.68	-	-
$\dot{x}_{max}$ (m/s)	0.05	-	-
λ	1	-	-
η	0.001	-	-

Table 1: Parameter values for the inverted pendulum system

The rule to switch between the two controllers is shown in Fig. 5b. It corresponds to the purpose of each controller. When θ and $\dot{\theta}$ are small, Controller 0 is used, and when θ and $\dot{\theta}$ are large, Controller 1 is used. Numerical values for all parameters are given in Table 1.

This system fits into our formulation as follows. The continuous state is the pendulum state vector and the discrete state is the controller index (0 or 1). The continuous dynamics is selected by the controller index. Figure 6a illustrates the threshold event generation for this system. The event v_1 in Fig. 5, which causes the transition from Controller 0 to Controller 1, occurs when the continuous trajectory $(\theta, \dot{\theta})$ hits the rectangle B from the inside. Similarly, the event v_0, which causes the transition from Controller 1 to Controller 0, occurs when the continuous trajectory $(\theta, \dot{\theta})$ hits the rectangle A from the outside. Thus, A and B serve as event surfaces in our formulation. An example of a system trajectory is shown in Fig. 6b. The set of initial conditions for our system is $\theta(0) \in [-50, -45]$ deg, $\dot{\theta}(0) = 0$ deg/s, and $\dot{x}(0) = 0$ m/s. Since the system starts from a large initial angle, the initial controller index is 1.



(a) threshold surfaces

(b) an example trajectory

Figure 6: Event generation and control switching in the inverted pendulum system

We wish to verify that the system does not switch indefinitely between the two controllers, and that the final controller is always Controller 0. This specification can be written as a specification language $L_{spec} = (v_0 v_1)^* v_0 v_\perp$, and the verification problem reduces to checking whether $L(H) \subseteq L_{spec}$.

As shown in Proposition 1, the language generated by an approximating automaton $L(\mathcal{A})$ always contains $L(H)$, i.e. $L(H) \subseteq L(\mathcal{A})$. If we can find an approximating automaton $\mathcal{A}$ such that $L(\mathcal{A}) \subseteq L_{spec}$, then $L(H) \subseteq L_{spec}$ and we have the desired verification result. (We use $\mathcal{A}$ to denote an approximating automaton in this example to avoid confusing the automaton with the event surface A.)

To simplify the mapping computations, we project the continuous dynamics of the system into the $\theta - \dot{\theta}$ plane so that all the mappings defined in Sect. 3 can be computed by simulating bounding trajectories which start from some end points. We do not need to do anything for Controller 1 as the closed-loop dynamics of θ and $\dot{\theta}$ under Controller 1 is already decoupled from $\dot{x}$. For Controller 0, the projection is achieved through the use of two extremal systems whose saturation term in the control law is replaced by $\dot{x}_{max}$ and $-\dot{x}_{max}$.

The construction of an initial automaton and its subsequent refinements proceeds as follows. First, we must define initial partitions $\mathcal{P}_0(u)$ for each controller index u to construct an initial automaton $\mathcal{A}_0$. Here, we make some simplifications for the partitions by looking at the switching rule in Fig. 5b. In the switching rule, when the discrete state $u = 1$, the event v_1 is ignored. Thus, whenever v_0 is triggered, the discrete state u after the transition must be 0. Similarly, since v_0 is ignored when $u = 0$, $u = 1$ after a transition that triggers v_1. From the above discussion, we can see that the pairs $(A, 1)$ and $(B, 0)$ are not possible. Thus, we will only maintain a partition of A (instead for both A and B) for Controller 0 and a partition of B for Controller 1.

Iteration 0: We select the trivial initial partitions $\mathcal{P}_0(0) = \{A\}$ and $\mathcal{P}_0(1) = \{B\}$ as shown in Fig. 7a. From Fig. 6b, we see that there is a trajectory from $(X_0, 1)$ to $(A, 0)$, from $(A, 0)$ to $(B, 1)$, from $(A, 0)$ to $q_\perp$, and from $(B, 1)$ to $(A, 0)$. Thus, by Definition 2, we can define an automaton $\mathcal{A}_0$ shown in Fig. 7b. As $\mathcal{A}_0$ can generate an infinite length sequence $(v_0 v_1)^\omega$, it does not satisfy the specification. We then proceed with the refinement procedure to obtain the next automaton $\mathcal{A}_1$.

Iteration 1: From the state transition diagram of $\mathcal{A}_0$, we see that $(B, 1)$ has only one successor. Therefore, the refinement procedure will not refine the partition for Controller 1 in this step and $\mathcal{P}_1(1) = \mathcal{P}_0(1) = \{B\}$. For Controller 0, $(A, 0)$ has two successors, namely $(B, 1)$ and $q_\perp$. From simulation, we found that $\mathcal{B}_0(B \uparrow v_1 \to A) = A_1 \cup A_2 \cup A_4 \cup A_5 \cup A_6 \cup A_8$, and $\mathcal{N}_0 \cap A = A_2 \cup A_3 \cup A_4 \cup A_6 \cup A_7 \cup A_8$, where A_i's are shown in Fig. 7c. When the `disjoint` procedure is applied to $\{\mathcal{B}_0(B \uparrow v_1 \to A), \mathcal{N}_0 \cap A\}$, we obtain the partition $\mathcal{P}_1(0) = \{A_1, \ldots, A_8\}$. Applying Definition 2, we obtain the automaton $\mathcal{A}_1$ shown in Fig. 7d. Since it is clear that for any subset of A, the controller index must be 0, and that for any subset of B, the controller index must be 1, controller indices are dropped from the state labels. Similarly, the event labels are dropped from the transitions because we can identify unambiguously the event that is generated when any subset of A or B is reached. We see that there are cycles in $\mathcal{A}_1$ still, therefore we iterate the refinement procedure one more time.

Iteration 2: We look at the automaton $\mathcal{A}_1$. In this iteration, no refinement is obtained for any $A_i, 1 \leq i \leq 8$ for the following reason. For any A_i, we found from simulation that $\mathcal{B}_0(B \uparrow v_1 \rightarrow A_i) = A_i$ if there is a transition $(A_i, 0) \rightarrow (B, 1)$ and $\mathcal{N}_0 \cap A_i = A_i$ if there is a transition $(A_i, 0) \rightarrow q_\perp$. Thus, when we collect the backward mappings from all successors of $(A_i, 0)$ and the null event set in A_i, we will just have $\{A_i, \ldots, A_i\}$ and the `disjoint` procedure will only return $\{A_i\}$ as the new partition of A_i for Controller 0. Thus, $\mathcal{P}_2(0) = \mathcal{P}_1(0)$.

For controller 1, $(B, 1)$ has 8 successors, $(A_1, 0), \ldots, (A_8, 0)$. The backward mapping found from simulation are $\mathcal{B}_1(A_i \uparrow v_0 \rightarrow B) = B_i$, where B_i are shown in Fig. 7e. Since all B_i are already disjoint, we have that $\mathcal{P}_2(1) = \{B_1, \ldots, B_8\}$. We then apply Definition 2 to obtain the automaton $\mathcal{A}_2$ in Fig. 7f. We see that cycles no longer exist in $\mathcal{A}_2$ and all paths from X_0 go through some A_i before entering $q_\perp$. Thus, we have that the specification is verified for this automaton.

7 Discussion

This paper presents a method for creating automata that approximate the sequential behavior of hybrid systems characterized by threshold events, and an algorithm for refining a given approximation. In Sect. 5.3, we have shown that if the refinement procedure reaches a fixed point, then the language of the automaton is exactly the language of the hybrid system. Such a fixed point may not exist, however, since an infinite state discrete model may be needed to realize the event language of the hybrid system (see [8] and the references therein for recent results regarding finite-state representations of hybrid systems).

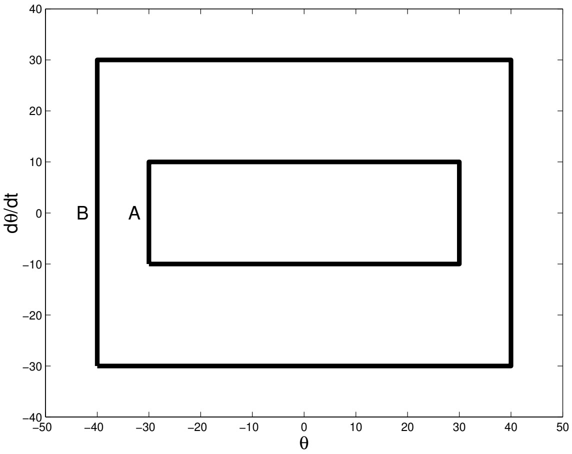
The major obstacle towards applying this automata construction and refinement methodology to any real system is the lack of efficient methods for computing the mappings of families of continuous trajectories. For two dimensional systems, we can compute the mappings between surfaces by simulating bounding trajectories, as shown in the inverted pendulum example. In higher dimensions, simulating a finite number of trajectories is insufficient for arbitrary continuous dynamics. Several researchers have been working on methods for approximating flows of continuous systems (see, e.g., [11] and [6]). A method was developed recently for computing conservative approximations to families of continuous trajectories for deterministic state equations (rather than differential inclusions) in arbitrary dimensions using sequences of convex polyhedral sets [3]. The attractive features of this technique are that the approximation error does not grow with simulation time, and the error can be as small as desired, albeit at the expense of more computation time.

For a class of so-called linear hybrid systems similar to the *linear hybrid automata*, the forward, backward mappings, and the null event set can be computed [2]. The method is based on the polyhedral computation that the verification tool HyTech [1] employs. For linear hybrid automata, the approximating automata approach offers an alternative to Hytech for verification. The HyTech verification approach is based on computation of the fixed point of the exact reachability region in the state space. The advantage of the automaton refinement procedure is that it always terminates at each stage, whereas the fixed point computation in HyTech may not terminate. As a result, there are some cases where the fixed-point computation in HyTech does not terminate, but the verification problem can be answered using approximating automata.

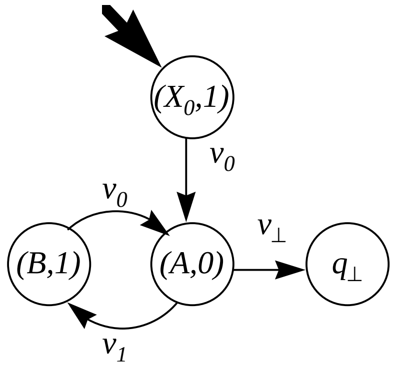
Open research issues include conditions under which the non-zeno assumption is guaranteed to hold, and methods for identifying what parts of the approximating automaton should be refined to reduce computation and state explosion.

Acknowledgments

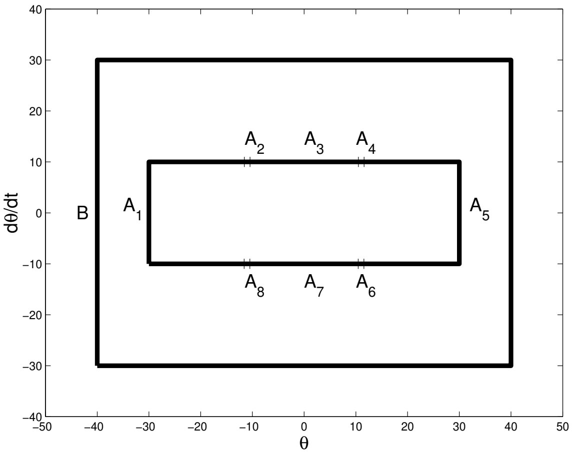
This research has been supported by Rockwell, Ford, and DARPA under contract number F33615-97-C-1012.



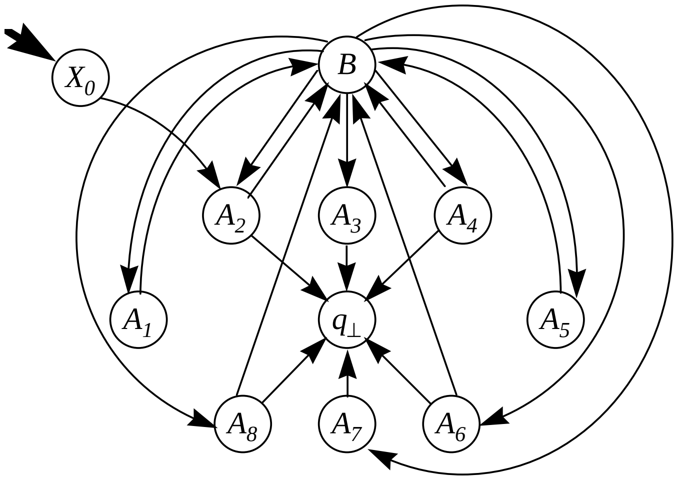
(a) Initial partitions



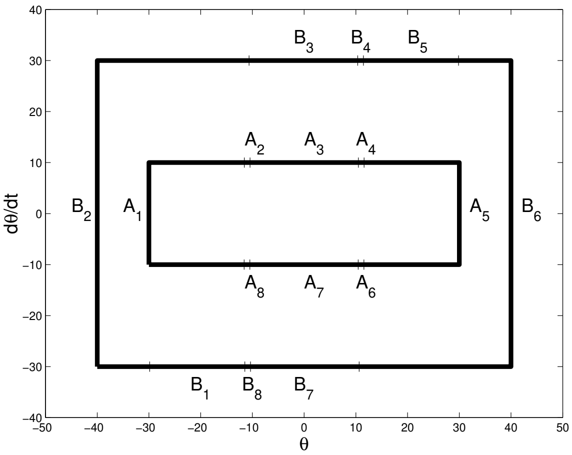
(b) Initial automaton $\mathcal{A}_0$



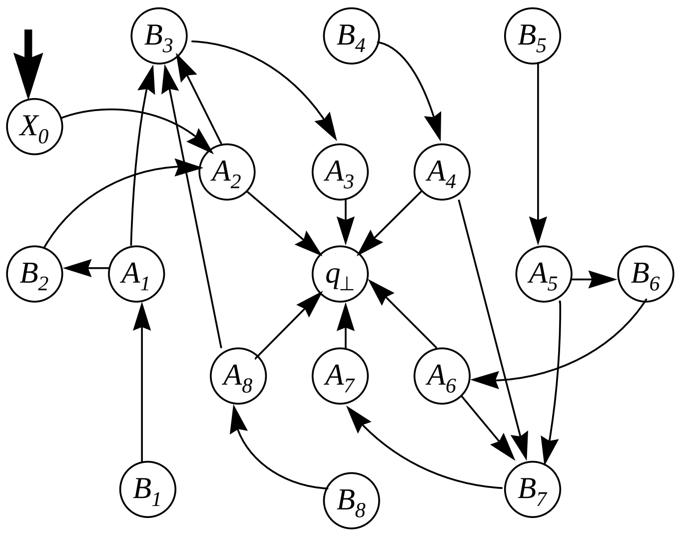
(c) Partitions after 1 iteration



(d) Automaton $\mathcal{A}_1$ after 1 iteration



(e) Partitions after 2 iterations



(f) Automaton $\mathcal{A}_2$ after 2 iterations

Figure 7:

Appendix

Proof of Lemma 1. We prove this lemma by induction on the length of the input string σ

Basis Step: $|\sigma| = 0$, i.e. $\sigma = \epsilon$. In this case, the lemma is true because $\delta_H((x_0, u_0), \epsilon) = \{(x_0, u_0)\}$, $\delta_A((X_0, u_0), \epsilon) = \{(X_0, u_0)\}$ and $x_0 \in X_0$.

Induction Step: Suppose that the lemma is true for all strings of length up to n. Let $\sigma = wv$ where $|w| = n$ and $v \in V$. We also assume that $\delta_H((x_0, u_0), wv) \neq \emptyset$ (since the lemma will be trivially true otherwise). For any $(x_j, u_j) \in \delta_H((x_0, u_0), wv)$, we can find an intermediate state (x_i, u_i) such that $(x_i, u_i) \in \delta_H((x_0, u_0), w)$ and $(x_j, u_j) \in \delta_H((x_i, u_i), v)$. By the induction hypothesis, we have that there exists $(P_i, u_i) \in Q_A$ such that $(P_i, u_i) \in \delta_A((X_0, u_0), w)$. By Definition 3, there is a continuous trajectory from x_i to x_j that triggers v at x_j, causing the discrete state to switch from u_i to u_j. Since x_j must be on an event surface, we can find a P_j in the surface partition $\mathcal{P}(u_j)$ such that $x_j \in P_j$. Since $x_i \in P_i$ and $x_j \in P_j$, we have that $(P_j, u_j) \in \delta_A((P_i, u_i), v)$. Finally, $(P_i, u_i) \in \delta_A((X_0, u_0), w)$ and $(P_j, u_j) \in \delta_A((P_i, u_i), v)$ imply that $(P_j, u_j) \in \delta_A((X_0, u_0), wv)$. Thus, the lemma is true for the string of length $n + 1$.

■

Proof of Lemma 2. Let $\sigma = wv_\perp$. Suppose $q_\perp \in \delta_H((x_0, u_0), wv_\perp)$. We can find an intermediate state $(x_i, u_i) \in \delta_H((x_0, u_0), w)$ such that $q_\perp \in \delta_H((x_i, u_i), v_\perp)$. By Lemma 1, we have that there is a $(P_i, u_i) \in Q_A$ such that $(P_i, u_i) \in \delta_A((X_0, u_0), w)$ and $x_i \in P_i$. Because $q_\perp \in \delta_H((x_i, u_i), v_\perp)$, it follows that $x_i \in \mathcal{N}_{u_i}$. Consequently, $P_i \cap \mathcal{N}_{u_i} \neq \emptyset$ and we have that $q_\perp \in \delta_A((P_i, u_i), v_\perp)$. Finally, $(P_i, u_i) \in \delta_A((X_0, u_0), w)$ and $q_\perp \in \delta_A((P_i, u_i), v_\perp)$ imply that $q_\perp \in \delta_A((X_0, u_0), wv_\perp)$.

■

Proof of Proposition 1. Suppose $\sigma \in L(H)$, then for some $x_0 \in X_0$ we consider any $w \in pre(\sigma)$. By Definition 4, $\delta_H((x_0, u_0), w) \neq \emptyset$. There are two possible types of states in $\delta_H((x_0, u_0), w)$.

Case 1 If $(x_i, u_i) \in \delta_H((x_0, u_0), w)$, then there exists some $(P_i, u_i) \in \delta_A((X_0, u_0), w)$ by Lemma 1.

Case 2 if $q_\perp \in \delta_H((x_0, u_0), w)$, then $q_\perp \in \delta_A((X_0, u_0), w)$ by Lemma 2.

In both cases, $\delta_A((X_0, u_0), w)$ is not empty for any prefix w of σ. Consequently, we have that $\sigma \in L(A)$.

■

Proof of Lemma 3 We prove this lemma by induction on the length of the input string σ.

Basis Step: $|\sigma| = 0$, i.e. $\sigma = \epsilon$. The lemma is true because $\delta_{N+1}((X_0, u_0), \epsilon) = \delta_N((X_0, u_0), \epsilon) = \{(X_0, u_0)\}$ and $X_0 \subseteq X_0$.

Induction Step: Suppose that the lemma is true for all strings of length up to n. Let $\sigma = wv$ where $|w| = n$ and $v \in V$. We also assume that $\delta_{N+1}((X_0, u_0), wv) \neq \emptyset$ (because the lemma is already true otherwise). For any $(P, u_j) \in Q_{N+1}$ such that $(P, u_j) \in \delta_{N+1}((X_0, u_0), wv)$, we can find an intermediate state $(S, u_i) \in Q_{N+1}$ such that $(S, u_i) \in \delta_{N+1}((X_0, u_0), w)$ and $(P, u_j) \in \delta_{N+1}((S, u_i), v)$. We know from Definition 2 that there exists a continuous trajectory from S to P under u_i that triggers the event v and causes the discrete state to switch from u_i to u_j. By the induction hypothesis, we have that there exists a state $(S', u_i) \in Q_N$ such that $(S', u_i) \in \delta_N((X_0, u_0), w)$ and $S \subseteq S'$. We can see from the refinement procedure that $P \subseteq P'$ for some $P' \in \mathcal{P}_N(u_j)$. Since $S \subseteq S'$, $P \subseteq P'$, and there exists a continuous trajectory from S to P, the same trajectory also goes from S' to P', triggering the same event and the same discrete state transition. Thus, $(P', u_j) \in \delta_N((S', u_i), v)$. Finally, $(S', u_i) \in \delta_N((X_0, u_0), w)$ and $(P', u_j) \in \delta_N((S', u_i), v)$ implies that $(P', u_j) \in \delta_N((X_0, u_0), wv)$. Thus, the lemma is true for the string of length $n + 1$.

■

Proof of Lemma 4. Let $\sigma = wv_\perp$. Suppose $q_\perp \in \delta_{N+1}((X_0, u_0), wv_\perp)$. We can find an intermediate state $(P, u_i) \in Q_{N+1}$ such that $(P, u_i) \in \delta_{N+1}((X_0, u_0), w)$ and $q_\perp \in \delta_{N+1}((P, u_i), v_\perp)$. By Lemma 3, we have that there exists another state $(P', u_i) \in Q_N$ such that $(P', u_i) \in \delta_N((X_0, u_0), w)$ and $P \subseteq P'$. From the definition of the automaton transition function, we have that $P \cap \mathcal{N}_{u_i} \neq \emptyset$. Since $P \subseteq P'$ and $P \cap \mathcal{N}_{u_i} \neq \emptyset$, we have that $P' \cap \mathcal{N}_{u_i} \neq \emptyset$. Consequently, $q_\perp \in \delta_N((P', u_i), v_\perp)$. Finally, $(P', u_i) \in \delta_N((X_0, u_0), w)$ and $q_\perp \in \delta_N((P', u_i), v_\perp)$ imply that $q_\perp \in \delta_N((X_0, u_0), wv_\perp)$.

■

Proof of Proposition 2. Suppose $\sigma \in L(A_{N+1})$, then we consider any $w \in pre(\sigma)$. It follows from Definition 5 that $\delta_{N+1}((X_0, u_0), w) \neq \emptyset$. We consider two types of states in $\delta_{N+1}((X_0, u_0), w)$.

Case 1 If $(P, u_i) \in \delta_{N+1}((X_0, u_0), w)$, then there exists some state $(P', u_i) \in \delta_N((X_0, u_0), w)$ by Lemma 3.

Case 2 if $q_\perp \in \delta_{N+1}((X_0, u_0), w)$, then $q_\perp \in \delta_N((X_0, u_0), w)$ by Lemma 4.

In both cases, $\delta_N((X_0, u_0), w)$ is not empty for any prefix w of σ. Thus, $\sigma \in L(A_N)$.

■

Proof of Lemma 5. We prove this by contradiction. Suppose that for (P_i, u_i) there exists a $(P_j, u_j) \in Q_A$ such that $(P_j, u_j) \in \delta_A((P_i, u_i), v_j)$ but $\mathcal{B}_{u_i}(P_j \uparrow v_j \to P_i) = S \subset P_i$. If we compute the backward mappings and the null event set from all successor states of (P_i, u_i) and apply the `disjoint` routine to this collection of subsets of P_i, we will have a refinement of P_i because S is a strict subset of P_i. Thus, $\mathcal{P}_{N+1}(u_i) \neq \mathcal{P}_N(u_i)$ and we have a contradiction.

For the null state $q_\perp$, if $q_\perp \in \delta_A((P_i, u_i), v_\perp)$ but $N_{u_i} \cap P_i \subset P_i$. We can construct a similar argument which implies that P_i will be broken down in the refinement procedure, leading to a contradiction.

■

Proof of Lemma 6. Again, we prove this lemma by induction on the length of event string σ.

Basis Step: $\sigma = \epsilon$. In this case, the lemma is true because $\delta_A((X_0, u_0), \epsilon) = \{(X_0, u_0)\}$ and $(x_0, u_0) \in \delta_H((x_0, u_0), \epsilon), \forall x_0 \in X_0$.

Induction Step: Assume the lemma is true for all event strings of length up to n. Let $\sigma = wv, |w| = n$ and $v \in V$. Suppose $(P_j, u_j) \in \delta_A((X_0, u_0), wv)$ for some $(P_j, u_j) \in Q_A$. We can find an intermediate state $(P_i, u_i) \in Q_A$ such that $(P_i, u_i) \in \delta_A((X_0, u_0), w)$ and $(P_j, u_j) \in \delta_A((P_i, u_i), v)$. By the induction hypothesis, we have that there exists $(x_i, u_i) \in \delta_H((x_0, u_0), w)$ for some $x_0 \in X_0$ such that $x_i \in P_i$. From Lemma 5, we have that $\mathcal{B}_{u_i}(P_j \uparrow v \to P_i) = P_i$. And since $x_i \in P_i$, it follows that there exists a trajectory under u_i from x_i to some $x_j \in P_j$ that triggers v and causes the discrete state to switch from u_i to u_j. Consequently, $(x_j, u_j) \in \delta_H((x_i, u_i), v)$. Since $(x_i, u_i) \in \delta_H((x_0, u_0), w)$ for some $x_0 \in X_0$ and $(x_j, u_j) \in \delta_H((x_i, u_i), v)$, we conclude that $(x_j, u_j) \in \delta_H((x_0, u_0), wv)$ for the same x_0. Thus, the lemma is also true for event strings of length $n + 1$.

■

Proof of Lemma 7. Let $\sigma = wv_\perp$. Suppose $q_\perp \in \delta_A((X_0, u_0), wv_\perp)$, then we can find an intermediate state $(P_i, u_i) \in Q_A$ such that $(P_i, u_i) \in \delta_A((X_0, u_0), w)$ and $q_\perp \in \delta_A((P_i, u_i), v_\perp)$. By Lemma 6, there exists $(x_i, u_i) \in \delta_H((x_0, u_0), w)$ for some $x_0 \in X_0$ such that $x_i \in P_i$. From Lemma 5, $\mathcal{N}_{u_i} \cap P_i = P_i$. Thus, we also have that $x_i \in \mathcal{N}_{u_i}$. Consequently, $q_\perp \in \delta_H((x_i, u_i), v_\perp)$. Finally, $(x_i, u_i) \in \delta_H((x_0, u_0), w)$ for some $x_0 \in X_0$ and $q_\perp \in \delta_H((x_i, u_i), v_\perp)$ imply that $q_\perp \in \delta_H((x_0, u_0), wv_\perp)$ for the same x_0.

■

Proof of Proposition 3. Since we have already proven that $L(H) \subseteq L(A)$, which is a result that is independent of the assumption stated for this proposition, it only remains to prove that $L(A) \subseteq L(H)$. Suppose $\sigma \in L(A)$. Consider any $w \in pre(\sigma)$. By definition, $\delta_A((X_0, u_0), w) \neq \emptyset$. There are two types of states that could be in $\delta_A((X_0, u_0), w)$.

Case 1 If $(P_i, u_i) \in \delta_A((X_0, u_0), w)$, then by Lemma 6, there exists $(x_i, u_i) \in \delta_H((x_0, u_0), w)$ for some $x_0 \in X_0$ such that $x_i \in P_i$.

Case 2 If $q_\perp \in \delta_A((X_0, u_0), w)$, then it follows from Lemma 7 that $q_\perp \in \delta_H((x_0, u_0), w)$ for some $x_0 \in X_0$.

In both cases, there exists $x_0 \in X_0$ such that $\delta_H((x_0, u_0), w) \neq \emptyset$. Since this statement is true for all $w \in pre(\sigma)$, we have that $\sigma \in L(H)$.

■

References

[1] R. Alur, T.A. Henzinger, and P.-H. Ho. Automatic symbolic verification of embedded systems. *IEEE Trans. on Software Engineering*, 22(3):181–201, Mar 1996.

[2] A. Chutinan and B.H. Krogh. Computing approximating automata for a class of linear hybrid systems. In *Hybrid Systems V: 5th International Hybrid Systems Workshop*, pages 261–269, University of Notre Dame, IN 46556, USA, September 11-13 1997. Extended abstract.

[3] A. Chutinan and B.H. Krogh. Computing polyhedral apporximations to dynamic flow pipes. In *Proc. 37th IEEE Conf. On Decision and Control*, Dec 1998.

[4] E. Clarke, O. Grumberg, and D. Long. Verification tools for finite-state concurrent systems. In *Proceedings of A Decade of Concurrency: Reflections and Perspectives*, pages 124–75, REX School/Symposium, Noordwijkerhout, The Netherlands, 1-4 June 1993. Springer-Verlag, Berlin, Germany, 1994.

[5] J.E.R. Cury, B.H. Krogh, and T. Niinomi. Synthesis of supervisory controllers for hybrid systems based on approximating automata. *IEEE Trans. on Automatic Control*, 43(4):564–569, Apr 1998. a preliminary version of this paper also appeared in *Proc. 34th IEEE CDC*, pp. 1461-1466.

[6] T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. Algorithmic analysis of nonlinear hybrid systems. *IEEE Trans. on Automatic Control*, 43(4):540–554, April 1998.

[7] R.P. Kurshan. *Computer-Aided Verification of Coordinating Processes.* Princeton University Press, 1994.

[8] G. Lafferriere, G.J. Papps, and S. Sastry. Hybrid systems with finite bisimulations. Technical Report UCB/ERL M98/15, University of California at Berkeley, April 1998.

[9] T. Niinomi and B.H. Krogh. Modeling and analysis of switched-mode hybrid systems driven by threshold events. In H. Khalil, J. Chow, and P.A. Ioannou, editors, *Proc. of Workshop on Advances in Control Systems and its Applications*, pages 155–172. Springer-Verlag, 1995.

[10] T. Niinomi, B.H. Krogh, and J.E.R. Cury. Refinements of approximating automata for synthesis of supervisory controllers for hybrid systems. In R. Alur, T.A. Henzinger, and E.D. Sontag, editors, *Hybrid Systems III*, pages 475–484. Springer-Verlag, 1996.

[11] A. Puri, P. Varaiya, and V. Borkar. *Hybrid Systes III: Verification and Control*, chapter ϵ-Approximation of Differential Inclusions, pages 362–372. Lecture Notes in Computer Science. Springer, 1996.

[12] P. J. Ramadge and W. M. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(1):81–98, 1989.

[13] R.S. Sreenivas and B.H. Krogh. On condition/event systems with discrete state realizations. *Discrete Event Dynamic Systems: Theory and Applications*, 1:209–236, 1991.