Hybrid I/O Automata

(extended abstract)

Nancy Lynch[1]* Roberto Segala[2] Frits Vaandrager[3]** H.B. Weinberg[1]***

[1] MIT Laboratory for Computer Science
Cambridge, MA 02139, USA
`{lynch,hbw}@theory.lcs.mit.edu`
[2] Dipartimento di Matematica, Universita' di Bologna
Piazza di Porta San Donato 5, 40127 Bologna, Italy
`segala@cs.unibo.it`
[3] CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
`fritsv@cwi.nl`

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.

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

* Supported by NSF Grant 9225124-CCR, U.S. Department of Transportation Contract DTRS95G-0001-YR.8, AFOSR-ONR Contract F49620-94-1-0199, and ARPA Contracts F19628-95-C-0118 and N00014-92-J-4033.

** Current affiliation: Computing Science Institute, University of Nijmegen, P.O. Box 9010, 9500 GL Nijmegen, The Netherlands, `Frits.Vaandrager@cs.kun.nl`.

*** Research partially supported by a National Science Foundation Graduate Fellowship.

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 iff 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 partioned 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 $\dot{y} \in [0,2]$, whereas B only allows trajectories with $\dot{y} = 1$.

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, nonZeno 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 $(\mathsf{R}, +)$, the real numbers with addition. Usually, $T = \mathsf{R}$ or 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 $I + t \triangleq \{t' + t \mid t' \in I\}$.

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 $\mathbf{Z}$ for the set of valuations of Z. Often, valuations will be referred to as *states*.

A *trajectory* over Z is a mapping $w : I \longrightarrow \mathbf{Z}$, where I is a left-closed interval of T with left endpoint equal to 0. With $dom(w)$ we denote the domain of w and with $trajs(Z)$ the collection of all trajectories over Z. If w is a trajectory then $w.ltime$, the *limit time* of w, is the supremum of $dom(w)$. Similarly, define $w.fstate$, the *first state* of w, to be $w(0)$, and if $dom(w)$ 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 $\wp(s)$ to be the point trajectory that maps 0 to s.

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

If w is a trajectory over Z and $Z' \subseteq Z$, then the *projection* $w \downarrow Z'$ is the trajectory over Z' with domain $dom(w)$ 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 $dom(w)$ 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 I', and if $w.lstate = w'.fstate$, then we define the *concatenation* $w \frown w'$ to be the trajectory with domain $I \cup (I' + u)$ given by

$$w \frown w'\ (t) \triangleq \begin{cases} w(t) & \text{if } t \in I, \\ w'(t \Leftrightarrow u) & \text{otherwise.} \end{cases}$$

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 $w_i.lstate = w_{i+1}.fstate$ for all i, then we define the *infinite concatenation*, written $w_1 \frown w_2 \frown w_3 \ldots$, to be the least function w such that $w(t + \sum_{j<i} w_j.ltime) = w_i(t)$ 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, $Closed(W)$ and $Full(W)$ denote the subsets of closed and full trajectories in W, respectively. Trajectory w is a *prefix* of trajectory w', notation $w \leq w'$, if either $w = w'$ or $w' = w \frown w''$, for some trajectory w''. With $Pref(W)$ we denote the *prefix-closure* of W: $Pref(W) \triangleq \{w \mid \exists w' \in W : w \leq w'\}$. Set W is *prefix closed* if $W = Pref(W)$. A trajectory in W is *maximal* if it is not a prefix of any other trajectory in W. We write $Max(W)$ for the subset of maximal trajectories in W.

2.2 Hybrid I/O Automata

A *hybrid I/O automaton (HIOA)* $A = (U, X, Y, \Sigma^{in}, \Sigma^{int}, \Sigma^{out}, \Theta, \mathcal{D}, \mathcal{W})$ 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 Σ^{in}, Σ^{int}, Σ^{out} of *input*, *internal* and *output actions*, respectively.
  We assume that Σ^{in} 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 $\Sigma^{ext} \triangleq \Sigma^{in} \cup \Sigma^{out}$ are called *external*, and actions in $\Sigma^{loc} \triangleq \Sigma^{int} \cup \Sigma^{out}$ are called *locally controlled*. We write $\Sigma \triangleq \Sigma^{in} \cup \Sigma^{loc}$.
- A nonempty set $\Theta \subseteq \mathbf{V}$ of *initial states* satisfying
  Init (start states closed under change of input variables)
  $\forall s, s' \in \mathbf{V} : s \in \Theta \wedge s \lceil L = s' \lceil L \quad \Rightarrow \quad s' \in \Theta$
- A set $\mathcal{D} \subseteq \mathbf{V} \times \Sigma \times \mathbf{V}$ of *discrete transitions* satisfying
  D1 (input action enabling)
  $\forall s \in \mathbf{V}, a \in \Sigma^{in}\ \exists s' \in \mathbf{V} : s \xrightarrow{a} s'$
  D2 (environment action only affect inputs)
  $\forall s, s' \in \mathbf{V} : s \xrightarrow{e} s' \quad \Rightarrow \quad s \lceil L = s' \lceil L$

D3 (input variable change enabling)
$\forall s, s', s'' \in \mathbf{V}, a \in \Sigma : s \overset{a}{\Leftrightarrow} s' \wedge s' \lceil L = s'' \lceil L \quad \Rightarrow \quad s \overset{a}{\Leftrightarrow} s''$
Here we used $s \overset{a}{\Leftrightarrow} s'$ as shorthand for $(s, a, s') \in \mathcal{D}$.

– A set $\mathcal{W}$ of trajectories over V satisfying
T1 (existence of point trajectories)
$\forall s \in \mathbf{V} : \wp(s) \in \mathcal{W}$
T2 (closure under subintervals)
$\forall w \in \mathcal{W}, I$ left-closed, non-empty subinterval of $dom(w)$: $w \dagger I \in \mathcal{W}$
T3 (completeness)
$(\forall t \in T^{\geq 0} : w \dagger [0, t] \in \mathcal{W}) \Rightarrow w \in \mathcal{W}$

Axiom **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 **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 **D2** says that e cannot change locally controlled variables. Axiom **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 **D2**, i.e., if two states only differ in their input variables then there exists an e transition between them. Axioms **D1-3** play a crucial role in our study of parallel composition. In particular **D2** and **D3** are used to avoid cyclic constraints during the interaction of two systems.

Axioms **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 $\mathcal{W}$.

Notation Let A be a HIOA as described above. If $s \in \mathbf{V}$ and $l \in \mathbf{L}$, then we write $s \overset{a}{\Leftrightarrow} l$ iff there exists an $s' \in \mathbf{V}$ such that $s \overset{a}{\Leftrightarrow} s'$ and $s' \lceil L = l$. In the sequel, the components of a HIOA A will be denoted by V_A, U_A, Σ_A, Θ_A, etc. Sometimes, the components of a HIOA A_i will also be denoted by V_i, U_i, Σ_i, Θ_i, etc.

2.3 Hybrid Executions

A *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 Σ_A.
2. If α is a finite sequence then it ends with a trajectory.
3. If w_i is not the last trajectory in α then its domain is a right-closed interval and $w_i.lstate \overset{a_{i+1}}{\Leftrightarrow}_A w_{i+1}.fstate$.

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 α span between successive trajectories. We write *h-frag*(A) for the set of all hybrid execution fragments of A.

If $\alpha = w_0 a_1 w_1 a_2 w_2 \cdots$ is a hybrid execution fragment then we define the *limit time* of α, notation $\alpha.\mathit{ltime}$, to be $\sum_i w_i.\mathit{ltime}$. Further, we define the *first state* of α, $\alpha.\mathit{fstate}$, to be $w_0.\mathit{fstate}$.

We distinguish several sorts of hybrid execution fragments. A hybrid execution fragment α is defined to be

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

If $\alpha = w_0 a_1 w_1 \cdots a_n w_n$ is a finite hybrid execution fragment then we define the *last state* of α, notation $\alpha.\mathit{lstate}$, to be $w_n.\mathit{lstate}$. 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 $\alpha = w_0 a_1 w_1 a_2 w_2 \cdots a_n w_n$ and a hybrid execution fragment $\alpha' = w'_0 a'_1 w'_1 a'_2 w'_2 \cdots$ of A can be *concatenated* if $w_n \frown w'_0$ is defined and a trajectory of A. In this case, the *concatenation* $\alpha \frown \alpha'$ is the hybrid execution fragment defined by

$$\alpha \frown \alpha' \triangleq w_0 a_1 w_1 a_2 w_2 \cdots a_n (w_n \frown w'_0) a'_1 w'_1 a'_2 w'_2 \cdots$$

2.4 Hybrid Traces

Suppose $\alpha = w_0 a_1 w_1 a_2 w_2 \cdots$ is a hybrid execution fragment of A. In order to define the *hybrid trace* of α, let

$$\gamma = (w_0 \downarrow E_A)\mathit{vis}(a_1)(w_1 \downarrow E_A)\mathit{vis}(a_2)(w_2 \downarrow E_A)\cdots,$$

where, for a an action, $\mathit{vis}(a)$ is defined equal to τ if a is an internal action or e, and equal to a otherwise. Here τ is a special symbol which, as in the theory of process algebra, plays the role of the 'generic' invisible action. An occurrence of τ in γ is called *inert* if the final state of the trajectory that precedes the τ equals the first state of the trajectory that follows it (after hiding of the internal variables). The *hybrid trace* of α, written $\mathit{htrace}(\alpha)$, is defined to be the sequence obtained from γ by removing all inert τ'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 $\mathit{h\text{-}traces}(A)$ 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., $U_1 = U_2$, $Y_1 = Y_2$, $\Sigma_1^{in} = \Sigma_2^{in}$ and $\Sigma_1^{out} = \Sigma_2^{out}$. 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: $A_1 \leq A_2 \triangleq \mathit{h\text{-}traces}(A_1) \subseteq \mathit{h\text{-}traces}(A_2)$.

3 Simulation Relations

Let A and B be comparable HIOA's. A *simulation* from A to B is a relation $R \subseteq \mathbf{V}_A \times \mathbf{V}_B$ 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 $r \overset{a}{\Leftrightarrow\!\!\rightarrow}_A r'$ and $r\ R\ s$ then B has a finite execution fragment α with $s = \alpha.\mathit{fstate}$, $\mathit{htrace}(\wp(r)\ a\ \wp(r')) = \mathit{htrace}(\alpha)$ and $r'\ R\ \alpha.\mathit{lstate}$.
3. If $r\ R\ s$ and w is a closed trajectory of A with $r = w.\mathit{fstate}$ then B has a finite execution fragment α with $s = \alpha.\mathit{fstate}$, $\mathit{htrace}(w) = \mathit{htrace}(\alpha)$ and $w.\mathit{lstate}\ R\ \alpha.\mathit{lstate}$.

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

Theorem 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 $i \neq j$,

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

If A_1 and A_2 are compatible then their *composition* $A_1 \| A_2$ is defined to be the tuple $A = (U, X, Y, \Sigma^{in}, \Sigma^{int}, \Sigma^{out}, \Theta, \mathcal{D}, \mathcal{W})$ given by

- $U = (U_1 \cup U_2) \Leftrightarrow (Y_1 \cup Y_2)$, $X = X_1 \cup X_2$, $Y = Y_1 \cup Y_2$
- $\Sigma^{in} = (\Sigma_1^{in} \cup \Sigma_2^{in}) \Leftrightarrow (\Sigma_1^{out} \cup \Sigma_2^{out})$, $\Sigma^{int} = \Sigma_1^{int} \cup \Sigma_2^{int}$, $\Sigma^{out} = \Sigma_1^{out} \cup \Sigma_2^{out}$
- $\Theta = \{s \in \mathbf{V} \mid s\lceil V_1 \in \Theta_1 \wedge s\lceil V_2 \in \Theta_2\}$
- Define, for $i \in \{1, 2\}$, projection function $\pi_i : \Sigma \rightarrow \Sigma_i$ by $\pi_i(a) \triangleq a$ if $a \in \Sigma_i$ and $\pi_i(a) \triangleq e$ otherwise. Then $\mathcal{D}$ is the subset of $\mathbf{V} \times \Sigma \times \mathbf{V}$ given by

  $$(s, a, s') \in \mathcal{D} \Leftrightarrow s\lceil V_1 \overset{\pi_1(a)}{\Leftrightarrow\!\!\rightarrow}_1 s'\lceil V_1\ \wedge\ s\lceil V_2 \overset{\pi_2(a)}{\Leftrightarrow\!\!\rightarrow}_2 s'\lceil V_2$$

- $\mathcal{W}$ is the set of trajectories over V given by

  $$w \in \mathcal{W} \Leftrightarrow w \downarrow V_1 \in W_1 \wedge w \downarrow V_2 \in W_2$$

Proposition 2. *$A_1 \| A_2$ is a HIOA.*

Theorem 3. *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 $A_1 \| B \leq A_2 \| B$.*

Two natural hiding operations can be defined on any HIOA A:
(1) If $S \subseteq \Sigma_A^{out}$, then $\mathsf{ActHide}(S, A)$ is the HIOA B that is equal to A except that $\Sigma_B^{out} = \Sigma_A^{out} \Leftrightarrow S$ and $\Sigma_B^{int} = \Sigma_A^{int} \cup S$.
(2) If $Z \subseteq Y_A$, then $\mathsf{VarHide}(Z, A)$ is the HIOA B that is the equal to A except that $Y_B = Y_A \Leftrightarrow Z$ and $X_B = X_A \cup Z$.

Theorem 4. *Suppose A and B are HIOA's with $A \leq B$, and let $S \subseteq \Sigma_A^{out}$ and $Z \subseteq Y_A$.*
Then $\mathsf{ActHide}(S, A) \leq \mathsf{ActHide}(S, B)$ *and* $\mathsf{VarHide}(Z, A) \leq \mathsf{VarHide}(Z, B)$.

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 $f^t \in \mathcal{F}_v$, where f^t is the function from T to the domain of v given by $f^t(t') \triangleq f(t' + t)$. Intuitively, the dynamic type $\mathcal{F}_v$ gives the collection of allowed trajectories for v. For instance, if $T = \mathsf{R}$ and v has domain 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 $z \in Z$, $w \downarrow z \in \mathcal{F}_z \lceil dom(w)$.

An *I/O behavior* is a triple $P = (U, Y, \mathcal{B})$, where

- U is a set of typed *input* variables;
- Y is a set of typed *output* variables with $U \cap Y = \emptyset$; we write $V \triangleq U \cup Y$;
- $\mathcal{B} \subseteq \mathcal{F}\text{-}trajs(V)$ 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 $t \in dom(w) \cap dom(w')$,
  $$(w \lhd t) \downarrow U = (w' \lhd t) \downarrow U \Rightarrow w(t) \lceil Y = w'(t) \lceil Y$$
  B2 (freedom of inputs)
  $$\forall w \in Full(\mathcal{F}\text{-}trajs(U))\ \exists w' \in Max(\mathcal{B}) : w' \downarrow U \leq w$$
  B3 (nonZenoness)
  $$Max(\mathcal{B}) \subseteq Closed(\mathcal{B}) \cup Full(\mathcal{B})$$

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 ∞. Note that for any I/O behavior P there is an output state $s \in \mathbf{Y}$ such that all trajectories w in $\mathcal{B}$ begin with s, i.e., $w(0)\lceil Y = s$.

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 $\mathcal{B}_P$. 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 $\mathcal{B}_i$, 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* $P_1 || P_2$ 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
- $\mathcal{B} \subseteq \mathcal{F}\text{-}trajs(U \cup Y)$ is given by $w \in \mathcal{B} \Leftrightarrow w \downarrow V_1 \in \mathcal{B}_1 \wedge w \downarrow V_2 \in \mathcal{B}_2$.

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 1. Suppose $T = \mathsf{R}$. For u, y variables whose dynamic type is the set of functions from R to R that have left-hand limits, define $\mathsf{Copy}(u, y)$ 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 $\mathsf{Copy}(u, y)$ and $\mathsf{Copy}(y, u)$ 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 $P = P_1 || P_2$ is an I/O behavior and, for each trajectory w of P,

$$w \in Max(\mathcal{B}_P) \Leftrightarrow (w \downarrow V_1 \in Max(\mathcal{B}_1) \vee w \downarrow V_2 \in Max(\mathcal{B}_2)).$$

Example 2. Suppose $T = \mathsf{R}$. For u, y variables whose dynamic type is the set of functions from R to R that have left-hand limits, define $\mathsf{Add1}(u, y)$ 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 $\mathsf{Add1}(u, y)$ and $\mathsf{Add1}(y, u)$ are compatible but not strongly compatible, even though their composition is an I/O behavior.

Let A be a HIOA and let $l \in \mathbf{L}_A$ 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 $w \in W$, $w(0)\lceil L_A = l$, 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 $(s\lceil L_1)$-process W_1 of A_1, and for each $(s\lceil L_2)$-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* ρ for A is a function that specifies, for each sentence α of A with $l = \alpha.\mathit{lstate}\lceil L_A$,

1. an l-process W^α of A,
2. a function $g^\alpha : \mathit{Closed}(W^\alpha) \times \Sigma_A^{in} \rightarrow \mathbf{L}_A$ satisfying

$$g^\alpha(w, a) = l \Rightarrow w.\mathit{lstate} \overset{a}{\Leftrightarrow\!\!\!\rightarrow}_A l.$$

3. a function $f^\alpha : \mathit{Closed}(\mathit{Max}(W^\alpha)) \rightarrow (\Sigma_A^{loc} \times \mathbf{L}_A)$ satisfying

$$f^\alpha(w) = (a, l) \Rightarrow w.\mathit{lstate} \overset{a}{\Leftrightarrow\!\!\!\rightarrow}_A l,$$

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

$$\mathcal{I} = w_1 \; a_1 \; b_1 \; w_2 \; a_2 \; b_2 \; \cdots$$

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

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 $b_i = \mathsf{T}$ 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 α. The outcome of the game is described formally in the following definition.

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

Let $l = \alpha.lstate \lceil L_A$. Then $\alpha_0 \triangleq \alpha \; e \; \wp(w_1(0) \cup l)$.

Here we extend α in a trivial way to a sentence in order to get into a situation where strategy ρ can be applied in combination with environment sequence $\mathcal{I}$. 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 α_i in terms of α_{i-1} as follows.

If α_{i-1} is admissible then $\alpha_i \triangleq \alpha_{i-1}$.

Otherwise, α_{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 $t = w_i.ltime$ and $t' = w_i'.ltime$. We distinguish between three cases:

1. If $t = t' = \infty$ then

   $$\alpha_i \triangleq \alpha_{i-1} \frown w_i'.$$

   This is the case where both the system and the environment have decided not to perform any discrete action.
2. If $t < t'$ or $t = t' < \infty \wedge b_i = \mathsf{T}$, then

   $$\alpha_i \triangleq \alpha_{i-1} \frown ((w_i' \trianglelefteq t) \; a_i \; \wp(w_{i+1}(0) \cup g^{\alpha_i}(w_i' \trianglelefteq t, a_i))).$$

   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 $t = t' < \infty \wedge b_i = \mathsf{F}$ and if we let $f^{\alpha_i}(w_i') = (a_i', l_i)$, then

$$\alpha_i \triangleq \alpha_{i-1} \frown (w_i' \; a_i' \; \wp(w_{i+1}(0) \cup l_i)).$$

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 α_i does not depend on the choice of w_i^+ since by axiom **B1** the prefix $w_i' \trianglelefteq t$ of w_i' that is used in the construction is determined uniquely by the fixed prefix w_i of w_i^+.

Proposition 5. *$\mathcal{O}_{\rho,\mathcal{I}}(\alpha)$ is a Zeno or admissible hybrid execution of A.*

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

We now come to the main result of this paper.

Theorem 6. *Suppose A_1 and A_2 are strongly compatible, receptive HIOA's. Then $A_1 \| A_2$ is receptive.*

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

Theorem 7. *Suppose A is a receptive HIOA, and let $S \subseteq \Sigma_A^{out}$ and $Z \subseteq Y_A$. Then $\mathsf{ActHide}(S, A)$ and $\mathsf{VarHide}(Z, A)$ are receptive.*

5.3 Strong Compatibility vs. Compatibility

In order to apply Theorem 6, one has to establish that the HIOA's A_1 and 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 A_1 and A_2 in such a class, the condition of strong compatibility in Theorem 6, 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

B4 $\forall w, w' \in \mathcal{B} : dom(w) = dom(w') \;\Rightarrow\; w \downarrow Y = w' \downarrow Y$

It is easy to verify that two autistic processes are strongly compatible iff 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 $\mathcal{C}^1$, 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 $\mathcal{C}^1$ 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 $\mathcal{C}^1$ 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

1. M. Abadi and L. Lamport. Composing specifications. *ACM Transactions on Programming Languages and Systems*, 1(15):73–132, 1993.
2. 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.
3. R. Alur, C. Courcoubetis, T.A. Henzinger, and P.-H. Ho. Hybrid automata: an algorithmic approach to the specification and verification of hybrid systems. In Grossman et al. [8], pages 209–229.
4. J.C.M. Baeten and W.P. Weijland. *Process Algebra*. Cambridge Tracts in Theoretical Computer Science 18. Cambridge University Press, 1990.
5. D.J.B. Bosscher, I. Polak, and F.W. Vaandrager. Verification of an audio control protocol. In *Proc. FTRTFT'94*, LNCS 863, pages 170–192. Springer-Verlag, 1994.
6. D. Dill. *Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits*. ACM Distinguished Dissertations. MIT Press, 1988.
7. R. Gawlick, R. Segala, J.F. Søgaard-Andersen, and N. Lynch. Liveness in timed and untimed systems. In *Proceedings* 21^{th} *ICALP*, LNCS 820. Springer-Verlag, 1994. A full version appears as MIT Technical Report number MIT/LCS/TR-587.
8. R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, editors. *Hybrid Systems*, LNCS 736. Springer-Verlag, 1993.
9. L. Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, March 1994.
10. N.A. Lynch. Modelling and verification of automated transit systems, using timed automata, invariants and simulations, 1996. This volume.
11. N.A. Lynch and M.R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proceedings* 6^{th} *PODC*, pages 137–151, August 1987. A full version is available as MIT Technical Report MIT/LCS/TR-387.
12. N.A. Lynch and F.W. Vaandrager. Forward and backward simulations – part II: Timing-based systems. Report CS-R9314, CWI, Amsterdam, March 1993. To appear in *Information and Computation*.
13. N.A. Lynch and F.W. Vaandrager. Forward and backward simulations. part I: Untimed systems. *Information and Computation*, 121(2):214–233, September 1995.
14. N.A. Lynch and H.B. Weinberg. Proving correctness of a vehicle maneuver: Deceleration. In *Proceedings Second European Workshop on Real-Time and Hybrid Systems*, Grenoble, France, June 1995.
15. O. Maler, Z. Manna, and A. Pnueli. From timed to hybrid systems. In *Proceedings REX Workshop*, LNCS 600, pages 447–484. Springer-Verlag, 1992.
16. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, 1992.
17. A. Nerode and A. Yakhnis. Concurrent programs as strategies in games. In Y. Moschovakis, editor, *Logic from Computer Science*. Springer-Verlag, 1992.
18. A. Pnueli and J. Sifakis, editors. *Special Issue on Hybrid Systems of Theoretical Computer Science,* 138(1). Elsevier Science Publishers, February 1995.
19. E.D. Sontag. *Mathematical Control Theory — Deterministic Finite Dimensional Systems*, TAM 6. Springer-Verlag, 1990.
20. H.B. Weinberg, N.A. Lynch, and N. Delisle. Verification of automated vehicle protection systems, 1996. This volume.

This article was processed using the LaTeX macro package with LLNCS style