Hybrid I/O Automata Revisited

Nancy Lynch[1]*, Roberto Segala[2]**, and Frits Vaandrager[3]***

[1] MIT Laboratory for Computer Science, Cambridge, MA 02139, USA, `lynch@theory.lcs.mit.edu`
[2] Dipartimento di Matematica, Università di Bologna, Piazza di Porta San Donato 5, 40127 Bologna, Italy, `segala@cs.unibo.it`
[3] Computing Science Institute, University of Nijmegen, P.O. Box 9010, 6500 GL Nijmegen, The Netherlands, `fvaan@cs.kun.nl`

Abstract. In earlier work, we developed a mathematical *hybrid I/O automaton (HIOA)* modeling framework, capable of describing both discrete and continuous behavior. This framework has been used to analyze examples of automated transportation systems, intelligent vehicle highway systems, air traffic control systems, and consumer electronics applications. Here, we reconsider the basic definitions of the HIOA framework, in particular, the dual use of external variables for discrete and continuous communication. We present a new HIOA model that is simpler than the earlier model, due to a clearer separation between discrete and continuous activity.

1 Introduction

Recent years have seen a rapid growth of interest in *hybrid systems*—systems that contain both discrete and continuous components, typically computers interacting with the physical world. Such systems are used in many application domains, including automated transportation, avionics, automotive control, process control, robotics, and consumer electronics. Motivated by a desire to describe and reason carefully about such applications, we are continuing our efforts to adapt techniques from computer science to the setting of hybrid systems.

In our previous work in this area, we developed a mathematical *hybrid I/O automaton* modeling framework [15, 16]. This framework supports description and analysis of hybrid systems using powerful methods of *parallel composition* and *levels of abstraction*. We also proved sufficient conditions for hybrid I/O automata to be *receptive*, which means that they allow time to advance to infinity independently of the input provided by the environment. We and others have used this framework to analyze examples of automated transportation systems [18, 13, 23, 22, 14, 10], intelligent vehicle highway systems [6, 12], air traffic control systems [11, 9], and consumer electronics systems [4].

* upported by AFOSR F49620-00-1-0097, F49620-97-1-0337; NTT MIT9904-12; NSF ACI-9876931, CCR-9909114, CCR-9804665; PATH 1784-18454LD.
** Supported by MURST project TOSCA.
*** Supported by Esprit Project 26270, Verification of Hybrid Systems (VHS).

In this paper, we present a *new hybrid I/O automaton model* that is considerably simpler than the earlier model, yet supports similar description and analysis methods and similar receptivity theorems. The main simplification is a clearer separation between the notions of discrete and continuous communication. We arrived at this separation as a result of reconsidering the relationship between the computer science notion of shared variable communication and the control theory notion of continuous flow across component boundaries.

Levels of abstraction, compositionality, and receptiveness for hybrid systems have also been addressed by Alur and Henzinger [2, 3] in their work on reactive modules. However, reactive modules communicate only via shared variables, and not via shared actions. In [3], a definition of receptiveness similar to the one in [15, 16] is proposed, and is shown to be preserved by composition. However, in [3], no circular dependencies ("feedback loops") are allowed among the continuous variables of the components, a restriction that greatly simplifies the analysis.

The rest of this paper is organized as follows. Section 2 defines notions that are useful for describing the behavior of hybrid systems: trajectories and hybrid sequences. Section 3 contains the theory for the *hybrid automaton (HA)* model, which has all of the structure of the HIOA model except for the division of external actions and variables into inputs and outputs. Section 4 introduces inputs and outputs, and presents the basic theory for HIOAs. Section 5 presents the new theory of receptiveness, including the main theorem, Theorem 7, stating that receptiveness is preserved by composition under certain compatibility conditions. Section 6 describes sufficient conditions for these compatibility conditions to hold, and in particular, describes Lipschitz automata.

2 Describing Hybrid Behavior

In this section, we give basic definitions that are useful for describing discrete and continuous system behavior, including discrete and continuous state changes, and discrete and continuous flow of information over component boundaries. Throughout this paper, we fix a *time axis* T, which is a compact subgroup of $(\mathsf{R}, +)$, the real numbers with addition.

2.1 Static and Dynamic Types

We assume a universal set V of *variables*. A variable represents either a location within the state of a system component, or a location where information flows from one system component to another. For each variable, we assume both a *(static) type*, which gives the set of values it may assume, and a *dynamic type*, which gives the set of trajectories it may follow. Our motivation for introducing dynamic types is that this allows us to define input enabling for hybrid I/O automata: if v is an input variable of HIOA $\mathcal{A}$ then, roughly speaking, we require that $\mathcal{A}$ accepts each input signal on v, as long as it respects the dynamic type of v. Since we are in a hybrid setting where discrete transitions may change the state at any time, elements of a dynamic type may contain (countably many) "discontinuities". Formally, we assume for each variable v:

- *type*(v), the *(static) type* of v. This is a set of values.
- *dtype*(v), the *dynamic type* of v. This is a set of functions from left-closed intervals of T to *type*(v) that is closed under the following operations:
  1. (Time shift) For each $f \in \mathit{dtype}(v)$ and $t \in \mathsf{T}$, $f + t \in \mathit{dtype}(v)$. Here $f + t$ is the function given by $(f + t)\,(t') = f(t' - t)$.
  2. (Subinterval) For each $f \in \mathit{dtype}(v)$ and each left-closed interval $J \subseteq \mathit{dom}(f)$, $f \lceil J \in \mathit{dtype}(v)$. Here $f \lceil J$ is the function obtained by restricting the domain of f to J.
  3. (Pasting) For each sequence $f_0, f_1, f_2, \ldots$ of functions in $\mathit{dtype}(v)$ such that (a) the domain of each f_i, except possibly for the last one, is right-closed, (b) for each nonfinal index i, $\max(\mathit{dom}(f_i)) = \min(\mathit{dom}(f_{i+1}))$, the function f given by $f(t) \triangleq f_i(t)$, where i is the smallest index with $t \in \mathit{dom}(f_i)$, is in $\mathit{dtype}(v)$.

Example 1. For any variable v, the set C of constant functions from a left-closed interval to *type*(v) is closed under time shift and subintervals. If the dynamic type of v is obtained by closing C under the pasting operation, then v is called a *discrete* variable, as in [19]. If we take $\mathsf{T} = \mathsf{R}$ and *type*$(v) = \mathsf{R}$, then other examples of dynamic types can be obtained by taking the pasting closure of the set of continuous or smooth functions, the set of integrable functions, or the set of measurable locally essentially bounded functions. The set of all functions from left-closed intervals of R to R is also a dynamic type.

In practice, dynamic types are often defined via pasting closure of a class of continuous functions. In these cases the elements of dynamic types are continuous from the left. Elsewhere in the literature on hybrid systems one often encounters functions that are continuous from the right (see, e.g., [8]). To some extent, the choice of how to define function values at discontinuities is arbitrary. An advantage of our choice is a nice correspondence between concatenation and prefix ordering of trajectories (see Lemma 2). In the rest of this paper, when we say that the dynamic type of a variable v equals S, we actually mean that the dynamic type of v is obtained by applying the above closure operations to S.

2.2 Trajectories

In this subsection, we define the notion of a *trajectory*, define operations on trajectories, and prove simple properties of trajectories and their operations. A trajectory is used to model the evolution of a collection of variables over an interval of time.

Basic Definitions Let V be a set of variables, that is, a subset of V. A *valuation* $\mathbf{v}$ for V is a function that associates to each variable $v \in V$ a value in *type*(v). We write *val*(V) for the set of valuations for V. Let J be a left-closed interval of T with left endpoint equal to 0. Then a *J-trajectory* for V is a function $\tau : J \rightarrow \mathit{val}(V)$, such that for each $v \in V$, $\tau \downarrow v \in \mathit{dtype}(v)$. Here $\tau \downarrow v$ is the function with domain J defined by $(\tau \downarrow v)(t) = \tau(t)(v)$.

We say that a J-trajectory is *finite* if J is a finite interval, *closed* if J is a (finite) closed interval, and *full* if $J = \mathsf{T}^{\geq 0}$. A *trajectory* for V is a J-trajectory for V, for any J. We write $\mathit{trajs}(V)$ for the set of all trajectories for V. For T a set of trajectories, $\mathit{finite}(T)$, $\mathit{closed}(T)$ and $\mathit{full}(T)$ denote the subsets of finite, closed and full trajectories in T, respectively. A trajectory with domain $[0, 0]$ is called a *point* trajectory. If $\mathbf{v}$ is a valuation then $\wp(\mathbf{v})$ denotes the point trajectory that maps 0 to $\mathbf{v}$.

If τ is a trajectory then $\tau.\mathit{ltime}$, the *limit time* of τ, is the supremum of $\mathit{dom}(\tau)$. Similarly, we define $\tau.\mathit{fval}$, the *first valuation* of τ, to be $\tau(0)$, and if τ is closed, we define $\tau.\mathit{lval}$, the *last valuation* of τ, to be $\tau(\tau.\mathit{ltime})$. For τ a trajectory and $t \in \mathsf{T}^{\geq 0}$, we define $\tau \trianglelefteq t \triangleq \tau \lceil [0, t]$, $\tau \triangleleft t \triangleq \tau \lceil [0, t)$, and $\tau \trianglerighteq t \triangleq (\tau \lceil [t, \infty)) - t$. Note that the result of applying the above operations is always a trajectory, except when the result is a function with an empty domain. By convention, $\tau \trianglelefteq \infty \triangleq \tau$ and $\tau \triangleleft \infty \triangleq \tau$.

Prefix Ordering Trajectory τ is a *prefix* of trajectory υ, denoted by $\tau \leq \upsilon$, if τ can be obtained by restricting υ to a non-empty, downward closed subset of its domain. Formally, $\tau \leq \upsilon$ iff $\tau = \upsilon \lceil \mathit{dom}(\tau)$. For T a set of trajectories for V, $\mathit{pref}(T)$ denotes the *prefix closure* of T. We say that T is *prefix closed* if $T = \mathit{pref}(T)$.

The following lemma gives a simple domain theoretic characterization of the set of trajectories over a given set V. (See [7] for basic definitions and results on complete partially ordered sets, (cpo's)).

Lemma 1. *Let V be a set of variables. Then the set $\mathit{trajs}(V)$ of trajectories for V, together with the prefix ordering $\leq$, is an algebraic cpo whose compact elements are the closed trajectories.*

Concatenation The concatenation of two trajectories is obtained by taking the union of the first trajectory and the function obtained by shifting the domain of the second trajectory until the start time agrees with the limit time of the first trajectory; the last valuation of the first trajectory, which may not be the same as the first valuation of the second trajectory, is the one that appears in the concatenation. Formally, let τ, υ be trajectories, with τ closed. Then the *concatenation* is the function given by $\tau \frown \upsilon \triangleq \tau \cup (\upsilon \lceil (0, \infty) + \tau.\mathit{ltime})$. Using the closure of dynamic types under time shift and pasting, it follows that $\tau \frown \upsilon$ is a trajectory. Observe that $\tau \frown \upsilon$ is finite (resp. closed, full) iff υ is finite (resp. closed, full). Observe also that concatenation is associative.

The following lemma, which is easy to prove, shows the close connection between concatenation and the prefix ordering.

Lemma 2. *Let τ, υ be trajectories with τ closed. Then $\tau \leq \upsilon$ iff there exists a trajectory τ' such that $\tau \frown \tau'$.*

Note that if $\tau \leq \upsilon$, then the trajectory τ' such that $\upsilon = \tau \frown \tau'$ is unique except that it has an arbitrary value for $\tau'.\mathit{fval}$. Note also that the "$\Leftarrow$" implication would not hold if the first valuation of the second argument, rather than the last valuation of the first argument, were used in the concatenation.

Using a limit construction, we can generalize the definition of concatenation for any (finite or countably infinite) number of arguments. Let $\tau_0, \tau_1, \tau_2, \ldots$ be a (finite or infinite) sequence of trajectories, such that τ_i is closed for each nonfinal index i. Define trajectories $\tau'_0, \tau'_1, \tau'_2, \ldots$ by $\tau'_i \triangleq \tau_0 \frown \tau_1 \frown \cdots \frown \tau_i$. We define the *concatenation* $\tau_0 \frown \tau_1 \frown \tau_2 \ldots$ to be $\lim_{i \to \infty} \tau'_i$. It is easy to prove that $\tau_0 \frown \tau_1 \frown \tau_2 \ldots$ is a trajectory.

2.3 Hybrid Sequences

In this subsection, we introduce the notion of a *hybrid sequence*, which is used to model a combination of changes that occur instantaneously and changes that occur over intervals of time. Our definition is parameterized by a set A of *actions*, which are used to model instantaneous changes and instantaneous synchronization with the environment, and a set V of *variables*, which are used to model changes over intervals and continuous interaction. We also define some special kinds of hybrid sequences and operations on hybrid sequences.

Basic Definitions An (A, V)-*sequence* is a finite or infinite alternating sequence $\alpha = \tau_0\, a_1\, \tau_1\, a_2\, \tau_2 \cdots$, where (1) each τ_i is a trajectory in $\mathit{trajs}(V)$, (2) each a_i is an action in A, (3) if α is a finite sequence then it ends with a trajectory, and (4) if τ_i is not the last trajectory in α then $\mathit{dom}(\tau_i)$ is closed. We define a *hybrid sequence* to be an (A, V)-sequence for some A and V.

Since the trajectories in a hybrid sequence can be point trajectories, our notion of hybrid sequence allows a sequence of discrete actions to occur at the same real time, with corresponding changes of state.

If α is a hybrid sequence, with notation as above, then we define the *first valuation* of α, $\alpha.\mathit{fval}$, to be $\tau_0.\mathit{fval}$, and we define the *limit time* of α, $\alpha.\mathit{ltime}$, to be $\sum_i \tau_i.\mathit{ltime}$. A hybrid sequence α is defined to be:

- *time-bounded* if $\alpha.\mathit{ltime}$ is finite.
- *admissible* if $\alpha.\mathit{ltime} = \infty$.
- *closed* if α is a finite sequence and the domain of its final trajectory is a closed interval. In this case we define the *last valuation* of α, $\alpha.\mathit{lval}$, to be $\mathit{last}(\alpha).\mathit{lval}$.
- *Zeno* if α is neither closed nor admissible, that is, if α is time-bounded and is either an infinite sequence, or else a finite sequence ending with a trajectory whose domain is right-open.

Prefix Ordering We say that (A, V)-sequence $\alpha = \tau_0\, a_1\, \tau_1 \ldots$ is a *prefix* of (A, V)-sequence $\alpha' = \tau'_0\, a'_1\, \tau'_1 \ldots$, denoted by $\alpha \leq \alpha'$, if either $\alpha = \alpha'$, or α is a finite sequence ending in some τ_k; $\tau_i = \tau'_i$, and $a_{i+1} = a'_{i+1}$ for every i, $0 \leq i < k$; and $\tau_k \leq \tau'_k$. Like the set of trajectories over V, the set of (A, V)-sequences is a cpo.

Lemma 3. *The set of (A, V)-sequences together with the prefix ordering $\leq$ is an algebraic cpo with as compact elements the set of closed (A, V)-sequences.*

Restriction Let A, A' be sets of actions and V, V' sets of variables. The (A', V')-restriction of an (A, V)-sequence is obtained by projecting the trajectories on the variables in V', removing the actions not in A', and concatenating the adjacent trajectories.

Lemma 4. *Restriction is a continuous operation with respect to prefix ordering.*

Concatenation Suppose α and α' are (A, V)-sequences, with α closed. Then the *concatenation* is the (A, V)-sequence given by

$$\alpha \frown \alpha' \triangleq \mathit{init}(\alpha)\ (\mathit{last}(\alpha) \frown \mathit{head}(\alpha'))\ \mathit{tail}(\alpha').$$

(If σ is a nonempty sequence then $\mathit{head}(\sigma)$ denotes the first element of σ and $\mathit{tail}(\sigma)$ denotes σ with its first element removed; if σ is finite, then $\mathit{last}(\sigma)$ denotes the last element of σ and $\mathit{init}(\sigma)$ denotes σ with its last element removed.)

Lemma 5. *Let α, α' be (A, V)-sequences with α closed. Then $\alpha \leq \alpha'$ iff there exists and (A, V)-sequence α'' such that $\alpha' = \alpha \frown \alpha''$.*

Note that if $\alpha \leq \alpha'$, then the (A, V)-sequence α'' such that $\alpha' = \alpha \frown \alpha''$ is unique except that it has an arbitrary value in $\mathit{val}(V)$ for $\alpha''.\mathit{fval}$.

Based on Lemma 5 and Lemma 3, we can extend concatenation to infinitely many (A, V)-sequences as follows. Let $\alpha_1, \alpha_2, \ldots$ be an infinite sequence of closed (A, V)-sequences. Then define the *concatenation* $\alpha_1 \frown \alpha_2 \frown \cdots$ to be $\lim_{i \to \infty} \alpha'_i$, where $\alpha'_i = \alpha_1 \frown \alpha_2 \frown \cdots \frown \alpha_i$.

3 Hybrid Automata

As a preliminary step toward defining hybrid I/O automata, we first define a slightly more general *hybrid automaton* model. Hybrid automata classify actions as external and internal, but do not further subdivide the external actions into input and output actions. Likewise, they classify variables as external and internal. The input/output distinction is added in Section 4. In addition to defining hybrid automata, we here define an implementation relation between hybrid automata and a composition operation.

3.1 Definition of Hybrid Automata

A *hybrid automaton (HA)* $\mathcal{A} = (W, X, \Theta, E, H, \mathcal{D}, \mathcal{T})$ consists of:

- A set W of *external variables* and a set X of *internal variables*, disjoint from each other. We call a valuation $\mathbf{x}$ for X a *state*, and we refer to $\mathit{val}(X)$ as the set of states of $\mathcal{A}$. We write $V \triangleq W \cup X$. Given a valuation $\mathbf{v}$ for V, we denote by $\mathit{state}(\mathbf{v})$ the state $\mathbf{v} \lceil X$.

– A nonempty set $\Theta \subseteq val(X)$ of *start states*.
– A set E of *external actions* and a set H of *internal actions*, disjoint from each other. We write $A \triangleq E \cup H$ and let $a, b, \ldots$ range over A.
– A set $\mathcal{D} \subseteq val(X) \times A \times val(X)$ of *discrete transitions.* We use $\mathbf{x} \xrightarrow{a}_{\mathcal{A}} \mathbf{x}'$ as shorthand for $(\mathbf{x}, a, \mathbf{x}') \in \mathcal{D}$. We sometimes drop the subscript, and write $\mathbf{x} \xrightarrow{a} \mathbf{x}'$, when $\mathcal{A}$ should be clear from the context.
– A set $\mathcal{T}$ of trajectories for V. Given a trajectory $\tau \in \mathcal{T}$ we denote $\tau.fval \lceil X$ by $\tau.fstate$, and, if τ is closed, $\tau.lval \lceil X$ by $\tau.lstate$. We require that the following axioms hold:

T1 (Prefix closure) For every $\tau \in \mathcal{T}$ and every $\tau' \leq \tau$, $\tau' \in \mathcal{T}$.
T2 (Suffix closure) For every $\tau \in \mathcal{T}$ and every $t \in dom(\tau)$, $\tau \trianglerighteq t \in \mathcal{T}$.
T3 (Concatenation closure) Let $\tau_0, \tau_1, \tau_2, \ldots$ be a sequence of trajectories in $\mathcal{T}$ such that, for each nonfinal index i, τ_i is closed and $\tau_i.lstate = \tau_{i+1}.fstate$. Then $\tau_0 \frown \tau_1 \frown \tau_2 \ldots \in \mathcal{T}$.

Axioms **T1-3** express some natural closure properties on the set of trajectories that we need for our results about parallel composition. In a composed system, any trajectory of any component may be interrupted at any moment by a discrete transition of another component. Axiom **T1** ensures that the part of the trajectory up to the discrete transition is a trajectory, and axiom **T2** ensures the remainder is a trajectory. Axiom **T3** is required because the environment of a hybrid automaton, as a result of internal discrete transitions, may change its continuous dynamics repeatedly, and the automaton must be able to follow this behavior. Even without performing discrete transitions itself, a hybrid automaton must be able to follow this type of behavior of its environment. In the earlier definition of hybrid automata presented in [15, 16], we used a special stuttering action e in place of axiom **T3**; this gave rise to technical complications.

Another major difference between our new definition and the earlier one is that the external variables are no longer considered to be part of the state; thus, for instance, the discrete transitions do not depend on the values of these variables. Analogous to the way in which external actions can be used to model synchronization of discrete transitions of different components, external variables allow us to model synchronization of continuous activity ("flow") between components. Because the external actions and external variables are not part of the state, we think of them as "ephemeral".

We often denote the components of a HA $\mathcal{A}$ by $W_{\mathcal{A}}$, $X_{\mathcal{A}}$, $\Theta_{\mathcal{A}}$, $E_{\mathcal{A}}$, etc, and the components of a HA $\mathcal{A}_i$ by W_i, X_i, Θ_i, E_i, etc. We sometimes omit these subscripts, where no confusion seems likely.

3.2 Executions and Traces

We now define execution fragments, executions, trace fragments, and traces, which are used to describe automaton behavior.

An *execution fragment* of a HA $\mathcal{A}$ is an (A, V)-sequence $\alpha = \tau_0\, a_1\, \tau_1\, a_2\, \tau_2 \cdots$, where (1) each τ_i is a trajectory in $\mathcal{T}$, and (2) if τ_i is not the last trajectory in

α then $\tau_i.\mathit{lstate} \overset{a_{i+1}}{\rightarrow} \tau_{i+1}.\mathit{fstate}$. An execution fragment records all the instantaneous, discrete state changes that occur during a specific evolution of a system, as well as the state changes and external variable changes that occur while time advances. We write $\mathit{frags}_{\mathcal{A}}$ for the set of all execution fragments of $\mathcal{A}$.

If α is an execution fragment, with notation as above, then we define the *first state* of α, $\alpha.\mathit{fstate}$, to be $\mathit{state}(\alpha.\mathit{fval})$, or equivalently, $\tau_0.\mathit{fstate}$. An execution fragment α is defined to be an *execution* if $\alpha.\mathit{fstate}$ is a start state, that is, is in Θ. We write $\mathit{execs}_{\mathcal{A}}$ for the set of all executions of $\mathcal{A}$.

If α is a closed execution fragment then we define the *last state* of α, $\alpha.\mathit{lstate}$, to be $\mathit{state}(\alpha.\mathit{lval})$, or equivalently, $\mathit{last}(\alpha).\mathit{lstate}$. A state of $\mathcal{A}$ is *reachable* if it is the last state of some closed execution of $\mathcal{A}$.

Lemma 6. *Let α and α' be execution fragments of $\mathcal{A}$ with α closed, and such that $\alpha.\mathit{lstate} = \alpha'.\mathit{fstate}$. Then $\alpha \frown \alpha'$ is an execution fragment of $\mathcal{A}$.*

Lemma 7. *Let α and α' be execution fragments of $\mathcal{A}$ with α closed. Then $\alpha \leq \alpha'$ iff there is an execution fragment α'' such that $\alpha' = \alpha \frown \alpha''$.*

The trace of an execution fragment records the external actions and the evolution of external variables. Formally, if α is an execution fragment, then the *trace* of α, denoted by $\mathit{trace}(\alpha)$, is the (E, W)-restriction of α. A *trace fragment* of a hybrid automaton $\mathcal{A}$ *from* a state $\mathbf{x}$ of $\mathcal{A}$ is a trace that arises from an execution fragment of $\mathcal{A}$ whose first state is $\mathbf{x}$. We write $\mathit{tracefrags}_{\mathcal{A}}(\mathbf{x})$ for the set of trace fragments of $\mathcal{A}$ from $\mathbf{x}$. Also, we define a *trace* of $\mathcal{A}$ to be a trace fragment from an initial state, that is, a trace that arises from an execution of $\mathcal{A}$, and write $\mathit{traces}_{\mathcal{A}}$ for the set of traces of $\mathcal{A}$.

Hybrid automata $\mathcal{A}_1$ and $\mathcal{A}_2$ are *comparable* if they have the same external actions and variables, that is, if $W_1 = W_2$ and $E_1 = E_2$. If $\mathcal{A}_1$ and $\mathcal{A}_2$ are comparable then we say that $\mathcal{A}_1$ *implements* $\mathcal{A}_2$, denoted by $\mathcal{A}_1 \leq \mathcal{A}_2$, if the traces of $\mathcal{A}_1$ are included among those of $\mathcal{A}_2$, that is, if $\mathit{traces}_{\mathcal{A}_1} \subseteq \mathit{traces}_{\mathcal{A}_2}$.

3.3 Simulation Relations

Let $\mathcal{A}$ and $\mathcal{B}$ be comparable HAs. A *simulation* from $\mathcal{A}$ to $\mathcal{B}$ is a relation $R \subseteq \mathit{val}(X_{\mathcal{A}}) \times \mathit{val}(X_{\mathcal{B}})$ satisfying the following conditions, for all states $\mathbf{x}_A$ and $\mathbf{x}_B$ of $\mathcal{A}$ and $\mathcal{B}$, respectively:

1. If $\mathbf{x}_A \in \Theta_A$ then there exists a state $\mathbf{x}_B \in \Theta_B$ such that $\mathbf{x}_A \; R \; \mathbf{x}_B$.
2. If $\mathbf{x}_A \; R \; \mathbf{x}_B$, $\mathbf{x}_A \overset{a}{\rightarrow}_A \mathbf{x}'_A$ and $\tau = \mathit{trace}(\wp(\mathbf{x}_A) \; a \; \wp(\mathbf{x}'_A))$, then $\mathcal{B}$ has a closed execution fragment α with $\alpha.\mathit{fstate} = \mathbf{x}_B$, $\mathit{trace}(\alpha) = \mathit{trace}(\tau)$, and $\mathbf{x}'_A \; R \; \alpha.\mathit{lstate}$.
3. If $\mathbf{x}_A \; R \; \mathbf{x}_B$ and τ is a closed trajectory of $\mathcal{A}$ with $\mathbf{x}_A = \tau.\mathit{fstate}$ and $\mathbf{x}'_A = \tau.\mathit{lstate}$, then $\mathcal{B}$ has a closed execution fragment α with $\alpha.\mathit{fstate} = \mathbf{x}_B$, $\mathit{trace}(\alpha) = \mathit{trace}(\tau)$, and $\mathbf{x}'_A \; R \; \alpha.\mathit{lstate}$.

Lemma 8. *Let $\mathcal{A}$ and $\mathcal{B}$ be comparable HAs, and let R be a simulation from $\mathcal{A}$ to $\mathcal{B}$. Let $\mathbf{x}_A$ and $\mathbf{x}_B$ be states of $\mathcal{A}$ and $\mathcal{B}$, respectively, such that $\mathbf{x}_A \; R \; \mathbf{x}_B$. Then $\mathit{tracefrags}_{\mathcal{A}}(\mathbf{x}_A) \subseteq \mathit{tracefrags}_{\mathcal{B}}(\mathbf{x}_B)$.*

Theorem 1. *Let $\mathcal{A}$ and $\mathcal{B}$ be comparable HAs, and let R be a simulation from $\mathcal{A}$ to $\mathcal{B}$. Then $traces_{\mathcal{A}} \subseteq traces_{\mathcal{B}}$.*

3.4 Composition

We now introduce the operation of composition for hybrid automata, which allows an automaton representing a complex system to be constructed by composing automata representing individual system components. We prove that the composition operation respects our implementation relationship (inclusion of sets of traces). Our composition operation identifies actions and variables with the same name in different component automata. When any component automaton performs a step involving an action a, so do all component automata that have a in their signatures. Common variables are shared among the components.

We define composition as a partial, binary operation on hybrid automata. Since internal actions of an automaton $\mathcal{A}_1$ are intended to be unobservable by any other automaton $\mathcal{A}_2$, we do not allow $\mathcal{A}_1$ to be composed with $\mathcal{A}_2$ unless the internal actions of $\mathcal{A}_1$ are disjoint from the actions of $\mathcal{A}_2$. Also, we require disjointness of the internal variables of $\mathcal{A}_1$ and the variables of $\mathcal{A}_2$. Formally, we say that hybrid automata $\mathcal{A}_1$ and $\mathcal{A}_2$ are *compatible* if for $i \neq j$, $X_i \cap V_j = H_i \cap A_j = \emptyset$. If $\mathcal{A}_1$ and $\mathcal{A}_2$ are compatible then their *composition* $\mathcal{A}_1 \| \mathcal{A}_2$ is defined to be the structure $\mathcal{A} = (W, X, \Theta, E, H, \mathcal{D}, \mathcal{T})$ where

- $W = W_1 \cup W_2$, $X = X_1 \cup X_2$, $E = E_1 \cup E_2$, $H = H_1 \cup H_2$.
- $\Theta = \{\mathbf{x} \in val(X) \mid \mathbf{x} \lceil X_1 \in \Theta_1 \wedge \mathbf{x} \lceil X_2 \in \Theta_2\}$.
- For each $\mathbf{x}, \mathbf{x}' \in val(X)$ and each $a \in A$, $\mathbf{x} \xrightarrow{a}_{\mathcal{A}} \mathbf{x}'$ iff for $i = 1, 2$, either (1) $a \in A_i$ and $\mathbf{x} \lceil X_i \xrightarrow{a}_i \mathbf{x}' \lceil X_i$, or (2) $a \notin A_i$ and $\mathbf{x} \lceil X_i = \mathbf{x}' \lceil X_i$.
- $\mathcal{T} \subseteq trajs(V)$ is given by $\tau \in \mathcal{T} \Leftrightarrow \tau \downarrow V_1 \in \mathcal{T}_1 \ \wedge \ \tau \downarrow V_2 \in \mathcal{T}_2$.

Proposition 1. *$\mathcal{A}_1 \| \mathcal{A}_2$ is a hybrid automaton.*

Theorem 2. *Suppose $\mathcal{A}_1, \mathcal{A}_2$ and $\mathcal{B}$ are HAs with $\mathcal{A}_1 \leq \mathcal{A}_2$, and suppose that each of $\mathcal{A}_1$ and $\mathcal{A}_2$ is compatible with $\mathcal{B}$. Then $\mathcal{A}_1 \| \mathcal{B} \leq \mathcal{A}_2 \| \mathcal{B}$.*

In the full version of this paper, we define two natural hiding operations on HAs, which hide external actions and external variables, respectively, and prove that these operations also respect the implementation preorder.

4 Hybrid I/O Automata

In this section we specialize the hybrid automaton model of Section 3 by adding a distinction between input and output.

4.1 Definition of Hybrid I/O Automata

A *hybrid I/O automaton (HIOA) $\mathcal{A}$* is a tuple $(\mathcal{H}, U, Y, I, O)$ where

- $\mathcal{H} = (W, X, \Theta, E, H, \mathcal{D}, \mathcal{T})$ is a hybrid automaton.

- U and Y partition W into *input* and *output* variables, respectively. Variables in $Z \triangleq X \cup Y$ are called *locally controlled*; as before we write $V \triangleq W \cup X$.
- I and O partition E into *input* and *output actions*, respectively. Actions in $L \triangleq H \cup O$ are called *locally controlled*; as before we write $A \triangleq E \cup H$.
- The following additional axioms are satisfied:

  E1 (Input action enabling)
  For all $\mathbf{x} \in \mathit{val}(X)$ and all $a \in I$ there exists $\mathbf{x}'$ such that $\mathbf{x} \xrightarrow{a} \mathbf{x}'$.

  E2 (Input flow enabling)
  For all $\mathbf{x} \in \mathit{val}(X)$ and $v \in \mathit{trajs}(U)$, there exists $\tau \in \mathcal{T}$ such that $\tau.\mathit{fstate} = \mathbf{x}, \tau \downarrow U \leq v$, and either
  1. $\tau \downarrow U = v$, or
  2. there exist $t \in \mathit{dom}(\tau)$ and $l \in L$ such that l is enabled from $\tau(t)$.

Input action enabling is the input enabling condition of ordinary I/O automata. Input flow enabling is a new corresponding condition for continuous interaction. It says that an HIOA should be able to accept any continuous input flow, either by letting time advance for the entire duration of the input flow, or by reacting with a locally controlled action after some part of the input flow has occurred.

An *execution* of an HIOA $\mathcal{A}$ is an execution of $\mathcal{H}_{\mathcal{A}}$. Similarly, a *trace* of $\mathcal{A}$ is a trace of $\mathcal{H}_{\mathcal{A}}$. Two HIOAs $\mathcal{A}_1$ and $\mathcal{A}_2$ are *comparable* if their inputs and outputs coincide, that is, if $I_1 = I_2$, $O_1 = O_2$, $U_1 = U_2$, and $Y_1 = Y_2$. If $\mathcal{A}_1$ and $\mathcal{A}_2$ are comparable, then $\mathcal{A}_1 \leq \mathcal{A}_2$ is defined to mean that the traces of $\mathcal{A}_1$ are included among those of $\mathcal{A}_2$: $\mathcal{A}_1 \leq \mathcal{A}_2 \triangleq \mathit{traces}_{\mathcal{A}_1} \subseteq \mathit{traces}_{\mathcal{A}_2}$. If $\mathcal{A}_1$ and $\mathcal{A}_2$ are comparable HIOAs then $\mathcal{H}_1$ and $\mathcal{H}_2$ are comparable and $\mathcal{A}_1 \leq \mathcal{A}_2$ iff $\mathcal{H}_1 \leq \mathcal{H}_2$.

The definition of simulation for HIOAs is the same as for HAs, and the soundness result carries over immediately to the enriched setting.

4.2 Composition

The definition of composition for HIOAs builds on the corresponding definition for HAs, but also takes the input/output structure into account. Just as in the definition of compatibility for HAs, we do not allow an HIOA $\mathcal{A}_1$ to be composed with an HIOA $\mathcal{A}_2$ unless the internal actions and variables of $\mathcal{A}_1$ are disjoint from the actions and variables, respectively, of $\mathcal{A}_2$. In addition, in order that the composition operation might satisfy nice properties (such as Theorem 7), we require that at most one component automaton "controls" any given action or variable; that is, we do not allow $\mathcal{A}_1$ and $\mathcal{A}_2$ to be composed unless the sets of output actions of $\mathcal{A}_1$ and $\mathcal{A}_2$ are disjoint and the sets of output variables of $\mathcal{A}_1$ and $\mathcal{A}_2$ are disjoint.

If $\mathcal{A}_1$ and $\mathcal{A}_2$ are compatible then their *composition* $\mathcal{A}_1 \| \mathcal{A}_2$ is defined to be the tuple $\mathcal{A} = (\mathcal{H}, U, Y, I, O)$ where $\mathcal{H} = \mathcal{H}_1 \| \mathcal{H}_2$, $U = (U_1 \cup U_2) - (Y_1 \cup Y_2)$, $Y = Y_1 \cup Y_2$, $I = (I_1 \cup I_2) - (O_1 \cup O_2)$, and $O = O_1 \cup O_2$.

The definition of compatibility given above is not quite strong enough to imply that the composition of two HIOAs is actually an HIOA. Thus, we define a stronger notion and say that compatible HIOAs $\mathcal{A}_1$ and $\mathcal{A}_2$ are *strongly compatible* if $\mathcal{A}_1 \| \mathcal{A}_2$ satisfies axiom **E2**. Strong compatibility implies that the reaction of the composed automaton to any input flow v must be the result of a deliberate reaction by either $\mathcal{A}_1$ or $\mathcal{A}_2$. That is, either both $\mathcal{A}_1$ and $\mathcal{A}_2$ accept v in its entirety, or one of the two reacts with a locally controlled action. No "time deadlock" is allowed due to incompatible reactions of $\mathcal{A}_1$ and $\mathcal{A}_2$.

Proposition 2. *The composition of two strongly compatible HIOAs is an HIOA.*

Theorem 3. *Suppose $\mathcal{A}_1, \mathcal{A}_2$ and $\mathcal{B}$ are HIOAs with $\mathcal{A}_1 \leq \mathcal{A}_2$, and each of $\mathcal{A}_1$ and $\mathcal{A}_2$ is strongly compatible with $\mathcal{B}$. Then $\mathcal{A}_1 \| \mathcal{B} \leq \mathcal{A}_2 \| \mathcal{B}$.*

5 Receptive Hybrid I/O Automata

In this section we adapt the notion of receptiveness [20] to our new framework. Informally speaking, a system is receptive provided that it admits a strategy for resolving its nondeterministic choices that never generates infinitely many locally controlled actions in finite time. An important consequence of this definition is that a receptive HIOA has some response defined for any sequence of discrete and continuous input. We show that receptiveness is closed under composition. Because of the improvements in our new model, the treatment of receptiveness in this paper is simpler than that in [20]; however, we only address admissibility here, and not general liveness properties as in [20].

An execution fragment of an HIOA is *locally-Zeno* if it is Zeno and contains infinitely many locally controlled actions. An HIOA $\mathcal{A}$ is *locally-Zeno* if it has at least one locally-Zeno execution fragment. In the rest of the paper we will be interested mainly in *non-locally-Zeno* HIOAs, that is, HIOAs that are not locally-Zeno. We use non-locally-Zeno HIOAs as the basis for defining receptiveness.

Theorem 4. *Let $\mathcal{A}_1$, $\mathcal{A}_2$ be strongly compatible non-locally-Zeno HIOAs. Then $\mathcal{A}_1 \| \mathcal{A}_2$ is also non-locally-Zeno.*

Theorem 5. *Let $\mathcal{A}$ be a non-locally-Zeno HIOA. Then, for each (I, U)-sequence β and each state $\mathbf{x}$, there is an execution fragment α of $\mathcal{A}$ such that (1) $\alpha.fstate = \mathbf{x}$, (2) $\alpha \lceil (I, U) = \beta$.*

The property stated in Theorem 5 is known in the literature as *I/O feasibility* [17]; it implies that any finite execution can be extended to an admissible execution, no matter what the environment does.

A *strategy* for an HIOA $\mathcal{A}$ is an HIOA $\mathcal{A}'$ that differs from $\mathcal{A}$ only in that $\mathcal{D}' \subseteq \mathcal{D}$ and $\mathcal{T}' \subseteq \mathcal{T}$. A strategy $\mathcal{A}'$ for an HIOA $\mathcal{A}$ can be viewed as a nondeterministic memoryless strategy in the sense of [5, 20] that chooses some of the evolutions that are possible from each of the states of $\mathcal{A}$. The fact that the states of $\mathcal{A}$ and $\mathcal{A}'$ are the same ensures that $\mathcal{A}'$ chooses evolutions for every state $\mathbf{x}$ of $\mathcal{A}$.

We say that an HIOA is *receptive* if it has a non-locally-Zeno strategy.

Theorem 6. *A receptive HIOA is I/O feasible.*

Theorem 7. *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two compatible receptive HIOAs with two strongly compatible non-locally-Zeno strategies $\mathcal{A}'_1$ and $\mathcal{A}'_2$, respectively. Then $\mathcal{A}_1 \| \mathcal{A}_2$ is a receptive HIOA with non-locally-Zeno strategy $\mathcal{A}'_1 \| \mathcal{A}'_2$.*

6 Sufficient Conditions for Strong Compatibility

In order to apply Theorem 7, one has to establish that two strategies are strongly compatible. This is difficult in general since it requires checking compatibility between the continuous dynamics of two systems. However, for certain restricted classes of HIOAs, strong compatibility follows directly from compatibility.

6.1 HIOAs with Restrictions on Input Variables

Our first example is the class of HIOAs without input variables. It is routine to verify that two HIOAs without input variables are strongly compatible iff they are compatible. From the perspective of classical control theory a system without input variables is uninteresting because it cannot be controlled; in a hybrid setting, however, a system without input variables can still interact with its environment via discrete input actions. *Linear hybrid automata* [1], for instance, have no input variables.

Another example is the class of *autistic* HIOAs—those for which the values of output variables do not depend on the values of input variables. Formally, an HIOA $\mathcal{A}$ is called *autistic* if for all $\tau \in \mathcal{T}$ and all $\upsilon \in \mathit{trajs}(U)$ such that $\mathit{dom}(\tau) = \mathit{dom}(\upsilon)$ there exists $\tau' \in T$ such that $\tau' \downarrow U = \upsilon$ and $\tau' \downarrow Y = \tau \downarrow Y$.

6.2 Lipschitz HIOAs

In this section, we define *Lipschitz HIOAs*, based on systems of differential equations using Lipschitz functions. We give examples of conditions on classes of Lipschitz HIOAs that imply strong compatibility. The ideas are derived from methods in the literature on control theory [21]. In control theory, continuous system behavior is typically defined using differential equations of the form:

$$D \triangleq \begin{cases} \dot{x} = f(x, u) \\ y = g(x) \end{cases}$$

where u, y, and x are the vectors of input, output, and state variables, respectively, together with a starting condition of the form $x(0) = x_0$.

To ensure that the system's behavior is defined, the differential equations must admit a solution for each possible starting condition. The following theorem from calculus gives sufficient conditions for a solution to exist.

Theorem 8 (Local existence). *If f is globally Lipschitz and u is $\mathcal{C}^1$, then for each starting condition $x(0) = x_0$ there is a unique solution to the equations of D, defined on a maximal neighborhood of 0, such that $x(0) = x_0$.*

Observe that, since the set of globally Lipschitz functions is closed under composition, the local existence theorem is valid also when the variables u are the result of a globally Lipschitz function applied to a C^1 function.

Suppose two interacting systems are described by sets of equations D_1 and D_2 of the form given above. Then their combined behavior can be described by the union of the sets of equations D_1 and D_2. It is easy to show that, if the functions occurring in D_1 and D_2 are globally Lipschitz, and D_1 and D_2 do not have any common output and state variables, then the union of these two sets of equations is expressible in the same form with functions that are globally Lipschitz. Thus, in this case no additional machinery is needed to prove that the behavior of the interacting systems is well defined. We define a set D of equations to be Lipschitz if functions f and g are globally Lipschitz.

To extend the above ideas to the hybrid case we define the notion of a Lipschitz HIOA. An HIOA $\mathcal{A}$ is *Lipschitz* if there is a subset M of its state variables (we call these the *mode variables*) such that:

L1 The dynamic type of each variable in M is piecewise constant.
L2 The dynamic type of each variable not in M is a subset of the set of real-valued functions defined on left-closed intervals of the reals that can be expressed in the form $h(c(\cdot))$ where h is a globally Lipschitz function and c is a C^1 function, closed under pasting.
L3 The values of the M variables are constant in each trajectory of $\mathcal{T}$.
L4 For each valuation $\mathbf{m}$ of M there is a Lipschitz system of equations $D_{\mathbf{m}}$ with input variables U, output variables Y, and state variables $X - M$ such that the following holds: If trajectory τ of $\mathcal{T}$ starts from a state $\mathbf{x}$ with $\mathbf{x} \lceil M = \mathbf{m}$, then $\tau \lceil V - M$ is expressible as the concatenation of countably many trajectories $\tau_0, \tau_1, \ldots$, where each τ_i is a solution to $D_{\mathbf{m}}$.

Define a Lipschitz HIOA to be *input bounded* if for each input variable u there exists a positive real value B such that every function in the dynamic type of u has range in $[-B, B]$.

Lemma 9. *Compatible input-bounded Lipschitz HIOAs are strongly compatible.*

Theorem 9. *The composition of two compatible input-bounded Lipschitz HIOAs is a Lipschitz HIOA.*

Theorem 10. *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be compatible receptive HIOAs with non-locally-Zeno, input-bounded, Lipschitz strategies. Then $\mathcal{A}_1 \| \mathcal{A}_2$ is a receptive HIOA with a non-locally-Zeno input-bounded Lipschitz strategy.*

Theorem 11. *The composition of two compatible receptive input-bounded Lipschitz HIOAs is a receptive input-bounded Lipschitz HIOA.*

The conclusion that we derive from Theorem 11 is that compatibility implies strong compatibility if we describe the continuous behaviors of HIOAs by means of differential equations of the form of D with functions f and g globally Lipschitz. In general, any choice of conditions on f, g, and u that guarantees local existence of unique solutions, continuity of solutions, and that is preserved by interaction between systems, can be used to define a class of automata for which strong compatibility follows from compatibility.

References

1. R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J.Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.
2. R. Alur and T.A. Henzinger. Reactive modules. *Proc. LICS'96*, pp. 207–218, 1996.
3. R. Alur and T.A. Henzinger. Modularity for timed and hybrid systems. In *Proc. of CONCUR'97*, LNCS 1243, pp. 74–88, 1997.
4. D.J.B. Bosscher, I. Polak, and F.W. Vaandrager. Verification of an audio control protocol. In *Proc. of FTRTFT'94*, LNCS 863, pp. 170–192, 1994.
5. D. Dill. *Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits.* ACM Distinguished Dissertations. MIT Press, 1988.
6. E. Dolginova and N.A. Lynch. Safety verification for automated platoon maneuvers: A case study. *Proc. of HART'97*, LNCS 1201, pp. 154–170, 1997.
7. C.A. Gunter. *Semantics of Programming Languages: Structures and Techniques.* MIT Press, Cambridge, Massachusetts, 1992.
8. A. Kapur, T.A. Henzinger, Z. Manna, and A. Pnueli. Proving safety properties of hybrid systems. In *Proc. of FTRTFT'94*, LNCS 863, pp. 431–454, 1994.
9. C. Livadas, J. Lygeros, and N.A. Lynch. High-level modelling and analysis of TCAS. In *Proc. of RTSS'99*, 1999.
10. C. Livadas and N.A. Lynch. Formal verification of safety-critical hybrid systems. In *Proc. of HSCC'98*, LNCS 1386, pp. 253–272, 1998.
11. J. Lygeros and N.A. Lynch. On the formal verification of the TCAS conflict resolution algorithms. In *Proc. of 36th IEEE Conference on Decision and Control*, pp. 1829–1834, 1997. Extended abstract.
12. J. Lygeros and N.A. Lynch. Strings of vehicles: Modeling and safety conditions. In *Proc. of HSCC'98*, LNCS 1386, pp. 273–288, 1998.
13. N.A. Lynch. Modelling and verification of automated transit systems, using timed automata, invariants and simulations. In *Hybrid Systems III*, LNCS 1066, 1996.
14. N.A. Lynch. A three-level analysis of a simple acceleration maneuver, with uncertainties. *Proc. of 3$^{\text{rd}}$ AMAST Workshop on Real-Time Systems*, pp. 1–22, 1996.
15. N.A. Lynch, R. Segala, F.W. Vaandrager, and H.B. Weinberg. Hybrid I/O automata. In *Hybrid Systems III*, LNCS 1066, pp. 496–510, 1996.
16. N.A. Lynch, R. Segala, F.W. Vaandrager, and H.B. Weinberg. Hybrid I/O automata. Report CSI-R9907, Computing Science Institute, Univ. of Nijmegen, 1999.
17. N.A. Lynch and F.W. Vaandrager. Action transducers and timed automata. *Formal Aspects of Computing*, 8(5):499–538, 1996.
18. N.A. Lynch and H.B. Weinberg. Proving correctness of a vehicle maneuver: Deceleration. *Proc. 2$^{\text{nd}}$ European Workshop on Real-Time and Hybrid Systems*, 1995.
19. O. Maler, Z. Manna, and A. Pnueli. From timed to hybrid systems. In *Proc. REX Workshop on Real-Time: Theory in Practice*, LNCS 600, pp. 447–484, 1992.
20. R. Segala, R. Gawlick, J.F. Søgaard-Andersen, and N.A. Lynch. Liveness in timed and untimed systems. *Information and Computation*, 141(2):119–171, March 1998.
21. E.D. Sontag. *Mathematical Control Theory — Deterministic Finite Dimensional Systems*, volume 6 of *Texts in Applied Mathematics.* Springer-Verlag, 1990.
22. H.B. Weinberg and N.A. Lynch. Correctness of vehicle control systems: A case study. In *Proc. RTSS'96*, pp. 62–72, 1996.
23. H.B. Weinberg, N.A. Lynch, and N. Delisle. Verification of automated vehicle protection systems. In *Hybrid Systems III*, LNCS 1066, pp. 101–113,1996.