Hybrid I/O automata

N. Lynch, R. Segala, F. Vaandrager and H.B. Weinberg

Computer Science/Department of Software Technology

CS-R9578 1995
Hybrid I/O Automata
(extended abstract)

Nancy Lynch
MIT Laboratory for Computer Science
Cambridge, MA 02139, USA
lynch@theory.lcs.mit.edu

Roberto Segala
Dipartimento di Matematica, Università di Bologna
Piazza di Porta San Donato 5, 40127 Bologna, Italy
segala@cs.unibo.it

Frits Vaandrager
CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
fritsv@cwi.nl

H.B. Weinberg
MIT Laboratory for Computer Science
Cambridge, MA 02139, USA
hb@theory.lcs.mit.edu

Abstract
We propose a new hybrid I/O automaton model that is capable of describing both continuous and discrete behavior. The model, which extends the timed I/O automaton model of [12, 7] and the phase transition system models of [15, 2], allows communication among components using both shared variables and shared actions. The main contributions of this paper are: (1) the definition of hybrid I/O automata and of an implementation relation based on hybrid traces, (2) the definition of a simulation between hybrid I/O automata and a proof that existence of a simulation implies the implementation relation, (3) a definition of composition of hybrid I/O automata and a proof that it respects the implementation relation, and (4) a definition of receptiveness for hybrid I/O automata and a proof that, assuming certain compatibility conditions, receptiveness is preserved by composition.

AMS Subject Classification (1991): 68Q05, 68Q60, 93A25, 93C83.
Keywords & Phrases: Hybrid systems, I/O automata, Simulation relations, Compositionality, I/O behaviors, Receptiveness.

Notes: The first author was supported by NSF Grant 9225124-CCR. U.S. Department of Transportation Contract DTRS595-0001-YR.8, AFOSR-ONR Contract F49620-94-1-0199, and ARPA Contracts F19628-95-C-0118 and N00014-92-J-4033. The second author's current affiliation is: Computing Science Institute, University of Nijmegen, P.O. Box 9010, 9500 GL Nijmegen, The Netherlands. The research of the fourth author was partially supported by a National Science Foundation Graduate Fellowship. This paper will appear in R. Alur, T. Henzinger, and E. Sontag, editors, Hybrid Systems III, LNCS, Springer-Verlag, 1996.
1. Introduction
In recent years, there has been a fast growing interest in hybrid systems [8, 18] — systems that contain both discrete and continuous components, typically computers interacting with the physical world. Because of the rapid development of processor and circuit technology, hybrid systems are becoming common in many application domains, including avionics, process control, robotics and consumer electronics. Motivated by a desire to formally specify and verify real-life applications, we are generalizing existing methods from computer science to the setting of hybrid systems. We are applying our results in a number of projects in the areas of personal rapid transit [14, 10, 20], intelligent vehicle highway systems, and consumer electronics [5].

Within the theory of reactive systems, which has been developed in computer science during the last 20 years, it is common to represent both a system and its properties as abstract machines (see, for instance [11, 4, 9]). A system is then defined to be correct if the abstract machine for the system implements the abstract machine for the specification in the sense that the set of behaviors of the first is included in that of the second. A major reason why this approach has been successful is that it supports stepwise refinement: systems can be specified in a uniform way at many levels of abstraction, from a description of their highest-level properties to a description of their implementation in terms of circuitry, and the various specifications can be related formally using the implementation relation. In this paper we generalize this and related ideas from the theory of reactive systems to the setting of hybrid systems. More specifically, we propose answers to the following four questions:

1. What system model do we use?
2. What implementation relation do we use?
3. How do we compose systems?
4. What does it mean for a system to be receptive?

The system model. Our new hybrid I/O automaton (HIOA) model is based on infinite state machines. The model allows both discrete state jumps, described by a set of labelled transitions, and continuous state changes, described by a set of trajectories. To describe the external interface of a system, the state variables are partitioned into input, output and internal variables, and the transition labels (or actions) are partitioned into input, output and internal actions. Our model is very general and contains no finiteness restrictions. More structure will have to be added in order to deal with applications, but the general model that we propose allows us to answer questions 2–4. HIOA's are inspired by the timed I/O automata of [12, 7] and the phase transition system models of [15, 2]. The main difference between HIOA’s and timed I/O automata is that, as in phase transition systems, trajectories are primitive in our model and not a derived notion. In the work on phase transition systems the main emphasis thus far has been on temporal logics and model checking. Questions 2–4 have not been addressed and perhaps for this reason the external interface is not an integral part of a phase transition system.
The implementation relation. The implementation relation that we propose is simply inclusion of the sets of hybrid traces. A hybrid trace records occurrences of input and output actions, and the evolution of input and output variables during an execution of a system. Thus HIOA \( B \) implements HIOA \( A \) if every behavior of \( B \) is allowed by \( A \). In this case, \( B \) is typically more deterministic than \( A \), both at the discrete and the continuous level. For instance, \( A \) might produce an output at an arbitrary time before noon, whereas \( B \) produces an output sometime between 10 and 11AM. Or \( A \) might allow any smooth trajectory for output variable \( y \) with \( y \in [0, 2] \), whereas \( B \) only allows trajectories with \( y = 1 \).

Within computer science, simulation relations provide a major technical tool to prove inclusion of behaviors between systems (see [13] for an overview). In this paper we propose a definition of a simulation between HIOA’s and show that existence of a simulation implies the implementation relation.

Composition. Within computer science various notions of composition have been proposed for models based on transition systems. One popular approach is to use the product construction from classical automata theory and to synchronize on common transition labels ("actions") [11]. In other approaches there are no transition labels to synchronize on, and communication between system components is achieved via shared variables [16, 9]. Shared action and shared variable communication are equally expressive, and the relationships between the two mechanisms are well understood: it depends on the application which of the two is more convenient to use. In control theory studies of dynamic feedback, communication between components is typically achieved via a connection map, which specifies how outputs and inputs of components are wired [19]. This communication mechanism can be expressed naturally using shared variables. Since we find it convenient to use communication via shared actions in the applications that we work on, our model supports both shared action and shared variable communication. Whereas shared actions always correspond to discrete transitions, shared variables can be used equally well for communication of continuously varying signals and for signals that can only change value upon occurrence of a discrete transition.

We prove that our composition operator respects the implementation relation: if \( A_1 \) implements \( A_2 \) then \( A_1 \) composed with \( B \) implements \( A_2 \) composed with \( B \). Such a result is essential for compositional design and verification of systems.

Receptiveness. The class of HIOA’s is very general and allows for systems with bizarre timing behavior. We can describe systems in which time cannot advance at all or in which time advances in successively smaller increments but never beyond a certain bound, so called Zeno behavior. We do not want to accept such systems as valid implementations of any specification since, clearly, they will have no physical realization. Therefore we only accept receptive HIOA’s as implementations, i.e., HIOA’s in which time can advance to infinity independently of the input provided by the environment. Inspired by earlier work of [6, 1, 7] on (timed) discrete event systems, we define receptivity in terms of a game between system and environment in which the goal of the system is to construct an infinite, non-Zeno execution, and the goal of the environment is to prevent this. It is interesting to compare our games with the games of Nerode and Yakhnis [17]. Since the purpose of the latter games is the extraction of digital control to meet performance specifications, the environment player
may choose all disturbances. Irrespective of the disturbances the system should realize a given performance specification. The purpose of our games is to show that regardless of the input provided by its environment, a HIOA can exhibit proper behavior. Therefore, in our games the system resolves all nondeterminism due to internal disturbances (which express implementation freedom), even though the environment may choose all the input signals.

The main technical result that we prove about receptivity is that, assuming certain compatibility conditions, receptiveness is preserved by composition.

2. HYBRID I/O AUTOMATA AND THEIR BEHAVIOR
In this section we introduce HIOA's and define an implementation relation between these automata. Since the notion of a trajectory plays an important role in the model, we start out with the definition of trajectories and some operations on them.

2.1 Trajectories
Throughout this paper, we fix a time axis $T$, which is a subgroup of $(\mathbb{R}, +)$, the real numbers with addition. Usually, $T = \mathbb{R}$ or $\mathbb{Z}$, but also the degenerated time axis $T = \{0\}$ is allowed. An interval $I$ is a convex subset of $T$. We denote intervals as usual: $[t_1, t_2] = \{t \in T \mid t_1 \leq t \leq t_2\}$, etc. For $I$ an interval and $t \in T$, we define $I + t = \{t' + t \mid t' \in I\}$.

We assume a universal set $\mathcal{V}$ of variables. Variables in $\mathcal{V}$ are typed, where the type of a variable, such as reals, integers, etc., indicates the domain over which the variable ranges. Let $Z \subseteq \mathcal{V}$. A valuation of $Z$ is a mapping that associates to each variable of $Z$ a value in its domain. We write $Z$ for the set of valuations of $Z$. Often, valuations will be referred to as states.

A trajectory over $Z$ is a mapping $w : I \rightarrow Z$, where $I$ is a left-closed interval of $T$ with left endpoint equal to 0. With $\text{dom}(w)$ we denote the domain of $w$ and with $\text{traj}(Z)$ the collection of all trajectories over $Z$. If $w$ is a trajectory then $w.ltime$, the limit time of $w$, is the supremum of $\text{dom}(w)$. Similarly, define $w.fstate$, the first state of $w$, to be $w(0)$, and if $\text{dom}(w)$ is right-closed, define $w.lstate$, the last state of $w$, to be $w(w.ltime)$. A trajectory with domain $[0, 0]$ is called a point trajectory. If $s$ is a state then define $\varphi(s)$ to be the point trajectory that maps 0 to $s$.

For $w$ a trajectory and $t \in T^\geq 0$, we define $w \lhd t \triangleq w \upharpoonright [0, t]$ and $w \lhd \infty \triangleq w \upharpoonright [0, t)$. (Here $\upharpoonright$ denotes the restriction of a function to a subset of its domain.) Note that $w \lhd 0$ is not a trajectory. By convention, $w \lhd \infty = w \lhd \infty \triangleq w$. Similarly we define, for $w$ a trajectory and $I$ a left-closed interval with minimal element $l$, the restriction $w \upharpoonright I$ to be the function with domain $(I \cap \text{dom}(w)) - l$ given by $w \upharpoonright I(t) \triangleq w(t + l)$. Note that $w \upharpoonright I$ is a trajectory iff $I \in \text{dom}(w)$.

If $w$ is a trajectory over $Z$ and $Z' \subseteq Z$, then the projection $w \downarrow Z'$ is the trajectory over $Z'$ with domain $\text{dom}(w)$ defined by $w \downarrow Z'(t)(z) \triangleq w(t)(z)$. The projection operation is extended to sets of trajectories by pointwise extension. Also, if $w$ is a trajectory over $Z$ and $z \in Z$, then the projection $w \downarrow z$ is the function from $\text{dom}(w)$ to the domain of $z$ defined by $w \downarrow z(t) \triangleq w(t)(z)$.

If $w$ is a trajectory with a right-closed domain $I = [0, u]$, $w'$ is a trajectory with domain
We extend the concatenation operator to a countable sequence of trajectories: if \( w_i \) is a trajectory with domain \( I_i \), \( 1 \leq i < \infty \), where all \( I_i \) are right-closed, and if \( w_i . \text{fstate} = w_{i+1} . \text{fstate} \) for all \( i \), then we define the infinite concatenation, written \( w_1 \preceq w_2 \preceq w_3 \ldots \), to be the least function \( w \) such that \( w(t + \sum_{j<i} w_j . \text{fstate}) = w_i(t) \) for all \( t \in I_i \).

A trajectory \( w \) is closed if its domain is a (finite) closed interval and full if its domain equals \( T^{\geq 0} \). For \( W \) a set of trajectories, \( \text{Closed}(W) \) and \( \text{Full}(W) \) denote the subsets of closed and full trajectories in \( W \), respectively. Trajectory \( w \) is a prefix of trajectory \( w' \), notation \( w \preceq w' \), if either \( w = w' \) or \( w = w' \preceq w'' \), for some trajectory \( w'' \). With \( \text{Pref}(W) \) we denote the prefix-closure of \( W \): \( \text{Pref}(W) \triangleq \{ w \mid \exists w' \in W : w \preceq w' \} \). Set \( W \) is prefix closed if \( W = \text{Pref}(W) \). A trajectory in \( W \) is maximal if it is not a prefix of any other trajectory in \( W \). We write \( \text{Max}(W) \) for the subset of maximal trajectories in \( W \).

2.2 Hybrid I/O Automata
A hybrid I/O automaton (HIOA) \( A = (U, X, Y, \Sigma^i, \Sigma^i_n, \Sigma^o, \Theta, \mathcal{D}, W) \) consists of the following components:

- Three disjoint sets \( U, X \) and \( Y \) of variables, called input, internal and output variables, respectively.
  Variables in \( E \triangleq U \cup Y \) are called external, and variables in \( L \triangleq X \cup Y \) are called locally controlled. We write \( V \triangleq U \cup L \).

- Three disjoint sets \( \Sigma^i, \Sigma^i_n, \Sigma^o \) of input, internal and output actions, respectively.
  We assume that \( \Sigma^i \) contains a special element \( e \), the environment action, which represents the occurrence of a discrete transition outside the system that is unobservable, except (possibly) through its effect on the input variables. Actions in \( \Sigma^o \triangleq \Sigma^o_n \cup \Sigma^o \) are called external, and actions in \( \Sigma^o \triangleq \Sigma^o_n \cup \Sigma^o \) are called locally controlled. We write \( \Sigma \triangleq \Sigma^i \cup \Sigma^o \).

- A nonempty set \( \Theta \subseteq V \) of initial states satisfying

  \[ \text{Init (start states closed under change of input variables)} \]
  \[ \forall s, s' \in V : s \in \Theta \land s \models L = s' \models L \Rightarrow s' \in \Theta \]

- A set \( \mathcal{D} \subseteq V \times \Sigma \times V \) of discrete transitions satisfying

  \[ \text{D1 (input action enabling)} \]
  \[ \forall s \in V, a \in \Sigma^i \exists s' \in V : s \xrightarrow{a} s' \]

  \[ \text{D2 (environment action only affect inputs)} \]
  \[ \forall s, s' \in V : s \xrightarrow{e} s' \Rightarrow s \models L = s' \models L \]
A set \( \mathcal{W} \) of trajectories over \( V \) satisfying

\[
\begin{align*}
\text{T1 (existence of point trajectories)} & : \forall s \in V : \varrho(s) \in \mathcal{W} \\
\text{T2 (closure under subintervals)} & : \forall w \in \mathcal{W}, I \text{ left-closed, non-empty subinterval of } \text{dom}(w) : w \upharpoonright I \in \mathcal{W} \\
\text{T3 (completeness)} & : \forall t \in T^{\geq 0} : w \upharpoonright [0, t] \in \mathcal{W} \Rightarrow w \in \mathcal{W}
\end{align*}
\]

Axiom \textbf{Init} says that a system has no control over the initial values of its input variables: if one valuation is allowed then any other valuation is allowed also.

Axiom \textbf{D1} is a slight generalization of the input enabling condition of the (classical) I/O automaton model: it says that in each state each input action is enabled, including the environment action \( e \). The second axiom \textbf{D2} says that \( e \) cannot change locally controlled variables. Axiom \textbf{D3} expresses that, since input variables are not under control of the system, these variables may be changed in an arbitrary way after any discrete action. The three axioms together imply the converse of \textbf{D2}, i.e., if two states only differ in their input variables then there exists an \( e \) transition between them. Axioms \textbf{D1-3} play a crucial role in our study of parallel composition. In particular \textbf{D2} and \textbf{D3} are used to avoid cyclic constraints during the interaction of two systems.

Axioms \textbf{T1-3} state some natural conditions on the set of trajectories that we need to set up our theory: existence of point trajectories, closure under subintervals, and the fact that a full trajectory is in \( \mathcal{W} \) iff all its prefixes are in \( \mathcal{W} \).

\textit{Notation} \quad Let \( A \) be a HIOA as described above. If \( s \in V \) and \( l \in L \), then we write \( s \overset{a}{\rightarrow} l \) iff there exists an \( s' \in V \) such that \( s \overset{a}{\rightarrow} s' \) and \( s'[L] = l \). In the sequel, the components of a HIOA \( A \) will be denoted by \( V_A, U_A, \Sigma_A, \Theta_A \), etc. Sometimes, the components of a HIOA \( A \) will also be denoted by \( V_i, U_i, \Sigma_i, \Theta_i \), etc.

2.3 \textit{Hybrid Executions}

A \textit{hybrid execution fragment} of \( A \) is a finite or infinite alternating sequence \( \alpha = w_0 a_1 w_1 a_2 w_2 \cdots \), where:

1. Each \( w_i \) is a trajectory in \( \mathcal{W}_A \) and each \( a_i \) is an action in \( \Sigma_A \).
2. If \( \alpha \) is a finite sequence then it ends with a trajectory.
3. If \( w_i \) is not the last trajectory in \( \alpha \) then its domain is a right-closed interval and \( w_i \text{.lstate} \overset{a_{i+1}}{\rightarrow} w_{i+1} \text{.fstate} \).
An execution fragment records all the discrete changes that occur in the evolution of a system, plus the “continuous” state changes that take place in between. The third item says that the discrete actions in \( \alpha \) span between successive trajectories. We write \( h\text{-}\text{frag}(A) \) for the set of all hybrid execution fragments of \( A \).

If \( \alpha = w_0a_1w_1a_2w_2 \cdots \) is a hybrid execution fragment then we define the limit time of \( \alpha \), notation \( \alpha.ltime \), to be \( \sum_i w_i.ltime \). Further, we define the first state of \( \alpha \), \( \alpha.fstate \), to be \( w_0.fstate \).

We distinguish several sorts of hybrid execution fragments. A hybrid execution fragment \( \alpha \) is defined to be

- an execution if the first state of \( \alpha \) is an initial state,
- finite if \( \alpha \) is a finite sequence and the domain of its final trajectory is a right-closed interval,
- admissible if \( \alpha.ltime = \infty \),
- Zeno if \( \alpha \) is neither finite nor admissible, and
- a sentence if \( \alpha \) is a finite execution that ends with a point trajectory.

If \( \alpha = w_0a_1w_1 \cdots a_nw_n \) is a finite hybrid execution fragment then we define the last state of \( \alpha \), notation \( \alpha.lstate \), to be \( w_n.lstate \). A state of \( A \) is defined to be reachable if it is the last state of some finite hybrid execution of \( A \).

A finite hybrid execution fragment \( \alpha = w_0a_1w_1a_2w_2 \cdots a_nw_n \) and a hybrid execution fragment \( \alpha' = w'_0a'_1w'_1a'_2w'_2 \cdots \) of \( A \) can be concatenated if \( w_n \sim w'_0 \) is defined and a trajectory of \( A \). In this case, the concatenation \( \alpha \prec \alpha' \) is the hybrid execution fragment defined by

\[
\alpha \prec \alpha' \triangleq w_0a_1w_1a_2w_2 \cdots a_n(w_n \sim w'_0)a'_1w'_1a'_2w'_2 \cdots
\]

2.4 Hybrid Traces
Suppose \( \alpha = w_0a_1w_1a_2w_2 \cdots \) is a hybrid execution fragment of \( A \). In order to define the hybrid trace of \( \alpha \), let

\[
\gamma = (w_0 \downarrow E_A)\text{vis}(a_1)(w_1 \downarrow E_A)\text{vis}(a_2)(w_2 \downarrow E_A) \cdots,
\]

where, for \( a \) an action, \( \text{vis}(a) \) is defined equal to \( \tau \) if \( a \) is an internal action or \( e \), and equal to \( a \) otherwise. Here \( \tau \) is a special symbol which, as in the theory of process algebra, plays the role of the ‘generic’ invisible action. An occurrence of \( \tau \) in \( \gamma \) is called inert if the final state of the trajectory that precedes the \( \tau \) equals the first state of the trajectory that follows it (after hiding of the internal variables). The hybrid trace of \( \alpha \), written \( h\text{trace}(\alpha) \), is defined to be the sequence obtained from \( \gamma \) by removing all inert \( \tau \)’s and concatenating the surrounding trajectories.

The hybrid traces of \( A \) are the hybrid traces that arise from all the finite and admissible hybrid executions of \( A \). We write \( h\text{-}\text{traces}(A) \) for the set of hybrid traces of \( A \).
HIOA's $A_1$ and $A_2$ are *comparable* if they have the same external interface, i.e., $U_1 = U_2$, $Y_1 = Y_2$, $X_i^\text{in} = X_j^\text{in}$ and $X_i^\text{out} = X_j^\text{out}$. If $A_1$ and $A_2$ are comparable then $A_1 \leq A_2$ is defined to mean that the hybrid traces of $A_1$ are included in those of $A_2$: $A_1 \leq A_2 \triangleq \text{h-traces}(A_1) \subseteq \text{h-traces}(A_2)$.

### 3. Simulation Relations

Let $A$ and $B$ be comparable HIOA's. A *simulation* from $A$ to $B$ is a relation $R \subseteq V_A \times V_B$ satisfying the following conditions, for all states $r$ and $s$ of $A$ and $B$, respectively:

1. If $r \in \Theta_A$ then there exists $s \in \Theta_B$ such that $r R s$.
2. If $r \xrightarrow{a} A r'$ and $r R s$ then $B$ has a finite execution fragment $\alpha$ with $s = \alpha.fstate$, $\text{htrace}(\varphi(r) a \varphi(r')) = \text{htrace}(\alpha)$ and $r' R \alpha.lstate$.
3. If $r R s$ and $w$ is a closed trajectory of $A$ with $r = w.fstate$ then $B$ has a finite execution fragment $\alpha$ with $s = \alpha.fstate$, $\text{htrace}(w) = \text{htrace}(\alpha)$ and $w.lstate R \alpha.lstate$.

Note that by Condition 3 and the existence of point trajectories (axiom T1), $r R s$ implies that $r [E_A = s [E_B$.

**Theorem 3.1** If $A$ and $B$ are comparable HIOA's and there is a simulation from $A$ to $B$, then $A \leq B$.

### 4. Parallel Composition and Hiding

We say that HIOA's $A_1$ and $A_2$ are *compatible* if, for $i \neq j$,

$$X_i \cap V_j = Y_i \cap Y_j = \Sigma_i^\text{int} \cap \Sigma_j = \Sigma_i^\text{out} \cap \Sigma_j = \emptyset.$$

If $A_1$ and $A_2$ are compatible then their composition $A_1 \parallel A_2$ is defined to be the tuple $A = (U, X, Y, X^\text{in}_1, X^\text{in}_2, \Theta, D, W)$ given by

- $U = (U_1 \cup U_2) - (Y_1 \cup Y_2)$, $X = X_1 \cup X_2$, $Y = Y_1 \cup Y_2$
- $X_1^\text{in} = (X_1^\text{in} \cup X_2^\text{in}) - (X_1^\text{out} \cup X_2^\text{out})$, $X_2^\text{in} = X_2^\text{in} \cup X_1^\text{in}$, $Y^\text{out} = X_1^\text{out} \cup X_2^\text{out}$
- $\Theta = \{s \in V \mid s[V_1 \in \Theta_1 \land s[V_2 \in \Theta_2\}$
- Define, for $i \in \{1, 2\}$, projection function $\pi_i : \Sigma \rightarrow \Sigma_i$ by $\pi_i(a) = a$ if $a \in \Sigma_i$ and $\pi_i(a) = e$ otherwise. Then $D$ is the subset of $V \times \Sigma \times V$ given by
  $$\{ (s, a, s') \in D \Leftrightarrow s[V_1 \pi_1[a] \land s[V_1 \land s[V_2 \pi_2[a]] s' [V_2\}$
- $\mathcal{W}$ is the set of trajectories over $V$ given by
  $$w \in \mathcal{W} \Leftrightarrow w \downarrow V_1 \in W_1 \land w \downarrow V_2 \in W_2$$

**Proposition 4.1** $A_1 \parallel A_2$ is a HIOA.
Theorem 4.2 Suppose $A_1, A_2$ and $B$ are HIOA's with $A_1 \leq A_2$, and each of $A_1$ and $A_2$ is compatible with $B$. Then $A_1 \parallel B \leq A_2 \parallel B$.

Two natural hiding operations can be defined on any HIOA $A$:

1. If $S \subseteq \Sigma_A^{\text{out}}$, then $\text{ActHide}(S, A)$ is the HIOA $B$ that is equal to $A$ except that $\Sigma_B^{\text{out}} = \Sigma_A^{\text{out}} - S$ and $\Sigma_B^{\text{int}} = \Sigma_A^{\text{int}} \cup S$.

2. If $Z \subseteq Y_A$, then $\text{VarHide}(Z, A)$ is the HIOA $B$ that is equal to $A$ except that $Y_B = Y_A - Z$ and $X_B = X_A \cup Z$.

Theorem 4.3 Suppose $A$ and $B$ are HIOA's with $A \leq B$, and let $S \subseteq \Sigma_A^{\text{out}}$ and $Z \subseteq Y_A$. Then $\text{ActHide}(S, A) \leq \text{ActHide}(S, B)$ and $\text{VarHide}(Z, A) \leq \text{VarHide}(Z, B)$.

5. Receptiveness

We call a HIOA feasible if any finite execution can be extended to an admissible execution. The main significance of feasibility is to guarantee that a HIOA is meaningful in the sense that it cannot block time. Unfortunately feasibility is not preserved by parallel composition, and thus we need to impose additional restrictions on a HIOA so that the feasibility property is guaranteed to be preserved by parallel composition. Our ideal objective would be to find the weakest restrictions that need to be imposed; here we just propose some restrictions, although we have not proved that they are the weakest. Below we define a notion of receptiveness and prove that it is preserved by composition under some reasonable assumptions.

5.1 I/O Behaviors

The concept of an I/O behavior plays an important role in the definition of receptiveness. Intuitively, an I/O behavior is a set of trajectories that arise from an HIOA after choosing initial values for the local variables and resolving all internal nondeterminism.

We assume, for each variable $v \in \mathcal{V}$, a dynamic type $\mathcal{F}_v$, which is a nonempty collection of functions from $T$ to the domain of $v$. We require the sets $\mathcal{F}_v$ to be time-invariant: for each $f \in \mathcal{F}_v$ and each $t \in T$, also $\Delta f \in \mathcal{F}_v$, where $\Delta f$ is the function from $T$ to the domain of $v$ given by $\Delta f(t') = f(t' + t)$. Intuitively, the dynamic type $\mathcal{F}_v$ gives the collection of allowed trajectories for $v$. For instance, if $T = \mathbb{R}$ and $v$ has domain $\mathbb{R}$, then $\mathcal{F}_v$ will be the set of all continuous or smooth functions, or the set of all measurable locally essentially bounded functions [19]. If $v$ is a "discrete" variable (in the sense of [15]), then $\mathcal{F}_v$ is the set of all the constant functions. If $Z \subseteq \mathcal{V}$ then we write $\mathcal{F} \text{-trajs}(Z)$ for the set of trajectories $w$ over $Z$ with the property that for all $z \in Z$, $w \downarrow z \in \mathcal{F}_z[\text{dom}(w)]$.

An I/O behavior is a triple $P = (U, Y, B)$, where

- $U$ is a set of typed input variables;
- $Y$ is a set of typed output variables with $U \cap Y = \emptyset$; we write $V \overset{\Delta}{=} U \cup Y$;
- $B \subseteq \mathcal{F} \text{-trajs}(V)$ is a prefix closed set of trajectories satisfying
B1 (functional dependence of outputs from inputs)
For all $w, w' \in \mathcal{B}$ and for all $t \in \text{dom}(w) \cap \text{dom}(w')$,
\[
(w \downarrow t) \downarrow U = (w' \downarrow t) \downarrow U \Rightarrow w(t) = w'(t)
\]

B2 (freedom of inputs)
\[\forall w \in \text{Full}(\mathcal{F}\text{-tmajs}(U)) \exists w' \in \text{Max}(B) : w' \downarrow U \leq w\]

B3 (nonZenoness)
\[
\text{Max}(B) \subseteq \text{Closed}(B) \cup \text{Full}(B)
\]

Axiom B1 says that the output at time $t$ is fully determined by the inputs at times up to, but not including, $t$. Roughly speaking, axiom B2 expresses that the input is a signal that is imposed by the environment and over which the system has no control. However, in a hybrid world a continuous phase of a system can be interrupted at any time by the occurrence of a discrete transition. A system may, for instance, perform a locally controlled discrete action as soon as the input reaches a threshold value. Therefore, axiom B2 only requires that for each full input signal there exists a maximal trajectory that, when projected on its input, forms a prefix of this input signal. Axiom B3 states that each maximal trajectory is either closed or full. Together, B2 and B3 imply that in an I/O behavior each input signal is accepted up to and including some finite time $t$ or up to $\infty$. Note that for any I/O behavior $P$ there is an output state $s \in \mathcal{Y}$ such that all trajectories $w$ in $\mathcal{B}$ begin with $s$, i.e., $w(0) | \mathcal{Y} = s$.

Our I/O behaviors can be viewed as a special case of the I/O behaviors of Sontag [19]. Sontag defines I/O behaviors in terms of a response map from input signals up to time $t$ to the output at time $t$, but this presentation is equivalent to our definition in terms of trajectories over both inputs and outputs. Technically, we found it a bit easier to use trajectories in this paper. In [19], no assumptions are made about possible input signals and the length of maximal trajectories (our axioms B2 and B3). However, [19] singles out the so-called $\mathcal{V}$-complete I/O behaviors, which are I/O behaviors that accept any input of type $\mathcal{V}$.

In the sequel, the components of an I/O behavior $P$ will be denoted by $V_P, U_P, Y_P$, and $B_P$. Also, if no confusion can arise, the components of an I/O behavior $P_i$ will be denoted by $V_i, U_i, Y_i$, and $B_i$, etc.

Two I/O behaviors $P_1$ and $P_2$ are compatible if $Y_1 \cap Y_2 = \emptyset$. In this case, we define the composition $P_1 \parallel P_2$ to be the structure $P = (U, Y, \mathcal{B})$ where

- $U = (U_1 \cup U_2) - (Y_1 \cup Y_2)$,
- $Y = Y_1 \cup Y_2$, and
- $\mathcal{B} \subseteq \mathcal{F}\text{-tmajs}(U \cup Y)$ is given by $w \in \mathcal{B} \Rightarrow w \downarrow V_1 \in \mathcal{B}_1 \land w \downarrow V_2 \in \mathcal{B}_2$.

In general, the composition of two compatible I/O behaviors need not be an I/O behavior since there may be "too many solutions":

**Example 5.1** Suppose $T = \mathbb{R}$. For $u, y$ variables whose dynamic type is the set of functions from $\mathbb{R}$ to $\mathbb{R}$ that have left-hand limits, define $\text{Copy}(u, y)$ to be the I/O behavior that, for $t > 0$, copies input $u$ to output $y$, and with the initial value of $y$ set to 0. Then the composition of
Copy\((u, y)\) and Copy\((y, u)\) has no input variables and therefore just one full input trajectory is allowed. However, there is more than one output trajectory and thus the composition does not satisfy axiom B1.

It may also occur that the composition of two compatible I/O behaviors yields an I/O behavior, even though there exists no “solution” in the sense that maximal trajectories can be merged. This motivates the following definition.

Two compatible I/O behaviors \(P_1\) and \(P_2\) are strongly compatible if \(P = P_1 \parallel P_2\) is an I/O behavior and, for each trajectory \(w\) of \(P\),

\[
w \in \text{Max}(B_P) \iff (w \downarrow V_1 \in \text{Max}(B_1) \lor w \downarrow V_2 \in \text{Max}(B_2)).
\]

**Example 5.2** Suppose \(T = R\). For \(u, y\) variables whose dynamic type is the set of functions from \(R\) to \(R\) that have left-hand limits, define Add1\((u, y)\) to be the I/O behavior whose output \(y\) is, for \(t > 0\), equal to the input \(u\) incremented by 1, and with the initial value of \(y\) set to 0. Then the I/O behaviors Add1\((u, y)\) and Add1\((y, u)\) are compatible but not strongly compatible, even though their composition is an I/O behavior.

Let \(A\) be a HIOA and let \(l \in L_A\) be a valuation of the local variables of \(A\). A nonempty set \(W\) of trajectories of \(A\) is called an \(l\)-process (or process) of \(A\) if \((U_A, L_A, W)\) is an I/O behavior and, for all \(w \in W\), \(w(0)[L_A = l\), i.e., the initial states of all trajectories in \(W\) agree with \(l\).

Two compatible HIOA’s \(A_1\) and \(A_2\) are strongly compatible if for each reachable state \(s\) of \(A_1\|A_2\), for each \((s[L_1]\)-process \(W_1\) of \(A_1\), and for each \((s[L_2]\)-process \(W_2\) of \(A_2\), the I/O behaviors \((U_1, L_1, W_1)\) and \((U_2, L_2, W_2)\) are strongly compatible.

### 5.2 Games and Strategies

Intuitively, a system is receptive if time can advance to infinity independently of the input provided by its environment, or equivalently, if it does not constrain its environment. In [6, 1, 7] various notions of receptivity have been defined in terms of games. Below, we extend these ideas to the setting of HIOA’s. The interaction between a system and its environment is represented as a two person game in which the goal of the system is to construct an admissible execution, and the goal of the environment is to prevent this. The system is receptive if it has a strategy by which it can always win the game, irrespective of the behavior of the environment.

Formally, a strategy \(\rho\) for \(A\) is a function that specifies, for each sentence \(\alpha\) of \(A\) with \(l = \alpha.lstate\mid L_A\),

1. an \(l\)-process \(W^\alpha\) of \(A\),
2. a function \(g^\alpha : \text{Closed}(W^\alpha) \times \Sigma^\alpha_A \rightarrow L_A\) satisfying
   \[
g^\alpha(w, a) = l \Rightarrow w.lstate \xrightarrow{\alpha} A l.
   \]
3. a function \(f^\alpha : \text{Closed}(\text{Max}(W^\alpha)) \rightarrow (\Sigma^\alpha_A \times L_A)\) satisfying
   \[
f^\alpha(w) = (a, l) \Rightarrow w.lstate \xrightarrow{\alpha} A l,
   \]
At the beginning and immediately after each discrete transition, a strategy produces a process \( W \) that starts in the current local state. By doing this, a strategy resolves all nondeterminism for the next continuous phase. Typically, choosing a process amounts to fixing the trajectories for certain internal variables that represent disturbances, and deciding at which time the next locally controlled action will be performed. Once a process has been selected, the input signal fully determines the next trajectory in the execution of the system. Since at any point the environment may produce a discrete input action, a strategy also specifies, through the function \( g \), what will be the next local state after such an action. The values of the input variables after a discrete step are determined by the environment. Through the function \( f \), a strategy specifies, for each maximal and closed trajectory of the selected process, which locally controlled step will be performed at the end of this trajectory.

In the game between the environment and the system the behavior of the environment is represented by an environment sequence. This is an infinite alternating sequence

\[
I = w_1 a_1 b_1 w_2 a_2 b_2 \ldots
\]

of closed or full trajectories \( w_i \in \mathcal{F}\text{-trajs}(U_A) \), actions \( a_i \in \Sigma^\text{in}_A \), and booleans \( b_i \in \{T,F\} \).

In the \( i \)-th move of the game, the environment produces input signal \( w_i \). If \( w_i \) is finite then the environment produces discrete action \( a_i \) right after signal \( w_i \). The boolean \( b_i \) serves to break ties in case the environment and the system both want to perform a discrete action at the same time: if \( b_i = T \) then the environment is allowed to make a move and otherwise the system may perform an action. As in [7], our game starts after a finite execution \( \alpha \). The outcome of the game is described formally in the following definition.

Let \( A \) be a HIOA, \( \rho \) a strategy for \( A \), \( I \) an environment sequence for \( A \) (with \( \rho \) and \( I \) as defined above), and let \( \alpha \) be a finite hybrid execution of \( A \). We define the outcome \( O_{\rho,I}(\alpha) \) as the limit of the sequence \( (\alpha_i)_{i \geq 0} \) of hybrid executions that is constructed inductively below. Each \( \alpha_i \) is either a sentence or admissible.

Let \( l = \text{a.state}[L_A] \). Then \( \alpha_0 \triangleq \alpha \in \varphi(w_1(0) \cup l) \).

Here we extend \( \alpha \) in a trivial way to a sentence in order to get into a situation where strategy \( \rho \) can be applied in combination with environment sequence \( I \). In the definition, \( \cup \) is the operation that takes the union of two functions, each viewed as a set of pairs. The first argument of \( \cup \) yields the values for the input variables and the second argument the values for the locally controlled variables.

For \( i > 0 \), define \( \alpha_i \) in terms of \( \alpha_{i-1} \) as follows.

If \( \alpha_{i-1} \) is admissible then \( \alpha_i \triangleq \alpha_{i-1} \).

Otherwise, \( \alpha_{i-1} \) is a sentence. Pick any full trajectory \( w^+_i \in \mathcal{F}\text{-trajs}(U_A) \) with \( w_i \leq w^+_i \). Then by axiom B2 there is a maximal execution \( w'_i \in W^\alpha \) with \( w'_i \downarrow U_A \leq w^+_i \). By axiom B1, \( w'_i \) is uniquely determined by the choice of \( w^+_i \). Let \( t = w_i.\text{itime} \) and \( t' = w'_i.\text{itime} \). We distinguish between three cases:

1. If \( t = t' = \infty \) then

\[
\alpha_i \triangleq \alpha_{i-1} \circ w'_i.
\]
5. Receptiveness

This is the case where both the system and the environment have decided not to perform any discrete action.

2. If \( t < t' \) or \( t = t' < \infty \land b_i = T \), then

\[
\alpha_i \overset{b}{=} \alpha_{i-1} \cup (w'_i \triangleq t) a_i \phi(w_{i+1}(0) \cup g^\alpha(w'_i \triangleq t, a_i)).
\]

This is the case where, after an initial fragment of \( w'_i \), the environment produces an input action \( a_i \). The resulting state after this action is obtained by taking the union of the first state of the next input trajectory and the local state that is specified by the \( g \)-part of the strategy.

3. If \( t' < t \) or \( t = t' < \infty \land b_i = F \) and if we let \( f^\alpha(w'_i) = (a'_i, l_i) \), then

\[
\alpha_i \overset{b}{=} \alpha_{i-1} \cup (w'_i a'_i \phi(w_{i+1}(0) \cup l_i)).
\]

This is the case where, after \( w'_i \) has been completed, the system performs a locally controlled step as specified by the \( f \)-part of the strategy.

Note that the definition of \( \alpha_i \) does not depend on the choice of \( w_i^+ \) since by axiom B1 the prefix \( w'_i \triangleq t \) of \( w'_i \) that is used in the construction is determined uniquely by the fixed prefix \( w'_i \) of \( w_i^+ \).

**Proposition 5.3** \( O_{\rho, I}(\alpha) \) is a Zeno or admissible hybrid execution of \( \mathcal{A} \).

A hybrid execution \( \alpha \) of a HIOA \( \mathcal{A} \) is Zeno-tolerant if it is Zeno, contains infinitely many input actions and only finitely many locally controlled actions. A strategy \( \rho \) for \( \mathcal{A} \) is Zeno-tolerant if for each environment sequence \( I \) and for each finite execution \( \alpha \), \( O_{\rho, I}(\alpha) \) is either admissible or Zeno-tolerant. We call \( \mathcal{A} \) receptive iff there exists a Zeno-tolerant strategy for \( \mathcal{A} \). Note that each receptive HIOA is trivially feasible.

We now come to the main result of this paper.

**Theorem 5.4** Suppose \( \mathcal{A}_1 \) and \( \mathcal{A}_2 \) are strongly compatible, receptive HIOA's. Then \( \mathcal{A}_1 \| \mathcal{A}_2 \) is receptive.

The corresponding result for the hiding operations is much easier to prove:

**Theorem 5.5** Suppose \( \mathcal{A} \) is a receptive HIOA, and let \( S \subseteq \Sigma_{A}^{out} \) and \( Z \subseteq Y_A \). Then \( \text{ActHide}(S, \mathcal{A}) \) and \( \text{VarHide}(Z, \mathcal{A}) \) are receptive.

5.3 Strong Compatibility vs. Compatibility

In order to apply Theorem 5.4, one has to establish that the HIOA's \( \mathcal{A}_1 \) and \( \mathcal{A}_2 \) are strongly compatible. From control theory it is well-known that this is a difficult problem in general. However, it is possible to identify certain classes of I/O behaviors for which strong compatibility reduces to compatibility. This means that for all processes of \( \mathcal{A}_1 \) and \( \mathcal{A}_2 \) in such a
5. Receptiveness

class, the condition of strong compatibility in Theorem 5.4, which in general is hard to check, reduces to the syntactic condition of compatibility.

A first example can be obtained by considering what we call *autistic* I/O behaviors. These are I/O behaviors that accept any input but produce an output that is totally unrelated to this input. Formally, an I/O behavior is called *autistic* if it satisfies the axiom

\[ \forall w, w' \in B : \text{dom}(w) = \text{dom}(w') \Rightarrow w \downarrow Y = w' \downarrow Y \]

It is easy to verify that two autistic processes are strongly compatible if and only if they are compatible. From the perspective of classical control theory autistic processes are definitely of no interest: why have an input if it is not used at all? In a hybrid setting, however, an automaton that does not process its input in a continuous manner can still monitor this input and perform a discrete transition when some threshold is reached. In *linear hybrid automata* [3, 2], for instance, there is no continuous processing of inputs and all underlying processes are autistic.

Less trivial examples of classes of I/O behaviors for which strong compatibility reduces to compatibility can be found in the literature on control theory [19]. In control theory it is common to express the continuous behavior of a system by means of differential equations; thus, to be sure that a system is well described, the differential equations need to admit a unique solution for each possible starting condition of the system. A typical approach is to describe a system through differential equations of the form

\[
E \triangleq \begin{cases} 
\dot{x} &= f(x, u) \\
y &= g(x) 
\end{cases}
\]

where \(u, y,\) and \(x\) are the input, output, and internal vectors of variables, respectively. It is known from calculus that if \(f\) is globally Lipschitz and \(u\) is \(C^1\), then for each fixed starting condition \(x(0) = x_0\) there is a unique solution to the equations of \(E\), defined on a maximal neighborhood of 0, such that \(x(0) = x_0\). Suppose that the dynamic type of each input variable is the set of all \(C^1\) functions. Consider the set \(W\) of all the solutions to \(E\) for each possible choice of \(x_0\) and of \(u(t)\), and let \((U, X \cup Y, W')\) be any I/O behavior whose trajectories are prefixes of trajectories in \(W\). We say that \((U, X \cup Y, W')\) is an I/O behavior of \(E\).

Consider now two systems, described by equations \(E_1\) and \(E_2\) with the same form as \(E\), and suppose there are no common locally controlled variables in \(E_1\) and \(E_2\). The interaction between \(E_1\) and \(E_2\) can be described by a new set of equations \(E_3\) obtained by considering together the equations of \(E_1\) and \(E_2\). If also the \(g\) functions of \(E_1\) and \(E_2\) are globally Lipschitz, then it is easy to show that \(E_3\) can be represented in the same form as \(E\) where \(f\) and \(g\) are globally Lipschitz. Furthermore, let \(P_1\) and \(P_2\) be any two I/O behaviors of \(E_1\) and \(E_2\), respectively. Then it is the case that \(P_1\) and \(P_2\) are strongly compatible and that \(P_3\) is an I/O behavior of \(E_3\).

Therefore, if we choose the dynamic type of each variable to be the set of all \(C^1\) functions, then strong compatibility reduces to compatibility for I/O behaviors of systems of equations \(E\), where \(f\) and \(g\) are globally Lipschitz. In general, any choice of conditions on \(f\) and \(u\) that guarantee local existence of unique solutions and that are preserved by interaction between
systems can be used as a basis to define a class of processes for which strong compatibility reduces to compatibility.

Acknowledgment We thank Jan van Schuppen for constructive criticism.

References


