HIOA$^+$: Specification Language and Proof Tools for Hybrid Systems*

Sayan Mitra
MIT Laboratory for Computer Science,
200 Technology Square, Cambridge, MA 02139
mitras@theory.lcs.mit.edu

Abstract

The Hybrid I/O Automaton (HIOA) framework [21] has been used for safety verification of several hybrid systems from different application domains [17, 20, 25]. In most of these earlier studies the trajectories of HIOA were described in an ad hoc manner, combining integral and algebraic equations with IF-THEN-ELSE conditions. This paper introduces $HIOA^+$, which is an HIOA whose trajectories are specified by a set of *activities*. An *activity* consists of a set of trajectories called the *state model*, an invariant, and a stopping condition. The paper describes the semantics for composing $HIOA^+s$. It proposes extensions to the IOA language [11] for specifying $HIOA^+$s.

This paper also shows how $HIOA^+$ specifications can be translated to the language of the theorem prover PVS [26]. The specification language and the translation scheme are illustrated with the Generalized Railroad Crossing example. Our experience with PVS has been promising, indicating that $HIOA^+$ is suitable model for building theorem proving tools for hybrid systems.

1 Introduction

The literature contains several approaches for modeling and analyzing hybrid systems [10, 2]. Control theorists have approached hybrid systems as a special class of dynamical systems with both real and boolean variables. Typically these approaches focus on the continuous dynamics of the system and address problems like stability, controllability, and observability of the system. Com-

*Funding for this research has been provided by AFRL contract F33615-01-C-1850 and DARPA/AFOSR MURI Award F49620-02-1-0325

puter scientists, on the other hand, have been more concerned with the verification problem, and have extended state transition models to capture both discrete and continuous evolution of the system [1, 14, 21]. Generally, algorithmic verification is desirable because it is automatic. In case of hybrid systems however, it is applicable to a restricted subclass of linear hybrid automata only [16]. Deductive verification, or theorem proving techniques, on the other hand, require human interaction, but can be used for any hybrid system.

The Hybrid Input/Output Automaton (HIOA) framework [21] provides a mathematical model for compositional specification and deductive verification of hybrid systems. Discrete and continuous evolution of the variables of a hybrid system are captured in a HIOA model as discrete state transitions and trajectories respectively. The HIOA framework has been used for safety verification of several hybrid systems [29, 20, 17, 9]. In these earlier applications, the trajectories of HIOAs were described by ad hoc combinations of integral and algebraic equations, and IF-THEN-ELSE conditions. As the interaction between the discrete transitions and the trajectories are often very complex, it is desirable to have precise semantics for the trajectory specifications. This is also necessary for building proof tools for HIOA.

In this paper, we introduce a new automaton model called HIOA$^+$, which is the same as HIOA, except that the set of trajectories of a HIOA$^+$ is specified by a set of *activities*. Informally, an activity specifies the trajectories of the hybrid system in a particular mode; the complete set of trajectories is specified by a set of activities. This modular description of trajectories makes HIOA$^+$ easier to specify and analyze. We define the composition operation for HIOA$^+$s, and show that this definition is consistent with the composition of HIOAs. We propose a few extensions of the IOA language [11] for specifying HIOA$^+$.

Finally, we show how HIOA$^+$ specifications can be translated to the language of SRI's Prototype Verification System (PVS) [26]. We have created a template theory in PVS which can be instantiated with particular variables, actions, and activities to generate the PVS specification of the corresponding HIOA$^+$. So far, we have not implemented the tool for automatic translation. The usage of the specification language and the translation scheme are illustrated with the Generalized Railway-gate Crossing (GRC) benchmark problem [12, 1, 13, 5].

The organization of the rest of the paper is as follows: The next section cites and differentiates related work. Section 3 briefly reviews the HIOA model. Section 4 presents the new HIOA$^+$ model.

Section 5 proposes linguistic extensions of the IOA Language for HIOA$^+$. Section 6 describes the translation of HIOA$^+$ to PVS theories. The concluding remarks are made in Section 7.

2 Related Work

The majority of the specification languages developed for hybrid systems are intended for simulation, for example Shift [7], or for supporting model checking tools like ddt [4], CheckMate [6], or HyTech [15]. The language based on the HIOA$^+$ presented in this paper is most closely related to the language based on the phase transition model of Manna and Sipma [23]. Both languages use a set of *activities* to specify the evolution of the system in the different modes, and discrete transitions to describe the jumps between the modes; however the two languages differ in the following ways:

1. The set of activities is finite in a phase transition system, but it may be infinite in HIOA$^+$.

2. The restriction on the progress of time in a phase transition system is specified by a single assertion over the variables. Our language restricts progress of time in the individual activities by *stopping conditions*.

3. In the phase transition model, input, output, and state variables are not distinguished. An activity specifies the evolution of all the continuous variables by a set of differential equations. In our specification language, the evolution of a continuous variable can be specified by differential or algebraic equations. In accordance with the standard state variable representation [18] from control theory, generally, the evolution of the state variables are specified by differential equations, and the evolution of the output variables are specified by algebraic equations.

4. The interpretations of the activities in the two languages are different. In the phase transition system, each activity is associated with an infinite set of time step transitions $\tau[\Delta]$, one for each possible interval duration Δ. These transitions are responsible for advancing the global time and the evolution of the continuous variables. In HIOA$^+$, an activity of defines a set of *trajectories* for the automaton. A trajectory is a map from an interval of time to valuations of the variables.

Our work on formalizing HIOA$^+$ in PVS is similar in spirit to that of the Timed Automaton Modeling Environment (TAME) [3] — a specialized theorem prover environment built on top of PVS for Lynch-Vaandrager (LV) timed automata [22]. The difference between our work and TAME arises from the differences in the underlying models. LV-timed automata communicate via shared actions alone, unlike HIOAs, which also communicate via shared variables. The evolution of continuous variables in LV-timed automata is modeled by time passage actions which capture their cumulative changes over an interval, and therefore, the executions are simply sequences of actions. In the HIOA$^+$ model, the evolution of the continuous state variables may depend on the shared variables, and therefore, the executions are modeled as alternating sequences of actions and trajectories.

3 Hybrid I/O Automata

For a complete discussion of the HIOA modeling framework refer to [21]. If f is a function and S is a set then we write $f\lceil S$ for the function g with $dom(g) = dom(f) \cap S$ such that for every $c \in dom(g)$, $g(c) = f(c)$. If also the range of f is a set of functions then we write $f \downarrow S$ for the function g with $dom(g) = dom(f)$ such that $g(c) = f(c)\lceil S$ for every $c \in dom(g)$.

3.1 The HIOA Model

A hybrid I/O automaton captures the hybrid behavior of a system in terms of discrete transitions and continuous evolution of its state variables. Let V be the set of variables of automaton $\mathcal{A}$. Each $v \in V$ is associated with a *(static) type* defining the set of values v can assume. A *valuation* $\mathbf{v}$ for V is a function that associates each variable $v \in V$ to a value in $type(v)$. The set of valuations for V is written as $val(V)$. A trajectory τ of V is defined as a mapping $\tau : J \rightarrow val(V)$ where J is a left closed interval of time. If J is right closed then τ is said to be *closed* and its *limit time* is the supremum of the domain of τ, also written as $ltime(\tau)$. Each variable $v \in V$ is also associated with a *dynamic type* (or *dtype*) which is the set of trajectories that v may follow.

A hybrid I/O automaton $\mathcal{A}$ is a tuple $(V, A, Q, \Theta, \mathcal{D}, \mathcal{T})$ where : (1) V is a set of variables, partitioned into *internal* X, *input* U, and *output variables* Y. The internal variables are also called state variables. $Z \triangleq X \cup Y$ is the set of *locally controlled or local variables.* (2) A is a set of actions, partitioned into *internal* H, *input* I, and *output actions* O. (3) $Q \subseteq val(X)$ is a set of states, (4)

$\Theta \subseteq Q$ is a non-empty set of *start states*, (5) $\mathcal{D} \subseteq Q \times A \times Q$ is a set of *discrete transitions.* A transition $(\mathbf{x}, a, \mathbf{x}') \in \mathcal{D}$ is written in short as $\mathbf{x} \stackrel{a}{\rightarrow}_{\mathcal{A}} \mathbf{x}'$. (6) $\mathcal{T}$ is a set of *trajectories* for V, such that for every trajectory τ in $\mathcal{T}$, and for every $t \in \tau.dom$, $(\tau \downarrow X)(t) \in Q$. It is required that $\mathcal{T}$ is closed under prefix, suffix, and concatenation.

In addition a HIOA has to satisfy: **E1** *Input action enabling*, which prevents it from blocking any input action, and **E2** *Input trajectory enabling*, which ensures that it is able to accept any trajectory of the input variables either by allowing time to progress for the entire length of the trajectory or by reacting with some internal action before that. An automaton which does not necessarily satisfy **E1** and **E2** is called a *pre-HIOA*. We define the tuple $\mathcal{H} = (V, A, Q, \Theta, \mathcal{D})$ as a **pseudo-HIOA**, that is, a (pre) HIOA without a set of trajectories.

3.2 Executions of HIOAs and pre-HIOA

The first state $\tau(0)\lceil X$ of a trajectory is denoted by $\tau.fstate$. If $\tau.dom$ is finite then $\tau.lstate$ is defined as $\tau(\tau.ltime)\lceil X$. An *execution* of $\mathcal{A}$ is a finite or infinite sequence of actions and trajectories $\alpha = \tau_0, a_1, \tau_1, a_2 \ldots$, where (1) each $\tau_i \in \mathcal{T}$, (2) $\tau_0.fstate \in \Theta$ and (3) if τ_i is not the last trajectory in α then τ_i is finite and $\tau_i.lstate \stackrel{a_{i+1}}{\rightarrow} \tau_{i+1}.fstate$. An execution is *closed* if the sequence is finite and the domain of the final trajectory is a finite closed interval. The length of an execution is the number of elements (actions and trajectories) in the sequence.

3.3 Composition of HIOAs, pre-HIOAs, and pseudo-HIOAs

The parallel composition of (pre) HIOAs allows an automaton model of a complex system to be constructed from component automaton models. Since the internal actions (and variables) of one component automaton are not observable by other automata, and it is required that each output action (and variable) is controlled by exactly one automaton, therefore two (pre) HIOAs can be composed only if they meet certain compatibility conditions.

Definition 1 *A pair of (pre or pseudo) HIOAs $\mathcal{A}_1$ and $\mathcal{A}_2$ are* **compatible** *if, for $i, j \in \{1, 2\}$, $i \neq j$, $X_i \cap V_j = Y_i \cap Y_j = H_i \cap A_j = O_i \cap O_j = \emptyset$.*

Definition 2 (From [21]) *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be compatible (pre) HIOAs. Their* **composition** $\mathcal{A}_1||\mathcal{A}_2$ *is defined to be the tuple $\mathcal{A} = (V, A, Q, \Theta, D, \mathcal{T})$, where*

$$V = U \cup Y \cup X, \quad U = (U_1 \cup U_2) - (Y_1 \cup Y_2), \quad Y = Y_1 \cup Y_2, \quad X = X_1 \cup X_2, \tag{1}$$

$$A = I \cup O \cup H, \quad I = (I_1 \cup I_2) - (O_1 \cup O_2), \quad O = O_1 \cup O_2, \quad H = H_1 \cup H_2, \tag{2}$$

$$Q = \{\mathbf{x} \in val(X) \mid \mathbf{x}\lceil X_1 \in Q_1 \wedge \mathbf{x}\lceil X_2 \in Q_2\}, \tag{3}$$

$$\Theta = \{\mathbf{x} \in Q \mid \mathbf{x}\lceil X_1 \in \Theta_1 \wedge \mathbf{x}\lceil X_2 \in \Theta_2\}, \tag{4}$$

$$\forall \mathbf{x}, \mathbf{x}' \in Q, \forall a \in A,\ \mathbf{x} \xrightarrow{a}_{\mathcal{A}} \mathbf{x}' \iff (a \in A_i \wedge \mathbf{x}\lceil X_i \xrightarrow{a}_i \mathbf{x}'\lceil X_i) \vee (a \notin A_i \wedge \mathbf{x}\lceil X_i = \mathbf{x}'\lceil X_i), i \in \{1, 2\} \tag{5}$$

$$\mathcal{T} \subseteq trajs(V), \tau \in \mathcal{T} \iff \tau \downarrow V_1 \in \mathcal{T}_1 \wedge \tau \downarrow V_2 \in \mathcal{T}_2. \tag{6}$$

The composition of two HIOAs is guaranteed to be a pre-HIOA, but in general it is not necessarily a HIOA, because it may not satisfy **E2**. Composition of pseudo-HIOAs is defined in the same way as given by Definition 2, except that the composition rule (6) for the set of trajectories is not applicable.

4 HIOA with Added Structure

The HIOA model provides the semantics for composition and implementation of hybrid systems, and it is fairly general in terms of modeling the continuous evolution. Apart from the prefix, suffix, and concatenation closure requirements mentioned above, there are no other restrictions on the trajectories. Absence of structure in the trajectories provides no guidance towards specifying them systematically. Most hybrid systems, however, do manifest a *mode-switching* structure [1]. In order to specify the trajectories of HIOA in a modular fashion, we introduce the notion of *activities*.

4.1 Activities

Definition 3 *An* **activity** *c of pseudo-HIOA $\mathcal{H}$ is a triple (E, P, T), where: (1) $E \subseteq trajs(V)$ is called a state model for the variables of $\mathcal{H}$, and is closed under prefix, suffix, and countable concatenation, (2) $P \subseteq Q$ is an invariant, and (1) $T \subseteq Q$ is a stopping condition.*

Definition 4 *An activity $c = (E, P, T)$* **specifies** *a set $[c]$ of trajectories, given by, $[c] \triangleq E \cap \{\tau \mid \forall t \in dom(\tau), \tau(t) \in P \ \wedge (\tau(t) \in T \Rightarrow t = ltime(\tau))\}$. The set of trajectories specified by a set of activities $\mathcal{C}$ is given by $[\mathcal{C}] \triangleq \cup_{c\ \in\ \mathcal{C}}[c]$.*

Definition 5 *A set $\mathcal{C}$ of activities for pseudo-HIOA $\mathcal{H}$ is* **consistent** *if every $c \in \mathcal{C}$ is an activity of $\mathcal{H}$, and for every $c, c' \in \mathcal{C}$, $c \neq c'$ implies that P_c and $P_{c'}$ are disjoint.*

Lemma 1 *Let $\mathcal{C}$ be a consistent set of activities for pseudo-HIOA $\mathcal{H}$. Then $[\mathcal{C}]$ is closed under prefix, suffix, and countable concatenation.*

Proof: Let $c = (E_c, P_c, T_c)$ be a member of the set $\mathcal{C}$. Consider a trajectory $\tau \in [c]$, and let τ' be a prefix of τ. Since $\tau \in E_c$ and E_c is closed under prefix, therefore $\tau' \in E_c$. Since τ' is a prefix of τ and for all $t \in dom(\tau)$, $\tau(t) \in P_c$, therefore, for all $t \in dom(\tau')$, $\tau'(t) \in P$. Likewise, it follows that, for all $t \in dom(\tau')$, except possibly $t = ltime(\tau')$, $\tau(t') \notin T_c$. From the above arguments it follows that $\tau' \in [c]$, that is, $[c]$ is closed under prefix. By a similar set of arguments it can be shown that $[c]$ is also closed under suffix and countable concatenation. It follows that the union $[\mathcal{C}] = \cup_{c \in \mathcal{C}}[c]$ is also closed under prefix and suffix. To show closure under concatenation, let $\tau_0, \tau_1, \tau_2, \ldots$ be a sequence of trajectories in $[\mathcal{C}]$, such that, for each non-final index i, τ_i is closed and $\tau_i.lstate = \tau_{i+1}.fstate$. We have to show that $\tau = \tau_0 \frown \tau_1 \frown \tau_2 \ldots \in [\mathcal{C}]$. Let $\tau_i \in [c]$, $\tau_{i+1} \in [c']$, where $c, c' \in \mathcal{C}$. So $\tau_i.lstate \in P_c$ and $\tau_{i+1}.fstate \in P_{c'}$. Let us assume for the sake of contradiction that $c \neq c'$. Since $\mathcal{C}$ is consistent set of activities, $P_c \cap P_{c'} = \emptyset$. But we have $\tau_i.lstate = \tau_{i+1}.fstate \in P_c \cap P_{c'}$, contradicting our assumption. Therefore, $c = c'$, and every τ_i in the sequence is in $[c]$. As $[c]$ is closed under countable concatenation, $\tau \in [\mathcal{C}]$. $\Box$

4.2 HIOA$^+$

Definition 6 *A* **pre-HIOA$^+$** *is defined to be a tuple $\mathcal{A} = (\mathcal{H}, \mathcal{C})$, where $\mathcal{H}$ is a pseudo-HIOA, and $\mathcal{C}$ is a consistent set of activities for $\mathcal{H}$. In addition, if the automaton $(\mathcal{H}, [\mathcal{C}])$ satisfies* **E1** *and* **E2**, *then $\mathcal{A}$ is called a* **HIOA$^+$**.

Lemma 2 *If $(\mathcal{H}, \mathcal{C})$ is a (pre) HIOA$^+$ then $(\mathcal{H}, [\mathcal{C}])$ is a (pre) HIOA.*

Proof: Since $(\mathcal{H}, \mathcal{C})$ is a (pre) HIOA$^+$, it follows from Lemma 1 that $[\mathcal{C}]$ is closed under prefix, suffix, and concatenation, and therefore $(\mathcal{H}, [\mathcal{C}])$ is a HIOA. $\Box$

In the rest of the paper we shall mostly refer to HIOA and HIOA$^+$, but the corresponding statements for pre-HIOA and pre-HIOA$^+$ also hold unless stated otherwise.

4.3 Composing HIOA$^+$s

The definition of the composition of HIOA^+s is based on the corresponding definitions for pseudo-HIOA and the composition of activities.

Definition 7 *HIOA^+s $\mathcal{A}_1 = (\mathcal{H}_1, \mathcal{C}_1)$ and $\mathcal{A}_2 = (\mathcal{H}_2, \mathcal{C}_2)$, are* **compatible** *iff the pseudo-HIOAs $\mathcal{H}_1$ and $\mathcal{H}_2$ are compatible.*

For defining the composition operation for HIOA$^+$ we have to define the composition of activities first. For the following discussion, let $\mathcal{A}_1 = (V_1, A_1, Q_1, \Theta_1, \mathcal{D}_1, C_1)$ and $\mathcal{A}_2 = (V_2, A_2, Q_2, \Theta_2, \mathcal{D}_2, \mathcal{C}_2)$ be compatible HIOA$^+$s, and let $\mathcal{A} = (V, A, Q, \Theta, \mathcal{D}, \mathcal{C})$ be their composition. Also assume that an activity $c_i = (P_i, T_i, E_i)$ belongs the set $\mathcal{C}_i$, for $i \in \{1, 2\}$.

Definition 8 *Let $c_1 \in \mathcal{C}_1$, and $c_2 \in \mathcal{C}_2$. The composed activity $c_1 \odot c_2$ is defined as a new activity $c = (P, T, E)$ where,*

$$E = \{\tau \mid \tau \in trajs(V) \ \wedge \ (\tau \downarrow V_1) \in E_1 \ \wedge \ (\tau \downarrow V_2) \in E_2\}, \tag{7}$$

$$P = \{\mathbf{x} \in Q \mid \mathbf{x}\lceil X_1 \in P_1 \ \wedge \ \mathbf{x}\lceil X_2 \in P_2\}, \tag{8}$$

$$T = \{\mathbf{x} \in Q \mid \mathbf{x}\lceil X_1 \in T_1 \ \vee \ \mathbf{x}\lceil X_2 \in T_2\}. \tag{9}$$

Equations (7) and (8) imply that a trajectory τ specified by a composed activity satisfies the state models and the invariants of *both* the constituent activities. Equation (9) implies that the trajectory τ stops when *either* of the component activities stop.

It is worth mentioning that Equation (9) is useful for deriving restrictions on the progress of time for a composed automata from the stopping conditions of its components. For example, consider a set S of compatible HIOA^+s sharing an external variable *now* which is the output of one of the automata in S, say $\mathcal{A}$. Automaton $\mathcal{A}$ stops time from progressing when some condition $M(now)$ is met. From Equation (9) it follows that any trajectory of any HIOA$^+$ in S stops whenever $M(now)$ is true. In case of discrete time control systems [25], the automaton $\mathcal{A}$ is typically a sampling component which forces some action to occur every Δ interval of time, and equation (9) implies that the maximum length of any trajectory of the composed automaton is Δ.

Definition 9 *The* **composition** *of the set of activities $\mathcal{C}_1$ with the set of activities $\mathcal{C}_2$ is a new set of activities $\mathcal{C}_1 \bigodot \mathcal{C}_2 = \mathcal{C} \triangleq \cup_{c_i \in \ \mathcal{C}_1} \cup_{c_j \in \ \mathcal{C}_2} c_i \odot c_j$.*

Next, we define the composition operation for HIOA$^+$s. Like the composition of HIOAs [21], the result of composing two compatible HIOA$^+$s is a pre-HIOA$^+$, and it is not necessarily a HIOA$^+$.

Definition 10 *If $\mathcal{A}_1$ and $\mathcal{A}_2$ are compatible HIOA$^+$s, then their* **composition** *is defined to be the tuple $\mathcal{A} = (V, A, Q, \Theta, \mathcal{D}, C)$, where V, A, Q, Θ, and $\mathcal{D}$ are obtained from Definition 2 and $\mathcal{C} = \mathcal{C}_1 \bigodot \mathcal{C}_2$.*

The following lemma states that the result of composing two activities of a pair of compatible HIOA$^+$s, is an activity of the composed pre-HIOA$^+$.

Lemma 3 *If c_1 and c_2 are activities of compatible HIOA$^+$s $\mathcal{A}_1$ and $\mathcal{A}_2$, then their composition $c = c_1 \odot c_2$ is an activity of the composed HIOA$^+$ $\mathcal{A} = \mathcal{A}_1 || \mathcal{A}_2$.*

Proof: Let $c = (E, P, T)$. From Definition 8 we know that $E \subseteq trajs(V)$, $P \subseteq Q$, and $T \subseteq Q$. Since the sets E_1 and E_2 are closed under prefix and suffix, it follows that E is also closed under these operations. To show that E is closed under countable concatenation, consider a sequence of trajectories $\tau_0, \tau_1, \tau_2, \ldots$ in E, such that, for each non-final index i, τ_i is closed and $\tau_i.lstate = \tau_{i+1}.fstate$. Therefore, $\tau_i.lstate \lceil X_j = \tau_{i+1}.fstate \lceil X_j$ for each non-final index i and $j = 1, 2$. Since E_1 and E_2 are closed under countable concatenation, it follows that $(\tau_0 \downarrow X_j) ^\frown (\tau_1 \downarrow X_j) ^\frown (\tau_2 \downarrow X_j) \ldots \in E_j$, for $j = 1, 2$. Since X_1 and X_2 are disjoint therefore, $\tau_0 ^\frown \tau_1 ^\frown \tau_2 \ldots \in E$. □

The next lemma states that the set of trajectories specified by the composition of two activities of two compatible HIOA$^+$s, is the same as the set of trajectories specified by the individual activities for the corresponding sets of variables.

Lemma 4 *Let $c_1 \in \mathcal{C}_1$ and $c_2 \in C_2$. Then for any trajectory τ, $\tau \in [c_1 \odot c_2]$ if and only if $(\tau \downarrow V_1) \in [c_1]$ and $(\tau \downarrow V_2) \in [c_2]$.*

Proof: To show the forward implication, let $\tau \in [c_1 \odot c_2]$. First of all, from the definition of composition it follows that $(\tau \downarrow V_1) \in E_1$ and $(\tau \downarrow V_2) \in E_2$. Secondly, for every $t \in dom(\tau)$, $(\tau \downarrow X)(t) \in P$, implies that $(\tau \downarrow X_1)(t) \in P_1$ and $(\tau \downarrow X_2)(t) \in P_2$. Finally, if there exists a $\lambda \in dom(\tau)$ such that $(\tau \downarrow X)(\lambda) \in T$, then $ltime(\tau) = \lambda$. From the definition of composition of trajectories it follows that for every $t \in dom(\tau)$, if $t \neq \lambda$ then, $(\tau \downarrow X_1)(t) \notin T_1$ and $(\tau \downarrow X_2)(t) \notin T_2$. Therefore, $(\tau \downarrow V_1) \in [c_1]$ and $(\tau \downarrow V_2) \in [c_2]$.

For the reverse direction, let $c_1 \odot c_2 = (E, P, T)$, and consider a trajectory τ such that $(\tau \downarrow V_1) \in [c_1]$ and $(\tau \downarrow V_2) \in [c_2]$. Since $(\tau \downarrow V_1) \in E_1$, and $(\tau \downarrow V_2) \in E_2$, and $V = V_1 \cup V_2$, it follows that $\tau \in E$. Next, since for all $t \in dom(\tau)$, $(\tau \downarrow X_1) \in P_1$ and $(\tau \downarrow X_2) \in P_2$, therefore, for all $t \in dom(\tau)$, $(\tau \downarrow X)(t) \in P$. Finally, since for every $t \in dom(\tau)$ except possibly $t = ltime(\tau)$, $(\tau \downarrow X_1)(t) \notin T_1$ and $(\tau \downarrow X_2)(t) \notin T_2$. Therefore it follows that $(\tau \downarrow X)(t) \notin T$. □

The following theorem relates the composition operations for HIOAs, HIOA$^+$s, and activities. It states that, if the sets of trajectories of a pair compatible HIOAs, $(\mathcal{H}_1, \mathcal{T}_1)$ and $(\mathcal{H}_2, \mathcal{T}_2)$ can be specified by two sets of activities $\mathcal{C}_1$ and $\mathcal{C}_2$. Then, the result of their composition is equal to the HIOA obtained by combining the composition of the pseudo-HIOAs $\mathcal{H}_1, \mathcal{H}_2$ with the set of trajectories specified by the composition of the activities $\mathcal{C}_1$ and $\mathcal{C}_2$.

Theorem 1 *Let $\mathcal{A}_1 = (\mathcal{H}_1, \mathcal{T}_1)$ and $\mathcal{A}_2 = (\mathcal{H}_2, \mathcal{T}_2)$ be compatible HIOAs, such that $\mathcal{T}_1 = [\mathcal{C}_1]$ and $\mathcal{T}_2 = [\mathcal{C}_2]$. Then their parallel composition $\mathcal{A}_1 || \mathcal{A}_2$ is equal to $(\mathcal{H}_1 || \mathcal{H}_2, [\mathcal{C}_1 \bigodot \mathcal{C}_2])$.*

Proof: Let $\mathcal{T}$ be the set of trajectories of the composed pre-HIOA $(\mathcal{H}_1, T_1) || (\mathcal{H}_2, T_2)$. Since the composition of HIOAs and pseudo-HIOAs are identical except for the set of trajectories, it will be sufficient to show that for any trajectory τ, $\tau \in \mathcal{T}$ if and only if $\tau \in [\mathcal{C}_1 \bigodot \mathcal{C}_2]$.

For the forward direction, consider any trajectory $\tau \in \mathcal{T}$. From Equation (6) of Definition 2, it is known that $(\tau \downarrow V_1) \in [\mathcal{C}_1]$ and $(\tau \downarrow V_2) \in [\mathcal{C}_2]$. Therefore, there exists $c_k \in \mathcal{C}_1$, $c_l \in \mathcal{C}_2$, such that, $(\tau \downarrow V_1) \in [c_k]$ and $(\tau \downarrow V_2) \in [c_l]$. From Lemma 4, it follows that $\tau \in [c_k || c_l] \subseteq [\mathcal{C}_1 \bigodot \mathcal{C}_2]$.

For the reverse direction, consider a trajectory $\tau \in [c_k || c_l]$, where $c_k \in \mathcal{C}_1$ and $c_l \in C_2$. From Lemma 4, it follows that $(\tau \downarrow V_1) \in [c_k]$ and $(\tau \downarrow V_2) \in [c_l]$. Therefore, $(\tau \downarrow V_1) \in [\mathcal{C}_1]$ and $(\tau \downarrow V_2) \in [\mathcal{C}_2]$. From the Definition 2, it follows that $\tau \in \mathcal{T}$. □

5 Language Constructs

In this section we propose extensions to the IOA language [11] for specifying hybrid systems based on the HIOA$^+$ model.

5.1 Variables, Actions and Activities

A variable v is *discrete* if its dynamic type is the pasting closure of the set of constant functions from left closed intervals of time to $type(v)$. A variable is *continuous* if its dynamic type is the pasting closure of the set of continuous functions from left closed intervals of time to R. We assume that each variable is either discrete or continuous. Variables are declared by their names, types, and initial valuations (except input variables). In addition, the keyword **analog** is used to declare a continuous variable, otherwise it is assumed to be discrete.

The action signatures and the discrete transitions of HIOA$^+$ are written in the same way as in the IOA language. An activity $\alpha : (E, P, T)$ is written as: **activity** α **evolve** E **where** P **stop at** T. The preconditions of the discrete transitions, the invariants, and the stopping conditions of the activities are written as predicates on the state variables, with the usual boolean operators. If the stopping condition T is omitted in the specification of an activity, then it is assumed to be empty.

5.2 State Model

The state model is written as a set of *time-invariant*[1] differential and algebraic equations (or inequalities) specifying the evolution of the locally controlled (internal and output) continuous variables of the HIOA$^+$. The discrete variables are assumed to be constant over all trajectories and the trajectories of the input variables are not specified as they are not controlled by the automaton.

5.2.1 Syntax

Figure 1 gives the syntax for writing a general class of differential and algebraic equations for specifying state models. The keywords of the language are written in **boldface**, the terminals in Sans serif, and the non-terminals in `typewriter` fonts. Zero or more `x`'s separated by commas are written as `x,*`. Likewise, `x,+` stands for one or more `x`'s separated by commas. An algebraic equation (or inequality) for variable x is written as an assignment to x. A differential equation (or inclusion) for x is written as an assignment to $d(x)$. An equation is an assignment whose right hand side is a `term`. An inequality or an inclusion is an assignment whose right hand side is a `choice`. A `term` is constructed from variables, numerals, binary operators, and functions. A function is

[1] The time-invariance requirement is necessary for concatenation closure, and it does not limit the expressive power of the language.

either an elementary math function like `exp, ln, log10, sin, cos, tan, sqrt,` and `arctan,` or a locally declared function.

The sate model consists of a single differential or algebraic equation (or inequality) for each locally controlled continuous variable of the automaton. In addition, it may contain named functions declarations.

```
stateModel      ::= funcDecl;* assignment;+
funcDecl        ::= let  newFuncName '(' identifier,+ ')' '==' term
newFuncName     ::= identifier
assignment      ::= lvalue ':=' value
lvalue          ::= variable | 'd('variable')'
variable        ::= identifier {'.'  variable}*
value           ::= term | choice
choice          ::= choose {'[' | '('} term ',' term {']' | ')' }
term            ::= term binOp term | '(' term ')' | functionTerm | variable | number
binOp           ::= '+' | '-' | '\' | '*' | '**'
functionTerm    ::= functionName '(' term,+ ')'
```

Figure 1: Grammar for state model specification

5.2.2 Semantics

A trajectory τ belongs to the state model E if at all times in $dom(\tau)$, all the locally controlled continuous variables *satisfy* the differential and algebraic equations (or inequalities) specifying E, with $\tau(0)$ defining the initial valuations. Table 1 summarizes the rules which determine if a trajectory τ satisfies some given algebraic and differential equations or inequalities. In the table v is a locally controlled continuous variable, e is a real valued deterministic term, and $\tau.e$ denotes the function with domain $dom(\tau)$ that gives the value of e at all times over $dom(\tau)$. Further, for differential equations (inclusions) the terms e and e' should be integrable.

Assignment	Interpretation
$v := e$	$\forall t \in dom(\tau)$, $(\tau \downarrow v)(t) = \tau.e(t)$
$v :=$ **choose**$[e, e']$	$\forall t \in dom(\tau)$, $(\tau \downarrow v)(t)$ is in the interval $[\tau.e(t), \tau.e'(t)]$
$d(v) := e$	$\forall t \in dom(\tau)$, $(\tau \downarrow v)(t) = (\tau \downarrow v)(0) + \int_0^t \tau.e(t')\ dt'$
$d(v) :=$ **choose**$[e, e']$	$\forall t_1, t_2 \in dom(\tau)$, $\int_{t1}^{t2} \tau.e(t')dt' \leq (\tau \downarrow v)(t_2) - (\tau \downarrow v)(t_1) \leq \int_{t_1}^{t_2} \tau.e'(t')dt'$.

Table 1: Interpretation of state model assignments.

5.3 The GRC Problem Specification : An Example

The language constructs described above are illustrated by specifying the Generalized Railway-gate Crossing (GRC) problem. Although the continuous behavior involved in this problem is very simple, the language has been used to model systems with more complicated dynamics, see for example [25]. The Alur-Dill hybrid automaton [1] representation of the train, the gate, and the controller automata are shown in Figure 2. The equalities and the assignments on the arrowheads denote the preconditions and the effects of the jumps. Each circle corresponds to a mode of the hybrid system.

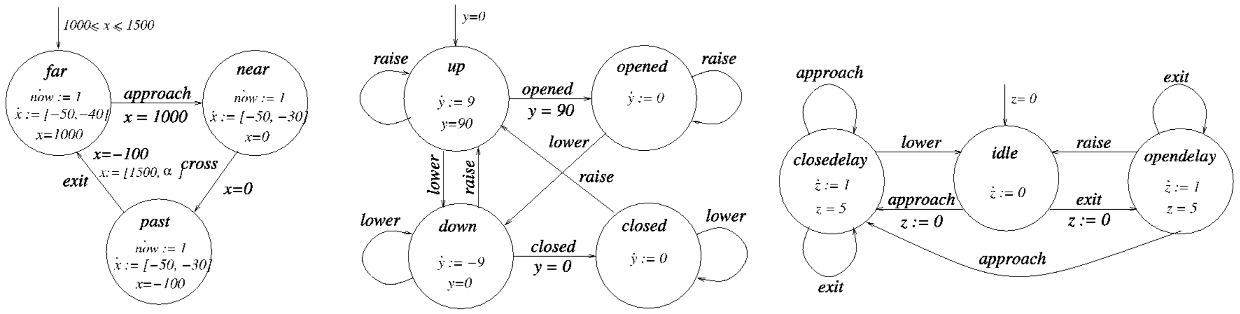

Figure 2: Train, Gate, and Controller automata.

The HIOA$^+$ specifications of the automata are given in Figures 3, 4 and 5. Generally, each circle in Figure 2 corresponds to an activity of HIOA$^+$. If however, the differential equations describing the state model and the invariants of two or more modes are identical then they can be expressed by a single activity. For example, the *opened* and *closed* modes of the Gate automaton are combined into the *stopped* activity of Figure 4.

6 HIOA$^+$ Specifications in PVS

Earlier experiences [17, 19, 25] with HIOA indicate that deductive verification of moderately complex systems involves considerable human effort. This effort could be significantly reduced by using a mechanical theorem prover. An important hurdle in using a theorem prover, however, is the need to first master the specification language and the logic of the theorem proving system. This problem can be addressed by building specialized interfaces which generate description of hybrid I/O automata in the language of the prover from the HIOA$^+$ specifications, and provide high level

```
type Train_dist = { far, near, past }

hybridautomaton Train
actions
    output approach, exit
    internal cross

variables
    internal analog x : Real choose [1000, 1500],
    output analog now : Real := 0,
    internal mode : Train_dist := far

discrete transitions
    output approach
        pre x = 1000  ∧ mode = far
        eff mode := near

    output exit
        pre x = −100  ∧ mode = past
        eff mode := far; x := choose [1500, ∞]

    output cross
        pre x = 0  ∧ mode = near
        eff mode := past

trajectories
    activity approaching
        where mode = far
        evolve d(x) := choose [−50, −40] ; d(now) := 1
        stop at x = 1000

    activity crossing
        when mode = near
        evolve d(x) := choose [−50, −30] ; d(now) := 1
        stop at x = 0

    activity exiting
        when mode = past
        evolve d(x) := choose [−50, −30] ; d(now) := 1
        stop at x = −100
```

Figure 3: HIOA$^+$ specification of `Train` automaton

```
type Gate_pos = { up, opened, down, closed }

hybridautomaton Gate
actions
    input raise, lower
    internal opened, closed

variables
    internal analog y : Real := 0,
    internal mode : Gate_pos := up

discrete transitions
    input raise
        eff if mode = closed  ∨ mode = down then mode := up fi

    input lower
        eff if mode = opened  ∨ mode = up then mode := down fi

    internal opened
        pre y = 90  ∧ mode = up
        eff mode := opened

    output closed
        pre y = 0  ∧ mode = down
        eff mode := closed

trajectories
    activity rising
        when mode = up
        evolve d(y) := 9
        stop at y = 90

    activity lowering
        when mode = down
        evolve d(y) := −9
        stop at y = 0

    activity stopped
        when mode = opened  ∨ mode = closed
        evolve d(y) := 0
```

Figure 4: HIOA$^+$ specification of `Gate` automaton

type `Controller_modes` = { `idle`, `open_delay`, `close_delay` }

hybridautomaton `Controller`

actions
  input *approach, exit*
  output *raise, lower*

variables
  internal analog z : `Real` := 0,
  internal *mode* : `Controller_modes` := `idle`

discrete transitions
  input *approach*
    eff if *mode* ≠ `close_delay` **then**
      $z := 0$; *mode* := `close_delay` **fi**

  input *exit*
    eff if *mode* = `idle` **then**
      $z := 0$; *mode* := `open_delay` **fi**

  output *raise*
    pre *mode* = `open_delay`
    eff *mode* := `idle`

  output *lower*
    pre *mode* = `close_delay`
    eff *mode* := `idle`

trajectories
  activity *idling*
    when *mode* = `idle`
    evolve $d(z) := 0$

  activity *open_wait*
    when *mode* = `open_delay`
    evolve $d(z) := 1$
    stop at $z = 5$

  activity *close_wait*
    when *mode* = `close_delay`
    evolve $d(z) := 1$
    stop at $z = 5$

Figure 5: HIOA$^+$ specification of `Controller` automaton

proof strategies for proving properties of HIOA. In this section we show how HIOA$^+$ specifications can be translated to PVS theories. PVS is a generalized specification and verification environment that provides an expressive specification language and an interactive theorem prover. For details on the PVS specification language and the prover we refer the reader to [27] and [28]. We have created a template PVS theory, which is instantiated with the actions, variables, and activities of a particular HIOA$^+$ to get PVS theories describing the HIOA$^+$, its executions, and the necessary induction principle to prove invariants. The basic translation scheme is illustrated with the GRC example[2].

6.1 Variables, Actions, and Activities

The variables of a HIOA are defined as a *record* of the individual variables with their corresponding types. In PVS record types are declared using the '[#' and '#]' parentheses.

```
TMODES: TYPE = {far, near, past}
tState: TYPE = [# x: real, mode: TMODES #]
```

[2]The PVS dump file can be obtained from `theory.lcs.mit.edu/mitras/research/pvs/GRC.dump`

```
train_action: DATATYPE
  BEGIN
    approach: approach?
    exit(p: {r: real | r ≥ 1500}): exit?
    cross: cross?
  END action

pre(a: train_action, s: tState): bool =
  CASES a
    OF approach: x(s) = 1000 ∧ tmode(s) = far,
    exit(p): x(s) = −100 ∧ tmode(s) = past,
    cross: x(s) = 0 ∧ tmode(s) = near
  ENDCASES

eff(a: train_action, s1, s2: tState): bool =
  CASES aOF
    approach: s2 = (# mode := near, x := x(s1) #),
    exit(p): s2 = (# mode := far, x := p #),
    cross: s2 = (# mode := past, x := x(s1) #),
  ENDCASES
```

Figure 6: PVS specification of `Train` actions.

The action signature is defined as an action datatype, with a separate constructors for each action type. The action signature and the discrete transitions of the `Train` automaton are shown in Figure 6. The syntax for defining an abstract datatype in PVS consists of names for the constructor (e.g., "approach") and the recognizer (e.g., "approach?") functions. The parameters of an action, if any, become parameters of the corresponding constructor.

The preconditions of the actions are defined as a function pre : $[a : \mathit{action}, s : \mathit{grcState}] \rightarrow \mathit{bool}$, that returns true if the action a is enabled at s. The effects of the actions are defined as a function eff : $[s_1, s_2 : \mathit{grcState}, a : \mathit{action}] \rightarrow \mathit{bool}$, that returns true if s_2 is a possible post-state of action a occurring at state s_1.

The names of the activities are also defined as an abstract datatype. For example, the activity names of the `Train` automaton, and the corresponding invariants, stopping conditions, and state models are shown in Figure 7. The nondeterminism in the activities are modeled by an additional parameter(w) in the constructors. The invariants and the stopping conditions of the activities are defined as boolean functions. The statemodel function defines the state model of each activity. Given an activity name a, a state s_1, and an interval of time J, it returns the lstate[3] of the J-trajectory starting from s_1.

[3] If the interval J is infinite, then the *lstate* is not defined, and a special "undefined state" value $\perp$ is returned. This is discussed in the next section.

```
train_activity: DATATYPE
  BEGIN
    approaching(w : ({r : real| − 50 ≤ r ≤ −40})): approaching?
    crossing(w : ({r : real| − 50 ≤ r ≤ −30})): crossing?
    exiting(w : ({r : real| − 50 ≤ r ≤ −30})): exiting?
  END train_activity

opcon(a: train_activity, s: tState): bool =
  CASES a OF
    approaching(w): tmode(s) = far ∧ x(s) ≥ 1000,
    crossing(w): tmode(s) = near ∧ (x(s) ≥ 0 ∧ x(s) ≤ 1000),
    exiting(w): tmode(s) = past ∧ (x(s) ≥ −100 ∧ x(s) ≤ 0)
  ENDCASES

statemodel(a: train_activity, fs: tState,
    t: (finiteTime?)): tState =
  CASES a OF
    approaching(w): (# mode := tmode(fs),
        x := x(fs) − w × dur(t) #),
    crossing(w): (# mode := tmode(fs),
        x := x(fs) − w × dur(t) #),
    exiting(w): (# mode := tmode(fs),
        x := x(fs) − w × dur(t) #),
  ENDCASES

stopcon(a: train_activity, s: tState): bool =
  CASES a OF
    approaching(w): x(s) = 1000,
    crossing(w): x(s) = 0,
    exiting(w): x(s) = −100
  ENDCASES
```

Figure 7: PVS specification of the activities of the `Train` automaton

6.2 Composition

From the specification of the component automata in PVS, it is straightforward to construct the specification of the composed automaton. The variables of the composed `GRC` automaton is defined as a record combining the component variables. Often it is useful to have a special "undefined state" (for example, when accessing the lstate of an unbounded trajectory), therefore we have lifted the `grcState` type with a special value ⊥ in the actual specification.

```
grcState: TYPE = [# now: (finiteTime?), g: gState, t: tState, c: cState #]
```

The action signature of the composed automaton is obtained by defining an abstract datatype which is the union of the datatype of the component actions. The composed `pre` and `eff` functions are also defined in the expected way. The set of discrete steps of the composed automaton is defined as a record type `DS` with a pair of states and an action(of the composed automaton), such that the first state satisfies the precondition of the action, and the second state is a possible post-state of the action.

```
dstep: TYPE = [# fstate: grcState, lstate: grcState #]
dstepTest(a: action, D: dstep): bool = pre(a, D`fstate) ∧ eff(a, D`fstate, D`lstate)
DS: TYPE = {ds: [# st: dstep, act: action #] | dstepTest(ds`act, ds`st)}
```

The activities of the composed automaton are defined as a record of the component activities. The invariants of the composed activities are defined as the conjunction of the component invariants. Likewise, the stopping conditions are defined as the disjunction of stopping condition of the component activities.

```
activity: TYPE = [# t: train_activity, g: gate_activity, c: con_activity #]
opcon(a: activity, s: grcState): bool =
     opcon(a`t, tstate(s)) ∧ opcon(a`g, gstate(s)) ∧ opcon(a`c, cstate(s))
stopcon(a: activity, s: grcState): bool =
     stopcon(a`t, tstate(s)) ∨ stopcon(a`g, gstate(s)) ∨ stopcon(a`c, cstate(s))
statemodel(a: activity, fs: grcState, t: (finiteTime?)): grcState =
    (# now := now(fs) + t, t := statemodel(a`t, tstate(fs), t),
       g := statemodel(a`g, gstate(fs), t), c := statemodel(a`c, cstate(fs), t) #)
```

The trajectory type defines a set of mappings from left closed intervals of time to grcState. The TrajTest(a,T) function captures the conditions under which a trajectory T is a member of the set of trajectories defined by activity a. The set of allowed trajectories of the composed automaton is defined as a record type TR, consisting of a trajectory and an activity such that the trajectory is a valid member of the set of trajectories defined by the activity.

```
trajectory: TYPE = [# left: (finiteTime?), length: Time, map:[Time → grcState] #]
TrajTest(a: activity, T:trajectory):
bool = ∀ (t: (domain(T))): traj(T, t) = statemodel(a, fstate(T),t − T`left) ∧ (opcon(a, traj(T, t)))
  ∧ (stopcon(a, traj(T, t)) ⊃ t = ltime(T))
TR: TYPE = {tr: [# traj: trajectory, actvy: activity #] | TrajTest(tr`actvy, tr`traj)}
```

6.3 Executions and Invariants

An execution of the hybrid automaton is defined as a possibly infinite sequence, of alternating elements of type TR and DS. We define a wrapper type ExeEle over the TR and DS types, and define Execution as a sequence of elements of type ExeEle satisfying the ExecSeqTest condition. In order to define Execution sequences we have used the PVS formalization of finite and infinite sequences presented in [8]. The ExecSeqTest returns true if an alternating sequence of discrete steps and trajectories is a valid execution of a hybrid I/O automaton. The set of reachable states R_GRC of the composed GRC automaton is defined.

```
ExeEle: DATATYPE
 BEGIN
  dstep(jump: DS): dstep?
  trajec(flow: TR): trajec?
 END ExeEle

Execution: TYPE = {E: nempty_fin_seq[Exeele] | ExecSeqTest(E)}
R_GRC: SetofStates = {s: grcState | ∃ (E: ClosedExec): s = lstate(E)}
```

In order to prove invariants of HIOA in PVS, we need an induction schema for sequences of ExeEle's. This is provided by the following Induct lemma. This lemma is used in turn show that any set I of grcState's is an invariant of GRC automaton, if the start states belong to I, and the discrete steps and trajectories preserve I.

f: VAR ClosedExec, p: VAR pred[ExeEle]
Induct: LEMMA $(p(f\text{`map}(0))) \wedge (\forall\ (i:\ (\text{dom}(f))): p(f\text{`map}(i)) \wedge (\text{dom}(f))(i+1)$
$\supset p(f\text{`map}(i+1))) \supset (\forall\ (i: (\text{dom}(f))): p(f\text{`map}(i)))$

IsInvariant?(I: SETOF[grcState]): bool = startSub(I) $\wedge$ dstepPrsrv(I) $\wedge$ trajPrsrv(I)
Invariance: THEOREM $\forall$ (I: SetofStates): IsInvariant?(I) $\supset$ (R_GRC $\subseteq I$)
SafeSet: SETOF[grcState] = $\{s$: grcState | tmode(s) = past $\supset$ gmode(s) = closed$\}$

Therefore, for proving safety of the GRC system, one has to show that the set of initial states are contained in SafeSet, and all possible discrete transitions and trajectories preserve this set. Most of the invariant proofs involve a large case analyses which are very well managed by the PVS prover. All the trivial and easy proof branches are resolved automatically by higher level proof commands like GRIND, for the more intricate cases the user involvement is necessary. The complete GRC specification and proofs of safety related invariants are available at:
`theory.lcs.mit.edu/mitras/research/pvs/GRC.dump`.

7 Conclusions

In this paper we have introduced a modified version of the HIOA of Lynch, Segala, and Vaandrager [21] called HIOA$^+$. In this new version of the model a set of trajectories of the hybrid system are specified in a modular fashion by a set of activities. We have defined the composition operation for HIOA$^+$s, and shown its equivalence with the composition of HIOAs. We have also proposed extensions of the IOA [11] language for specifying HIOA$^+$. Finally, we have shown how HIOA$^+$ specifications can be translated to PVS theories. This is a first step towards building a theorem proving environment for hybrid systems. Our experience in specifying the GRC problem using the new model, and proving its safety has been promising.

Acknowledgments

We would like to thank Professor Nancy Lynch and Dilsun Kirli Kaynar for many useful discussions and comments concerning this paper.

References

[1] Rajeev Alur, Costas Courcoubetis, Nicolas Halbwachs, Thomas A. Henzinger, P.-H. Ho, Xavier Nicollin, Alfredo Olivero, Joseph Sifakis, and Sergio Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1):3–34, 1995.

[2] P. Antsaklis and X. Koutsoukos. On hybrid control of complex systems: A survey. In *3rd International Conference ADMP'98, Automation of Mixed Processes: Dynamic Hybrid Systems*, pages 1–8, Reims, France, March 1998.

[3] M. Archer, C. Heitmeyer, and S. Sims. TAME: A PVS interface to simplify proofs for automata models. In *Proceedings of UITP '98*, July 1998.

[4] E. Asarin, O. Bournez, T. Dang, and O. Maler. Approximate reachability analysis of piecewise-linear dynamical systems. In B. Krogh and N. Lynch, editors, *Hybrid Systems: Computation and Control*, volume 1790 of *LNCS*, pages 20–31. Hybrid Systems: Computation and Control, 2000.

[5] Nikolaj S. Bjrner, Zohar Manna, Henny B. Sipma, and Toms E. Uribe. Deductive verification of real-time systems using STeP. In *4th International AMAST Workshop on Real-Time Systems*, May 1997.

[6] Alongkrit Chutinan and Bruce H. Krogh. Verification of polyhedral-invariant hybrid automata using polygonal flow pipe approximations. *Lecture Notes in Computer Science*, 1569:76–91, 1999.

[7] A. Deshpande, A. Gollu, and L. Semenzato. The SHIFT programming language for dynamic networks for hybrid automata. In *IEEE Transactions on Automatic Control*, volume 43(4), pages 584–587, April 1998.

[8] Marco Devillers, David Griffioen, and Olaf Muller. Possibly infinite sequences in theorem provers: A comparative study. In Elsa Gunther, editor, *Theorem Proving in Higher Order Logics (TPHOL'97), LNCS 1275, 89-104.* Springer-Verlag, 1997.

[9] Ekaterina Dolginova and Nancy Lynch. Safety verification for automated platoon maneuvers: A case study. In *HART'97 (International Workshop on Hybrid and Real-Time System)*, volume 1201 of *Lecture Notes in Computer Science series*, Grenoble, France, March 1997. Springer Verlag.

[10] M.M. Bayoumi G. Labinaz and K. Rudie. A survey of modeling and control of hybrid systems. In *Annual Reviews in Control*, volume 21, pages 79–92, 1997.

[11] Stephen Garland, Nancy Lynch, and Mandana Vaziri. IOA: A language for specifying, programming and validating distributed systems. Technical report, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA, October 1999.

[12] C. L. Heitmeyer, R. D. Jeffords, and B. G. Labaw. A benchmark for comparing different approaches for specifying and verifying real-time systems. In *10th International Workshop on Real-Time Operating Systems and Software*, May 1993.

[13] Connie Heitmeyer and Nancy Lynch. The generalized railroad crossing: A case study in formal verification of real-time system. In *Proceedings of the 15th IEEE Real-Time Systems Symposium*, San Juan, Puerto Rico, December 1994. IEEE Computer Society Press.

[14] Thomas Henzinger. The theory of hybrid automata. In *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science (LICS '96)*, pages 278–292, New Brunswick, New Jersey, 1996.

[15] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. Hytech: A model checker for hybrid systems. In *Computer Aided Verification (CAV '97)*, volume 1254 of *Lecture Notes in Computer Science*, pages 460–483, 1997.

[16] Thomas A. Henzinger and Rupak Majumdar. Symbolic model checking for rectangular hybrid systems. In *Proceedings of the Sixth International Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2000)*, pages 142–156. Lecture Notes in Computer Science 1785, Springer-Verlag, 2000.

[17] Carolos Livadas, John Lygeros, and Nancy A. Lynch. High-level modeling and analysis of TCAS. In *Proceedings of the 20th IEEE Real-Time Systems Symposium (RTSS'99),Phoenix, Arizona*, pages 115–125, December 1999.

[18] David G. Luenberger. *Introduction to Dynamic Systems: Theory, Models, and Applications*. John Wiley and Sons, Inc., New York, 1979.

[19] Nancy Lynch. Modelling and verification of automated transit systems, using timed automata, invariants and simulations. In T. Henzinger R. Alur and E. Sontag, editors, *Hybrid Systems III: Verification and Control (DIMACS/SYCON Workshop on Verification and Control of Hybrid Systems*, volume 1066 of *Lecture Notes in Computer Science*, pages 449–463, 1995.

[20] Nancy Lynch. A three-level analysis of a simple acceleration maneuver, with uncertainties. In *Proceedings of the Third AMAST Workshop on Real-Time Systems*, pages 1–22, Salt Lake City, Utah, March 1996. World Scientific Publishing Company.

[21] Nancy Lynch, Roberto Segala, and Frits Vaandraager. Hybrid I/O automata. To appear in *Information and Computation*. Also, Technical Report MIT-LCS-TR-827d, MIT Laboratory for Computer Science Technical Report, Cambridge, MA 02139, January 13, 2003. `theory.lcs.mit.edu/tds/papers/Lynch/HIOA-final.ps`.

[22] Nancy Lynch and Frits Vaandrager. Forward and backward simulations - part ii: Timing-based systiems. *Information and Computation*, 128(1):1–25, July 1996.

[23] Zohar Manna and Henny Sipma. Deductive verification of hybrid systems using STeP. In *Hybrid Systems: Computation and Control*, volume 1386 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998.

[24] Sayan Mitra, Yong Wang, Nancy Lynch, and Eric Feron. Application of hybrid I/O automata in safety verification of pitch controller for model helicopter system. Technical Report MIT-LCS-TR-880, MIT Laboratory for Computer Science, Cambridge, MA 02139, January 2003. http://theory.lcs.mit.edu/mitras/research/QuanTR02.ps.

[25] Sayan Mitra, Yong Wang, Nancy Lynch, and Eric Feron. Safety verification of model helicopter controller using hybrid Input/Output automata. In *HSCC'03, Hybrid System: Computation Control*, Prague, the Czech Republic, April 3-5 2003. Also, long version in [24].

[26] S. Owre, S. Rajan, J.M. Rushby, N. Shankar, and M.K. Srivas. PVS: combining specification, proof checking, and model checking. In Rajeev Alur and Thomas A. Henzinger, editors, *Computer-Aided Verification, CAV '96*, number 1102 in Lecture Notes in Computer Science, pages 411–414, New Brunswick, NJ, July/August 1996. Springer-Verlag.

[27] S. Owre, N. Shankar, J. M. Rushby, and D. W. J. Stringer-Calvert. *PVS Language Reference*. Computer Science Laboratory, SRI International, Menlo Park, CA, September 1999.

[28] N. Shankar, S. Owre, J. M. Rushby, and D. W. J. Stringer-Calvert. *PVS Prover Guide*. Computer Science Laboratory, SRI International, Menlo Park, CA, September 1999.

[29] H. B. Weinberg and Nancy Lynch. Correctness of vehicle control systems – a case study. In *17th IEEE Real-Time Systems Symposium*, pages 62–72, Washington, D. C., December 1996.