Hybrid I/O Automata

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

Computing Science Institute/

CSI-R9907 April 1999

Computing Science Institute Nijmegen
Faculty of Mathematics and Informatics
Catholic University of Nijmegen
Toernooiveld 1
6525 ED Nijmegen
The Netherlands

Hybrid I/O Automata*

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, 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

We propose a new *hybrid automaton* model that is capable of describing both continuous and discrete behavior. The model, which extends the timed automaton model of [33, 42] and the phase transition system models of [36, 2], allows communication among components using both shared variables and shared actions. The main contributions of this paper are: (1) a definition of hybrid automata and of an implementation relation based on *hybrid traces*, (2) a definition of a *simulation* between hybrid automata and a proof that existence of a simulation implies the implementation relation, (3) a definition of *composition* and *hiding* operations on hybrid automata and a proof that these operations respect the implementation relation, (4) a definition of *hybrid I/O automata*, which specialize hybrid automata by an additional distinction between input and output, and a proof that the results on simulation relations, composition and hiding carry over to this new setting, and (5) 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.
CR Subject Classification (1991): D.2.4, F.1.1, F.3.1.
Keywords & Phrases: Hybrid Systems, I/O Automata, Hybrid Automata, Simulation Relations, Compositionality, I/O Behaviors, Receptiveness.

*A preliminary version of this paper appeared as [29].

†Supported by NSF Grant 9225124-CCR, U.S. Dept. of Transportation Contract DTRS95G-0001-YR.8, AFOSR-ONR Contract F49620-94-1-0199, AFOSR Contract F49620-97-1-0337, and ARPA Contracts F19628-95-C-0118 and N00014-92-J-4033.

‡The research which led to this paper was initiated while the author was employed at CWI, Amsterdam, The Netherlands. The work was also supported by the HCM network EXPRESS and Esprit Project 26270, Verification of Hybrid Systems (VHS).

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

Contents

1 Introduction

In recent years, there has been a fast growing interest in *hybrid systems* [17, 39, 9, 6, 35, 18, 41, 44] — 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 [34, 27, 47, 45, 46, 28, 23, 24], intelligent vehicle highway systems [15, 26], avionics [25], automotive control [16], and consumer electronics [8].

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 [30, 7, 22]). 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?

System model. Our new *hybrid automaton (HA)* model is based on infinite state machines and contains no finiteness restrictions. 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 external and internal variables, and the transition labels (or actions) are partitioned into external and internal actions. In this way we can both model communication via shared variables, and communication via shared actions. The model allows us to answer questions 2 and 3. In order to answer question 4, we define *hybrid I/O automata (HIOA)*, which specialize hybrid

automata by an additional distinction between input and output. More structure will have to be added in order to deal with applications. HA's are inspired by the timed automata of [33, 42], the phase transition system models of [36, 2, 20], and the hybrid control systems of [10, 11]. The main difference between HA's and timed automata is that, as in phase transition and hybrid control 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, whereas in the work on hybrid control systems questions related to control and synthesis have been studied. Questions 2–4 have not been addressed and perhaps for this reason the external interface is not an integral part of phase transition systems and hybrid control systems. Questions 2–4 have been address by Alur and Henzinger [4, 5] in their work on reactive modules. In fact, our notions of input, output and internal variables correspond to their notions of external, interface and private module variables, respectively. In reactive modules all communication takes place via shared variables, and there are no shared actions. In the terminlogy of reactive modules, the only await dependencies that we allow in the HIOA model are dependencies of internal variables from input and output variables. In this way circular dependencies between variables are avoided when we compose HIOA's in parallel.

Implementation relation. The implementation relation that we propose is simply inclusion of the sets of hybrid traces. A *hybrid trace* records occurrences of external actions, and the evolution of external variables during an execution of a system. Thus HA $\mathcal{B}$ *implements* HA $\mathcal{A}$ if every behavior of $\mathcal{B}$ is allowed by $\mathcal{A}$. In this case, $\mathcal{B}$ is typically more deterministic than $\mathcal{A}$, both at the discrete and the continuous level. For instance, $\mathcal{A}$ might produce an output at an arbitrary time before noon, whereas $\mathcal{B}$ produces an output sometime between 10 and 11AM. Or $\mathcal{A}$ might allow any smooth trajectory for output variable y with $\dot{y} \in [0, 2]$, whereas $\mathcal{B}$ 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 [31] for an overview). In this paper we propose a definition of a *simulation* between HA'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") [19, 30]. In other approaches there are no transition labels to synchronize on, and communication between system components is achieved via shared variables [37, 22, 4]. 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 [21, 13]. 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 [43]. 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 communications of discrete variations of signals. By requiring that each output variable/action is controlled by at most one automaton, and that internal variables/actions of one automaton cannot be shared by the variables/actions of any other automaton, we avoid circular dependencies between variables in the composition.

We prove that our composition operator respects the implementation relation: if $\mathcal{A}_1$ implements $\mathcal{A}_2$ then $\mathcal{A}_1$ composed with $\mathcal{B}$ implements $\mathcal{A}_2$ composed with $\mathcal{B}$. Such a result is essential for compositional design and verification of systems.

Receptiveness. The class of HA'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* HA's as implementations, i.e., HA's in which time can advance to infinity independently of the input provided by the environment. Inspired by earlier work of [14, 1, 42] 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 [38]. 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, an HA 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.

In order to define receptivity, we need to specialize the HA model by adding a distinction between input and output. We prove that all our theorems on simulation relations, composition and hiding carry over to the resulting HIOA model. The main technical result that we prove about receptivity is that, assuming certain compatibility conditions, receptiveness is preserved by composition.

Receptivity for hybrid systems is also studied in [5]. A similar definition of receptivity is proposed, and shown to be preserved by parallel composition. However, in [5] no circular dependencies ("feedback loops") are allowed between the continuous variables of different components, a drastic restriction which very much simplifies the analysis.

2 Mathematical Preliminaries

2.1 Functions

With $dom(f)$ and $range(f)$ we denote the domain and range, respectively, of a function f. If f is a function and S a set, then we write $f \lceil S$ for the restriction of f to S, i.e., the function g with $dom(g) = dom(f) \cap S$ satisfying $g(c) = f(c)$ for each $c \in dom(g)$. We say that two functions f and g are *compatible* if $f \lceil dom(g) = g \lceil dom(f)$. If f and g are compatible functions then we write $f \cup g$ for the function h with $dom(h) = dom(f) \cup dom(g)$ satisfying, for each $c \in dom(h)$, if $c \in dom(f)$ then $h(c) = f(c)$ else $h(c) = g(c)$. More generally, if F is a set of pairwise compatible functions then we write $\bigcup F$ for the unique function g with $dom(g) = \bigcup\{dom(f) \mid f \in F\}$ satisfying, for each $f \in F$ and $c \in dom(f)$, $g(c) = f(c)$. If f is a function whose range is a set of functions, and S is a set, then we write $f \downarrow S$ for the restriction of the functions in $range(f)$ to S, i.e., the function g with $dom(g) = dom(f)$ defined by $g(c) \triangleq f(c) \lceil S$. The restriction operation $\downarrow$ is extended to sets of functions by pointwise extension. Also, if f is a function whose range consists of a set of functions that all have an element d in their domain, then $f \downarrow d$ is the function with domain $dom(f)$ defined by $f \downarrow d\ (c) \triangleq f(c)(d)$.

2.2 Sequences

Let S be any set. The sets of finite and infinite sequences of elements of S are denoted by S^* and S^ω, respectively. Concatenation of a finite sequence with a finite or infinite sequence is denoted by juxtaposition; λ denotes the empty sequence and the sequence containing one element $c \in S$ is denoted c. If σ is a nonempty sequence then $head(\sigma)$ returns the first element of σ, and $tail(\sigma)$ returns σ with its first element removed. Moreover, if σ is finite, then $last(\sigma)$ returns the last element of σ, and $init(\sigma)$ returns σ with its last element removed.

2.3 Time

Throughout this paper, we fix a *time axis* T, which is a compact subgroup of $(\mathsf{R}, +)$, the real numbers with addition. The reader may find it convenient to think of T as the set R of real numbers, but also the set Z of integers and the singleton set $\{0\}$ are examples of allowed time axes. An *interval* J is a nonempty, convex subset of T. We denote intervals as usual: $[t_1, t_2] = \{t \in \mathsf{T} \mid t_1 \leq t \leq t_2\}$, etc. An interval J is right-open (left-open) if it does not have a maximum (minimum) element, and right-closed (left-closed) otherwise. We write $\max J$ and $\min J$ for the maximum and minimum elements, respectively, of an interval J (if they exist), and $\inf J$ and $\sup J$ for the infimum and supremum, respectively, of J in $\mathsf{T} \cup \{-\infty, \infty\}$. For $K \subseteq \mathsf{T}$ and $t \in \mathsf{T}$, we define $K + t \triangleq \{t' + t \mid t' \in K\}$. Similarly, for a function f with domain K, we define $f + t$ to be the function with domain $K + t$ satisfying, for each $t' \in K + t$, $(f + t)\ (t') = f(t' - t)$.

3 Hybrid Automata and Their Behavior

In this section we introduce hybrid automata and define an implementation relation between these automata.

3.1 Variables, Valuations and Trajectories

We assume a universal set V of *variables*. To each variable v we associate a set of values $type(v)$, referred to as the *type* of the variable. A *valuation* of a set of variables 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 of V. Often, valuations will be referred to as *states*.

We also assume, for each variable $v \in \mathcal{V}$, a *dynamic type* $dtype(v)$, which is a set of (partial) functions from T to $type(v)$. We require $dtype(v)$ to be *time-invariant*: for each $f \in dtype(v)$ and each $t \in \mathsf{T}$, also $f+t \in dtype(v)$. Intuitively, the dynamic type $dtype(v)$ gives the collection of allowed trajectories for v. For instance, if $\mathsf{T} = \mathsf{R}$ and v has type R, then $dtype(v)$ can be the set of continuous or smooth functions, the set of integrable functions, or the set of measurable locally essentially bounded functions [43]. If $dtype(v)$ is the set of constant functions then v is called a *discrete* variable, as in [36].

Let V be a set of variables. A *trajectory* over V is a (total) function $\tau : J \to val(V)$, where J is a left-closed interval of T with left endpoint equal to 0, such that for each $v \in V$ there is an $f \in dtype(v)$ with $\tau \downarrow v = f \lceil dom(\tau)$. We write $trajs(V)$ for the set of all trajectories over V. A trajectory τ is *closed* if its domain is a (finite) closed interval and *full* if its domain equals $\mathsf{T}^{\geq 0} \triangleq \{t \in \mathsf{T} \mid t \geq 0\}$. For T a set of trajectories, $closed(T)$ and $full(T)$ denote the subsets of closed and full trajectories in T, respectively. If τ is a trajectory then $\tau.ltime$, the *limit time* of τ, is the supremum of $dom(\tau)$. Similarly, define $\tau.fstate$, the *first state* of τ, to be $\tau(0)$, and if τ is closed, define $\tau.lstate$, the *last state* of τ, to be $\tau(\tau.ltime)$. A trajectory τ is *finite* if its domain is a finite interval. 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 τ a trajectory and $t \in \mathsf{T}^{\geq 0}$, let $\tau \trianglelefteq t \triangleq \tau \lceil [0,t]$ and $\tau \triangleleft t \triangleq \tau \lceil [0,t)$. Note that $\tau \triangleleft 0$ has an empty domain and is thus not a trajectory. By convention, $\tau \trianglelefteq \infty \triangleq \tau \triangleq \tau \triangleleft \infty$. Similarly we define, for τ a trajectory, J a left-closed interval, and $t = \min J \in dom(\tau)$, the *curtailment* of τ to J to be the trajectory $\tau \swarrow J \triangleq (\tau \lceil J) - t$.

If τ and τ' are trajectories, τ is finite, and τ is closed implies $\tau.lstate = \tau'.fstate$ then the *concatenation* of τ and τ' is the function $\tau \frown \tau' \triangleq \tau \cup (\tau' + \tau.ltime)$. Note that $\tau \frown \tau'$ need not be a trajectory since it may not respect the dynamic types. More generally, if $\tau_0 \tau_1 \tau_2 \cdots$ is an infinite sequence of finite trajectories such that for all i, τ_i is closed implies $\tau_i.lstate = \tau_{i+1}.fstate$, then the *infinite concatenation* of this sequence is the function $\tau_0 \frown \tau_1 \frown \tau_2 \ldots \triangleq \bigcup\{\tau_i + \sum_{j<i} \tau_j.ltime \mid i \in \mathsf{N}\}$.

Trajectory τ is a *prefix* of trajectory τ', notation $\tau \leq \tau'$, if $\tau = \tau' \lceil dom(\tau)$. Note that $\tau \leq \tau'$ iff either $\tau = \tau'$ or $\tau' = \tau \frown \tau''$, for some τ''. For T a set of trajectories over V, $pref(T)$

denotes the *prefix-closure* of T: $pref(T) \triangleq \{\tau \in trajs(V) \mid \exists \tau' \in T : \tau \leq \tau'\}$. We say that T is *prefix closed* if $T = pref(T)$. A trajectory in T is *maximal* if it is not a prefix of any other trajectory in T. We write $max(T)$ for the subset of maximal trajectories in T.

3.2 Hybrid Automata

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

- Two disjoint sets W and X of variables, the *external* and *internal* variables, respectively.

  We write $V \triangleq W \cup X$, let $s, ..$ range over $val(V)$ and $\tau, ..$ over $trajs(V)$.

- Two disjoint sets E and H of *external* and *internal actions*, respectively.

  We assume that E contains a special element e, the *environment action*, which represents the occurrence of a discrete transition outside the system. We write $A \triangleq E \cup H$ and let $a, ..$ range over A.

- A nonempty set $\Theta \subseteq val(V)$ of *start states*.

- A set $\mathcal{D} \subseteq val(V) \times A \times val(V)$ of *discrete transitions* satisfying:

  D (Existence of stuttering transitions)
  $\forall s \in val(V) : s \xrightarrow{e}_{\mathcal{A}} s$.

  We use $s \xrightarrow{a}_{\mathcal{A}} s'$ as shorthand for $(s, a, s') \in \mathcal{D}$. We drop the subscript $\mathcal{A}$, and write $s \xrightarrow{a} s'$, whenever it is clear from the context.

- A set $\mathcal{T}$ of trajectories over V satisfying:

  T1 (Existence of point trajectories)
  $\forall s \in val(V) : \wp(s) \in \mathcal{T}$.

  T2 (Closure under curtailment)
  $\forall \tau \in \mathcal{T}\ \forall J$ left-closed subinterval of $dom(\tau) : \tau \swarrow J \in \mathcal{T}$.

  T3 (Completeness)
  $\forall \tau \in trajs(V) : (\forall t \in \mathsf{T}^{\geq 0} : \tau \trianglelefteq t \in \mathcal{T}) \quad \Rightarrow \quad \tau \in \mathcal{T}$.

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 $\mathcal{T}$ iff all its prefixes are in $\mathcal{T}$. (Actually axiom **T3** does not say "iff", but the missing direction follows easily from **T2**.) We call a transition $s \xrightarrow{e} s$ a *stuttering transition* for consistency with the existing literature on shared memory models [21, 22].

Notation In the sequel, the components of a HA $\mathcal{A}$ will often be denoted by $W_{\mathcal{A}}$, $X_{\mathcal{A}}$, $E_{\mathcal{A}}$, etc. The components of a HA $\mathcal{A}_i$ will be denoted by W_i, X_i, E_i, etc. We sometimes omit these subscripts, where no confusion is likely.

Example 3.1 (Timed automata) A timed automaton T in the sense of Lynch and Vaandrager [42, 33, 32] can be represented as a hybrid automaton $\mathcal{A}$ without external variables. Specifically, the external actions of $\mathcal{A}$ are the visible actions of T plus the environment action e, the only internal action of $\mathcal{A}$ is ι, and the only internal variable of $\mathcal{A}$ is x, where the domain of x is the set of states of T. If, for simplicity we identify the action τ of T with the action ι of $\mathcal{A}$, and each state s of T with the state of $\mathcal{A}$ in which x has value s, then the discrete transitions of $\mathcal{A}$ are all the discrete transitions of $\mathcal{A}$ plus stuttering transitions for each state, and the trajectories of $\mathcal{A}$ are all the trajectories of T. ■

Example 3.2 (Hybrid Systems) A hybrid system $\mathcal{S} = (Loc, Var, Lab, Edg, Act, Inv)$ in the sense of Alur et al [2] can be represented as a hybrid automaton $\mathcal{A} = (W, X, E, H, \Theta, \mathcal{D}, \mathcal{T})$ with

- W equal to the set Var of real-valued variables. In the approach of [2] the time domain equals the set R of real numbers. Since in the model of [2] there is no notion of dynamic type, we define the dynamic type of all variables in W to be maximal, i.e., the set of functions from R to R.
- $X = \{loc\}$, where loc a discrete internal variable whose type is the set Loc of locations. As usual $V = W \cup X$.
- E equal to the set Lab of synchronization labels. The stutter label $\tau \in Lab$ is identified with the environment action e.
- $H = \emptyset$: there are no no internal actions.
- $\Theta = \{s \in val(V) \mid \forall l \in Loc : s(loc) = l \Rightarrow s \lceil W \in Inv(l)\}$. Hybrid systems do not have a component that describes initial states. However, since a run of a hybrid system may only visit a state s if the valuation of its real valued variables is contained in the invariant set $Inv(s(loc))$, we need to constrain the set of initial states of $\mathcal{A}$. Formally, the definition of a hybrid system does not exclude the trivial case that Θ is empty; of course our translation only applies if $\Theta \neq \emptyset$.
- $\mathcal{D} = \{(s, a, s') \in \Theta \times E \times \Theta \mid \exists \mu : (s(loc), a, \mu, s'(loc)) \in Edg \wedge (s \lceil W, s' \lceil W) \in \mu\} \cup \{(s, e, s) \mid s \in val(V)\}$. Each discrete transition of $\mathcal{S}$ corresponds to a set of transitions of $\mathcal{A}$. Apart from the stutter transitions which we need to make axiom **D** hold, we only consider discrete transitions between states in Θ, since these are the only transitions that can occur in runs.

- $\mathcal{T} = \{\wp(s) \mid s \in val(V)\} \cup \{\tau \in trajs(V) \mid range(\tau) \subseteq \Theta \wedge \exists f \in Act(\tau(0)(loc)) : \tau \downarrow W \leq f\}$. $\mathcal{T}$ includes all point trajectories to make axiom **T1** hold, and furthermore all trajectories that only visit states in Θ and correspond to an activity in Act.

Axioms **D**, **T1** and **T3** are guaranteed by construction; axiom **T2** is guaranteed by the time-invariance property imposed on the activities of a hybrid system. ■

Concrete examples of hybrid automata can be found in numerous papers that deal with realistic applications [34, 27, 47, 45, 46, 28, 23, 24, 15, 26, 25, 16, 8].

3.3 Hybrid Executions

A *(hybrid) execution fragment* of a HA $\mathcal{A}$ 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 $\mathcal{T}$ and each a_i is an action in A.
2. If α is a finite sequence then it ends with a trajectory.
3. If τ_i is not the last trajectory in α then $dom(\tau_i)$ is closed and $\tau_i.lstate \stackrel{a_{i+1}}{\rightarrow} \tau_{i+1}.fstate$.

Thus a hybrid execution fragment is essentially an alternating sequence of trajectories and discrete transitions that span between these trajectories. It records all the instantaneous, discrete state changes that occur during a specific evolution of a system, as well as the state changes that occur within trajectories while time advances. When HAs are used to model real-world systems the values of the state variables will typically change continuously within a trajectory, but this continuity is not imposed by the above definition. We write $h\text{-}frag(\mathcal{A})$ for the set of all hybrid execution fragments of $\mathcal{A}$.

Given an execution fragment $\alpha = \tau_0\ a_1\ \tau_1\ a_2\ \tau_2 \cdots$, we say that an occurrence of an action a_i in α is *stuttering* if a_i equals e and its neighbor trajectories can be concatenated, i.e., $\tau_{i-1}.lstate = \tau_i.fstate$ and $\tau_{i-1} \frown \tau_i \in \mathcal{T}$. We define the *stuttering-free version of* α to be the execution fragment α' of $\mathcal{A}$ obtained from α by removing all occurrences of stuttering e actions, and concatenating all adjacent trajectories. We say that two execution fragments are *stuttering equivalent* if their stuttering-free versions are the same.

We define the *limit time* of α, denoted by $\alpha.ltime$, to be $\sum_i \tau_i.ltime$. Further, we define the *first state* of α, $\alpha.fstate$, to be $\tau_0.fstate$. We distinguish several sorts of execution fragments: α is defined to be

- an *execution* if the first state of α is a start state,
- *closed* if α is a finite sequence and the domain of its final trajectory is a closed interval,
- *admissible* if $\alpha.ltime = \infty$,

- *Zeno* if α is neither closed nor admissible, and
- a *sentence* if α is a finite execution that ends with a point trajectory.

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

A finite execution fragment α and an execution fragment α' of $\mathcal{A}$ can be *concatenated* if $last(\alpha) \frown head(\alpha')$ is defined and a trajectory of $\mathcal{A}$. (Note that *last* and *head* are ordinary sequence operations and they return trajectories.) In this case, the *concatenation* $\alpha \frown \alpha'$ is the execution fragment $init(\alpha)$ $(last(\alpha) \frown head(\alpha'))$ $tail(\alpha')$.

Let α, α' be execution fragments. We say that α is a *prefix* of α', notation $\alpha \leq \alpha'$, if either $\alpha = \alpha'$ or there exists some execution fragment α'' such that $\alpha \frown \alpha'' = \alpha'$.

Example 3.3 (Hybrid executions and timed automata) Consider a timed automaton T and its corresponding hybrid automaton $\mathcal{A}$ built according to Example 3.1. If we again identify τ and ι, and each state s of T with the state of $\mathcal{A}$ in which x has value s, then each timed execution fragment of T according to [33, 42] is also a stuttering-free hybrid execution fragment of $\mathcal{A}$. This construction can be inverted, and indeed there is a one-to-one correspondence between the timed execution fragments of T and the stuttering-free hybrid execution fragments of $\mathcal{A}$. Thus, all the concepts that are defined on timed automata based on timed execution fragments have corresponding concepts within hybrid automata. The corresponding concepts do not distinguish between stuttering equivalent hybrid execution fragments. ■

Example 3.4 (Hybrid executions and hybrid systems) Consider a hybrid system $\mathcal{S}$ and its corresponding hybrid automaton $\mathcal{A}$ built according to Example 3.2. There is a close correspondence between runs of $\mathcal{S}$ and hybrid executions of A. There does not exist a one-to-one relation however:

1. A run does not include information about the actions (synchronization labels) that occur during the evolution of a hybrid system.
2. A run is essentially a sequence $(\tau_0, t_0)(\tau_1, t_1)\cdots$ of pairs of an infinite trajectory and a time point. Intuitively, the system first follows trajectory τ_0 until time t_0, then trajectory τ_1 until time t_1, etc. This means that a run contains irrelevant information which cannot be incorporated in a hybrid execution: the part of τ_0 after time t_0, the part of τ_1 after time t_1, etc.
3. Since a run may only contain right closed trajectories, a hybrid execution that ends with a right-open trajectory needs to be encoded via the introduction of an infinite number of trajectories interleaved with stuttering steps.

■

3.4 Hybrid Traces

Suppose $\alpha = \tau_0\, a_1\, \tau_1\, a_2\, \tau_2 \cdots$ is a hybrid execution fragment. In order to define the *hybrid trace* of α, we first restrict the trajectories in α to the external variables, and rename all internal and environment actions to ι:

$$\gamma \;=\; (\tau_0 \downarrow W)\; vis(a_1)\; (\tau_1 \downarrow W)\; vis(a_2)\; (\tau_2 \downarrow W) \cdots,$$

where, for a an action, $vis(a)$ is defined to be equal to ι if $a \in H \cup \{e\}$, and equal to a otherwise. An occurrence of ι in γ is called *inert* if the trajectory that precedes ι can be concatenated with the trajectory that follows it (after hiding of internal variables), and the result is again a trajectory. The *hybrid trace* of α, written $htrace(\alpha)$, is defined to be the sequence obtained from γ by removing all inert ι's and concatenating the surrounding trajectories.

Lemma 3.5 *Let α_1 and α_2 be two stuttering equivalent hybrid execution fragments of $\mathcal{A}$. Then $htrace(\alpha_1) = htrace(\alpha_2)$.*

Proof: Simple consequence of the fact that stuttering e actions become inert ι's. ■

The *hybrid traces* of a hybrid automaton $\mathcal{A}$ are the hybrid traces that arise from all the closed and admissible hybrid executions of $\mathcal{A}$. We write $h\text{-}traces(\mathcal{A})$ for the set of hybrid traces of $\mathcal{A}$.

Example 3.6 (Hybrid traces and timed automata) Consider again a timed automaton T and its corresponding hybrid automaton $\mathcal{A}$ built according to Example 3.1. Let $\beta = \tau_0 a_1 \tau_1 \cdots$ be a hybrid trace of $\mathcal{A}$. Since $\mathcal{A}$ does not have any external variables, each trajectory of β associates the empty valuation with each time point in its domain. Thus, a hybrid trace β of $\mathcal{A}$ can be represented by a pair consisting of a sequence of actions paired with their time of occurrence (timed actions), and of the value $\beta.ltime$. This structure is called a *timed trace* in [33, 42]. Formally, the time of occurrence of an action is the sum of the suprema of the trajectories preceding the chosen occurrence. It is easy to observe that the transformation above can be reversed: there is a one-to-one correspondence between the timed traces of T and the hybrid traces of $\mathcal{A}$. ■

Hybrid automata $\mathcal{A}_1$ and $\mathcal{A}_2$ are *comparable* if they have the same external interface, i.e., $W_1 = W_2$ and $E_1 = E_2$. If $\mathcal{A}_1$ and $\mathcal{A}_2$ are comparable then we say that $\mathcal{A}_1$ *implements* A_2, notation $\mathcal{A}_1 \leq \mathcal{A}_2$, if the hybrid traces of $\mathcal{A}_1$ are included in those of $\mathcal{A}_2$, i.e., $h\text{-}traces(\mathcal{A}_1) \subseteq h\text{-}traces(\mathcal{A}_2)$.

4 Simulation Relations

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

1. If $r \in \Theta_{\mathcal{A}}$ then there exists $s \in \Theta_{\mathcal{B}}$ such that $r\ R\ s$.
2. If $r\ R\ s$ and $r \xrightarrow{a}_{\mathcal{A}} r'$ then $\mathcal{B}$ has a closed execution fragment α with $s = \alpha.fstate$, $htrace(\wp(r)\ a\ \wp(r')) = htrace(\alpha)$ and $r'\ R\ \alpha.lstate$.
3. If $r\ R\ s$ and τ is a closed trajectory of $\mathcal{A}$ with $r = \tau.fstate$ then $\mathcal{B}$ has a closed execution fragment α with $s = \alpha.fstate$, $htrace(\tau) = htrace(\alpha)$ and $\tau.lstate\ R\ \alpha.lstate$.

Lemma 4.1 *If R is a simulation from $\mathcal{A}$ to $\mathcal{B}$ and $r\ R\ s$, then $r \lceil W_{\mathcal{A}} = s \lceil W_{\mathcal{B}}$.*

Proof: By axiom **T1**, $\wp(r)$ is a trajectory of $\mathcal{A}$. By Property 3 of R, $\mathcal{B}$ has a closed execution fragment α with $s = \alpha.fstate$ and $htrace(\wp(r)) = htrace(\alpha)$. Thus, $r \lceil W_{\mathcal{A}} = \alpha.fstate \lceil W_{\mathcal{B}}$. That is, $r \lceil W_{\mathcal{A}} = s \lceil W_{\mathcal{B}}$. ■

Theorem 4.2 *If $\mathcal{A}$ and $\mathcal{B}$ are comparable and there is a simulation from $\mathcal{A}$ to $\mathcal{B}$, then $\mathcal{A} \leq \mathcal{B}$.*

Proof: Suppose that β is a hybrid trace of $\mathcal{A}$. We prove that β is also a hybrid trace of $\mathcal{B}$.

Let $\alpha = \tau_0 a_1 \tau_1 a_2 \tau_2 \cdots$ be an execution of $\mathcal{A}$ such that $\beta = htrace(\alpha)$. We can assume without loss of generality that all the trajectories of α are closed, since otherwise we can use axioms **T2** and **D** to break the last trajectory of α into infinitely many closed trajectories interleaved with environment actions. Let R be a simulation from $\mathcal{A}$ to $\mathcal{B}$. We define inductively a collection of closed execution fragments $\alpha_0, \alpha_1, \ldots$ of $\mathcal{B}$ such that for each i,

1. $\tau_i.lstate\ R\ \alpha_i.lstate$,
2. $\alpha_0 \frown \alpha_1 \frown \cdots \frown \alpha_i$ is an execution of $\mathcal{B}$, and
3. $htrace(\alpha_0 \frown \alpha_1 \frown \cdots \frown \alpha_i) = htrace(\tau_0 a_1 \tau_1 \cdots a_i \tau_i)$.

Thus, $\lim_{i\to\infty} \alpha_0 \frown \alpha_1 \frown \cdots \frown \alpha_i$ is an execution of $\mathcal{B}$ with trace β.

There is a start state s_0 of $\mathcal{B}$ such that $\tau_0.fstate\ R\ s_0$ and a closed execution α'_0 of $\mathcal{B}$ with first state s_0 and such that $htrace(\alpha'_0) = htrace(\tau_0)$ and $\tau_0.lstate\ R\ \alpha'_0.lstate$. We let α_0 be α'_0.

Assume by induction that Properties 1-3 hold for each $j \leq i$. Since R is a simulation relation from $\mathcal{A}$ to $\mathcal{B}$, there is a closed execution fragment α'_{i+1} of $\mathcal{B}$ such that

a. $\alpha'_{i+1}.fstate = \alpha_i.lstate$,
b. $htrace(\alpha'_{i+1}) = htrace(\wp(\tau_i.lstate) a_{i+1} \wp(\tau_{i+1}.fstate))$,
c. $\tau_{i+1}.fstate\ R\ \alpha'_{i+1}.lstate$,
d. α'_{i+1} contains only point trajectories and at least one action (this is implied by Item b and the fact that one can always append a stuttering step to an execution fragment),

and, based on Item c above, a closed execution fragment α''_{i+1} of $\mathcal{B}$ such that

e. $\alpha''_{i+1}.fstate = \alpha'_{i+1}.lstate$,

f. $htrace(\alpha''_{i+1}) = htrace(\tau_{i+1})$,

g. $\tau_{i+1}.lstate\ R\ \alpha''_{i+1}.lstate$.

From Items d and e, α''_{i+1} can be concatenated to α'_{i+1}. Define α_{i+1} to be $\alpha'_{i+1} \frown \alpha''_{i+1}$. We need to show that Properties 1-3 are valid for $j = i+1$. Property 1 follows immediately from Item g, Property 2 follows from Items a and d, and Property 3 follows from the induction hypothesis together with Items b and f. ■

5 Composition and Hiding

In this section we introduce the operations of composition and hiding for hybrid automata, and prove that hybrid trace inclusion is preserved by these operations.

The composition operation allows an automaton representing a complex system to be constructed by composing automata representing individual system components. The composition identifies actions and variables with the same name in different component automata. Common variables are shared among the component automata, and when any component automaton performs a step involving an action a, so do all component automata that have a in their signatures. We define composition as a partial, binary operation on hybrid automata. First, the operation is only defined if the initial conditions of both argument automata are consistent. Second, 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 $\mathcal{A}_2$. Third, 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

1. there exists a valuation s for $V = V_1 \cup V_2$ such that $s \lceil V_1 \in \Theta_1$ and $s \lceil V_2 \in \Theta_2$, and
2. 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 $\mathcal{A} = (W, X, E, H, \Theta, \mathcal{D}, \mathcal{T})$ given by

- $W = W_1 \cup W_2$, $X = X_1 \cup X_2$, $E = E_1 \cup E_2$, $H = H_1 \cup H_2$
- $\Theta = \{s \in val(V) \mid s \lceil V_1 \in \Theta_1 \wedge s \lceil V_2 \in \Theta_2\}$

- Let $A = A_1 \cup A_2$. Define, for $i \in \{1,2\}$, the projection function $\pi_i : A \to A_i$ by $\pi_i(a) \triangleq a$ if $a \in A_i$ and $\pi_i(a) \triangleq e$ otherwise. Then $\mathcal{D} \subseteq val(V) \times A \times val(V)$ is given by

$$(s,a,s') \in \mathcal{D} \quad \Leftrightarrow \quad s \lceil V_1 \stackrel{\pi_1(a)}{\rightarrow_1} s' \lceil V_1 \ \wedge \ s \lceil V_2 \stackrel{\pi_2(a)}{\rightarrow_2} s' \lceil V_2$$

- $\mathcal{T} \subseteq trajs(V)$ is given by $\tau \in \mathcal{T} \Leftrightarrow \tau \downarrow V_1 \in \mathcal{T}_1 \ \wedge \ \tau \downarrow V_2 \in \mathcal{T}_2$

Notation We extend the projection notation π_i $(i = 1,2)$ to states, trajectories and hybrid execution fragments in the obvious way. We further extend the projection notation to hybrid traces as follows. Let $\beta = \tau_0 a_1 \tau_1 a_2 \tau_2 \cdots$ be a hybrid trace of $\mathcal{A} = \mathcal{A}_1 \| \mathcal{A}_2$. Then $\pi_i(\beta)$ is the sequence obtained from $\pi_i(\tau_0)\ vis_i(a_1)\ \pi_i(\tau_1)\ vis_i(a_2)\ \pi_i(\tau_2) \cdots$ by removing all inert ι's and concatenating the surrounding trajectories.

Sometimes we use β to denote a *potential hybrid trace*, i.e., a sequence of alternate trajectories over $W_{\mathcal{A}}$ and actions from $(E_{\mathcal{A}} - \{e\}) \cup \{\iota\}$ without inert ι's. The notion of projection extends to potential hybrid traces in the obvious way.

Proposition 5.1 *$\mathcal{A}_1 \| \mathcal{A}_2$ is a hybrid automaton.*

Proof: Let $\mathcal{A}$ denote $\mathcal{A}_1 \| \mathcal{A}_2$ as above. We need to show that $\mathcal{A}$ satisfies the properties of a hybrid automaton (cf. Section 3.2). Disjointness of W and X, disjointness of E and H, and non-emptiness of Θ follow from compatibility. We verify the **D** and **T** properties one by one.

D Let $s \in val(V)$ and let $i \in \{1,2\}$. From **D** applied to $\mathcal{A}_i$, $s \lceil V_i \stackrel{e}{\rightarrow}_{\mathcal{A}_i} s \lceil V_i$. Thus, $s \stackrel{e}{\rightarrow}_{\mathcal{A}} s$.

T1 Let $s \in val(V)$, and let $i \in \{1,2\}$. From the definition of composition, $\pi_i(s) \in val(V_i)$, and from **T1** applied to $\mathcal{A}_i$, $\pi_i(\wp(s)) = \wp(\pi_i(s)) \in \mathcal{T}_i$. This implies that $\wp(s) \in \mathcal{T}$.

T2 Let $\tau \in \mathcal{T}$, let J be a left-closed subinterval of $dom(\tau)$ and let $i \in \{1,2\}$. From **T2** applied to $\mathcal{A}_i$, $\pi_i(\tau \swarrow J) = \pi_i(\tau) \swarrow J \in \mathcal{T}_i$. From the definition of the composition operator, $\tau \swarrow J \in \mathcal{T}$.

T3 Assume that, for each $t \in \mathsf{T}^{\geq 0}$, $\tau \trianglelefteq t \in \mathcal{T}$. Then, for each $i = 1,2$ and $t \in \mathsf{T}^{\geq 0}$, $\pi_i(\tau \trianglelefteq t) = \pi_i(\tau) \trianglelefteq t \in \mathcal{T}_i$, and from **T3** applied to $\mathcal{A}_i$, $\pi_i(\tau) \in \mathcal{T}_i$. Therefore, $\tau \in \mathcal{T}$. ■

Lemma 5.2 *Let $\mathcal{A} = \mathcal{A}_1 \| \mathcal{A}_2$ and let α be a hybrid execution of $\mathcal{A}$. Then,*

1. *α is closed iff $\pi_1(\alpha)$ is closed and $\pi_2(\alpha)$ is closed;*
2. *α is admissible iff $\pi_1(\alpha)$ is admissible and $\pi_2(\alpha)$ is admissible;*
3. *α is Zeno iff $\pi_1(\alpha)$ is Zeno and $\pi_2(\alpha)$ is Zeno.*

Proof: Simple application of the definitions. ■

Lemma 5.3 *Let $\mathcal{A} = \mathcal{A}_1 \| \mathcal{A}_2$, and let α be a hybrid execution of $\mathcal{A}$. Then, for $i = 1, 2$, $\pi_i(htrace(\alpha)) = htrace(\pi_i(\alpha))$.*

Proof: Let $\alpha = \tau_0 a_1 \tau_1 a_2 \tau_2 \cdots$. From the definitions of a hybrid trace and of π_i, $\pi_i(htrace(\alpha))$ is the sequence obtained from $\pi_i(\tau_0 \downarrow W_{\mathcal{A}})\, vis_i(vis_{\mathcal{A}}(a_1))\, \pi_i(\tau_1 \downarrow W_{\mathcal{A}}) \cdots$ by removing all inert ι's and concatenating the surrounding trajectories, while $htrace(\pi_i(\alpha))$ is the sequence obtained from $(\pi_i(\tau_0) \downarrow W_i)\, vis_i(\pi_i(a_1))\, (\pi_i(\tau_1) \downarrow W_i) \cdots$ by removing all inert ι's and concatenating the surrounding trajectories. Observe that $\pi_i(\tau_j \downarrow W_{\mathcal{A}}) = \tau_j \downarrow W_i = \pi_i(\tau_j) \downarrow W_i$ for each $j \geq 0$, and that $vis_i(vis_{\mathcal{A}}(a_j)) = vis_i(\pi_i(a_j))$ for each $j \geq 1$. Therefore, $\pi_i(htrace(\alpha)) = htrace(\pi_i(\alpha))$. ■

Lemma 5.4 *Let $\mathcal{A} = \mathcal{A}_1 \| \mathcal{A}_2$. Then h-traces$(\mathcal{A})$ is the set of potential hybrid traces of $\mathcal{A}$ whose projections are hybrid traces of $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively. That is, h-traces$(\mathcal{A}) = \{\beta \mid \pi_i(\beta) \in$ h-traces$(\mathcal{A}_i), i = 1, 2\}$.*

Proof: Let β be a hybrid trace of $\mathcal{A}$, and let $\alpha = \tau_0 a_1 \tau_1 \cdots$ be a hybrid execution of $\mathcal{A}$ such that $\beta = htrace(\alpha)$. Let $i \in \{1, 2\}$. From Lemma 5.3, $\pi_i(\beta) = htrace(\pi_i(\alpha))$. Since $\pi_i(\alpha)$ is a hybrid execution of $\mathcal{A}_i$, $\pi_i(\beta)$ is a hybrid trace of $\mathcal{A}_i$.

Conversely, let β be a potential hybrid trace of $\mathcal{A}$, i.e., a sequence of alternate trajectories over $W_{\mathcal{A}}$ and actions from $(E_{\mathcal{A}} - \{e\}) \cup \{\iota\}$ without inert ι's, so that $\pi_i(\beta)$ is a hybrid trace of $\mathcal{A}_i$, $i = 1, 2$. Then there are hybrid executions α_1 and α_2 of $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively, such that, for $i = 1, 2$, $htrace(\alpha_i) = \pi_i(\beta)$. Inserting environment actions appropriately in α_1 and α_2, we obtain two new hybrid executions $\alpha_1' = \tau_0^1 a_1^1 \tau_1^1 a_2^1 \tau_2^1 \cdots$ and $\alpha_2' = \tau_0^2 a_1^2 \tau_1^2 a_2^2 \tau_2^2 \cdots$ of $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively, such that, for $i = 1, 2$,

a. $htrace(\alpha_i') = \pi_i(\beta)$

b. for each $j \geq 0$, $\tau_j^1.ltime = \tau_j^2.ltime$

c. for each $j \geq 1$, either $a_j^1 = a_j^2$, or $a_j^1 = e$ or $a_j^2 = e$

Observe that an immediate consequence of Properties (a) and (b) is that for $i = 1, 2$ and $j \geq 0$,

d. $htrace(\tau_0^i a_1^i \tau_1^i a_2^i \tau_2^i \cdots a_j^i \tau_j^i) \leq \pi_i(\beta)$

e. $(\tau_0^1 a_1^1 \tau_1^1 a_2^1 \tau_2^1 \cdots a_j^1 \tau_j^1).ltime = (\tau_0^2 a_1^2 \tau_1^2 a_2^2 \tau_2^2 \cdots a_j^2 \tau_j^2).ltime$

We now build inductively a collection $\tau_0, \tau_1, \ldots$ of trajectories of $\mathcal{A}$ and a collection $a_1, a_2, \ldots$ of actions of $\mathcal{A}$ such that, for each $i \in \{1, 2\}$ and each $j \geq 0$,

1. $\tau_0 a_1 \tau_1 a_2 \tau_2 \cdots a_j \tau_j$ is a hybrid execution of $\mathcal{A}$
2. $\pi_i(\tau_0 a_1 \tau_1 a_2 \tau_2 \cdots a_j \tau_j) = \tau_0^i a_1^i \tau_1^i a_2^i \tau_2^i \cdots a_j^i \tau_j^i$
3. $\mathit{htrace}(\tau_0 a_1 \tau_1 a_2 \tau_2 \cdots a_j \tau_j) \leq \beta$

Then $\tau_0 a_1 \tau_1 \cdots$ is a hybrid execution of $\mathcal{A}$ whose hybrid trace is β. That is, β is a hybrid trace of $\mathcal{A}$.

For $j = 0$, since $\mathit{htrace}(\tau_0^i) \leq \beta$, $i = 1, 2$, and since $\tau_0^1.\mathit{ltime} = \tau_0^2.\mathit{ltime}$, we derive that for each variable $w \in W_1 \cap W_2$ and each time $t \leq \tau_0^1.\mathit{ltime}$, $\tau_0^1(t)(w) = \tau_0^2(t)(w)$. Thus, for each time $t \leq \tau_0^1.\mathit{ltime}$ the valuations $\tau_0^1(t)$ and $\tau_0^2(t)$ are compatible, and a trajectory τ_0 with domain $[0, \tau_0^1.\mathit{ltime}]$ can be defined as $\tau_0(t) = \tau_0^1(t) \cup \tau_0^2(t)$. Then, $\pi_i(\tau_0) = \tau_0^i$, $i = 1, 2$, which means that τ_0 is a trajectory of $\mathcal{A}$. This shows Properties 1 and 2. We are left to show that $\mathit{htrace}(\tau_0) \leq \beta$. Suppose for the sake of contradiction that $\mathit{htrace}(\tau_0) \not\leq \beta$. Express β as $\tau_0' a_1' \tau_1' \cdots$. Then, either $\tau_0.\mathit{ltime} > \tau_0'.\mathit{ltime}$, or $\tau_0.\mathit{ltime} \leq \tau_0'.\mathit{ltime}$ and $\tau_0 \lceil W_{\mathcal{A}} \not\leq \tau_0'$. In the first case either $\pi_1(\beta)$ or $\pi_2(\beta)$ would have a discrete action before time $\tau_0.\mathit{ltime}$, which is impossible since $\tau_0.\mathit{ltime} = \tau_0^1.\mathit{ltime} = \tau_0^2.\mathit{ltime}$; in the second case, either $\pi_1(\tau_0) \not\leq \tau_0^1$ or $\pi_2(\tau_0) \not\leq \tau_0^2$, which is impossible by definition of τ_0.

For the inductive step, suppose that Properties 1, 2, and 3 hold for j. Define a_{j+1} to be a_{j+1}^1, if $a_{j+1}^1 \neq e$, and be a_{j+1}^2, otherwise. Recall that, for $i \in \{1, 2\}$, $\mathit{htrace}(\tau_0^i a_1^i \tau_1^i a_2^i \tau_2^i \cdots a_j^i \tau_j^i) \leq \pi_i(\beta)$. Since, by Item e, $(\tau_0^1 a_1^1 \tau_1^1 a_2^1 \tau_2^1 \cdots a_{j+1}^1 \tau_{j+1}^1).\mathit{ltime} = (\tau_0^2 a_1^2 \tau_1^2 a_2^2 \tau_2^2 \cdots a_{j+1}^2 \tau_{j+1}^2).\mathit{ltime}$, and since, by Item b, $\tau_{j+1}^1.\mathit{ltime} = \tau_{j+1}^2.\mathit{ltime}$, we derive that for each variable $w \in W_1 \cap W_2$ and each time $t \leq \tau_{j+1}^1.\mathit{ltime}$, $\tau_{j+1}^1(t)(w) = \tau_{j+1}^2(w)$. Thus, for each time $t \leq \tau_{j+1}^1.\mathit{ltime}$ the valuations $\tau_{j+1}^1(t)$ and $\tau_{j+1}^2(t)$ are compatible, and a trajectory τ_{j+1} with domain $[0, \tau_{j+1}^1.\mathit{ltime}]$ can be defined as $\tau_{j+1}(t) = \tau_{j+1}^1(t) \cup \tau_{j+1}^2(t)$. Then, $\pi_i(\tau_{j+1}) = \tau_{j+1}^i$, $i = 1, 2$, which means that τ_{j+1} is a trajectory of $\mathcal{A}$. Furthermore, from Property 2 and the definition of a_{j+1}, for each $i \in \{1, 2\}$, $(\pi_i(\tau_j.\mathit{lstate}), \pi_i(a_{j+1}), \pi_i(\tau_{j+1}.\mathit{fstate}))$ is a discrete transition of $\mathcal{A}_i$. This proves Properties 1 and 2. We are left to show that $\mathit{htrace}(\tau_0 a_1 \tau_1 a_2 \tau_2 \cdots a_{j+1} \tau_{j+1}) \leq \beta$. Since by induction, $\mathit{htrace}(\tau_0 a_1 \tau_1 a_2 \tau_2 \cdots a_j \tau_j) \leq \beta$, we can express β as $\beta_1 \frown \beta_2$, where $\beta_1 = \mathit{htrace}(\tau_0 a_1 \tau_1 a_2 \tau_2 \cdots a_j \tau_j)$. Thus, for $i \in \{1, 2\}$, $\pi_i(\beta_2) = \mathit{htrace}(\wp(\tau_j^i.\mathit{lstate}) a_{j+1}^i \tau_{j+1}^i \cdots)$, and the problem is reduced to showing that $\mathit{htrace}(\wp(\tau_j.\mathit{lstate}) a_{j+1} \tau_{j+1}) \leq \beta_2$. We distinguish two cases.

- $\mathit{vis}_{\mathcal{A}}(a_{j+1})$ is an inert ι.

  In this case, for $i \in \{1, 2\}$, $\pi_i(\beta_2) = \mathit{htrace}(\tau_{j+1}^i \cdots)$, and the problem is reduced to showing that $\mathit{htrace}(\tau_{j+1}) \leq \beta_2$. Let $\beta_3 = \beta_2$.

- $\mathit{vis}_{\mathcal{A}}(a_{j+1})$ is not an inert ι.

  Observe that either $\mathit{vis}_{\mathcal{A}_1}(a_{j+1}^1)$ or $\mathit{vis}_{\mathcal{A}_2}(a_{j+1}^2)$ is not an inert ι. Thus, β_2 can be expressed as $\wp(\beta_2.\mathit{fstate}) \mathit{vis}_{\mathcal{A}}(a_{j+1}) \wp(\beta_3.\mathit{fstate}) \frown \beta_3$. Therefore, for $i \in \{1, 2\}$, $\pi_i(\beta_3) = \mathit{htrace}(\tau_{j+1}^i \cdots)$, and the problem is reduced to showing that $\mathit{htrace}(\tau_{j+1}) \leq \beta_3$.

Suppose for the sake of contradiction that $htrace(\tau_{j+1}) \not\leq \beta_3$. Express β_3 as $\tau_0' a_1' \tau_1' \cdots$. Then, either $\tau_{j+1}.ltime > \tau_0'.ltime$, or $\tau_{j+1}.ltime \leq \tau_0'.ltime$ and $\tau_{j+1} \lceil W_{\mathcal{A}} \not\leq \tau_0'$. In the first case either $\pi_1(\beta_3)$ or $\pi_2(\beta_3)$ would have a discrete action before time $\tau_{j+1}.ltime$, which is impossible since $\tau_{j+1}.ltime = \tau_{j+1}^1.ltime = \tau_{j+1}^2.ltime$; in the second case, either $\pi_1(\tau_{j+1}) \not\leq \tau_{j+1}^1$ or $\pi_2(\tau_{j+1}) \not\leq \tau_{j+1}^2$, which is impossible by definition of τ_{j+1}. ■

Theorem 5.5 *Suppose $\mathcal{A}_1, \mathcal{A}_2$ and $\mathcal{B}$ are HAs with $\mathcal{A}_1 \leq \mathcal{A}_2$, and each of $\mathcal{A}_1$ and $\mathcal{A}_2$ is compatible with $\mathcal{B}$. Then $\mathcal{A}_1 \| \mathcal{B} \leq \mathcal{A}_2 \| \mathcal{B}$.*

Proof: Let $\beta \in$ *h-traces*$(\mathcal{A}_1 \| \mathcal{B})$. From Lemma 5.4, $\pi_{\mathcal{A}_1}(\beta) \in$ *h-traces*$(\mathcal{A}_1)$ and $\pi_{\mathcal{B}}(\beta) \in$ *h-traces*$(\mathcal{B})$. Since $\mathcal{A}_1 \leq \mathcal{A}_2$, $\pi_{\mathcal{A}_1}(\beta) \in$ *h-traces*$(\mathcal{A}_2)$. Since $\mathcal{A}_1$ and $\mathcal{A}_2$ have the same external interface, $\pi_{\mathcal{A}_1}(\beta) = \pi_{\mathcal{A}_2}(\beta)$. Thus it follows from Lemma 5.4 that $\beta \in$ *h-traces*$(\mathcal{A}_2 \| \mathcal{B})$. ■

Two natural hiding operations can be defined on any HA $\mathcal{A}$:

(1) If $E \subseteq E_{\mathcal{A}} - \{e\}$, then $\mathsf{ActHide}(E, \mathcal{A})$ is the HA $\mathcal{B}$ that is equal to $\mathcal{A}$ except that $E_{\mathcal{B}} = E_{\mathcal{A}} - E$ and $H_{\mathcal{B}} = H_{\mathcal{A}} \cup E$.

(2) If $W \subseteq W_{\mathcal{A}}$, then $\mathsf{VarHide}(W, \mathcal{A})$ is the HA $\mathcal{B}$ that is equal to $\mathcal{A}$ except that $W_{\mathcal{B}} = W_{\mathcal{A}} - W$ and $X_{\mathcal{B}} = X_{\mathcal{A}} \cup W$.

Proposition 5.6 *Let $E \subseteq E_{\mathcal{A}} - \{e\}$ and $W \subseteq W_{\mathcal{A}}$. Then* $\mathsf{ActHide}(E, \mathcal{A})$ *and* $\mathsf{VarHide}(W, \mathcal{A})$ *are HAs.*

Proof: Straightforward application of the definitions. ■

Theorem 5.7 *Suppose $\mathcal{A}$ and $\mathcal{B}$ are HAs with $\mathcal{A} \leq \mathcal{B}$, and let $E \subseteq E_{\mathcal{A}} - \{e\}$ and $W \subseteq W_{\mathcal{A}}$. Then* $\mathsf{ActHide}(E, \mathcal{A}) \leq \mathsf{ActHide}(E, \mathcal{B})$ *and* $\mathsf{VarHide}(W, \mathcal{A}) \leq \mathsf{VarHide}(W, \mathcal{B})$.

Proof: Trivial argument since for each hybrid automaton $\mathcal{A}$, each $E \subseteq E_{\mathcal{A}} - \{e\}$, and each $W \subseteq W_{\mathcal{A}}$, *h-traces*$(\mathsf{ActHide}(E, \mathcal{A}))$ and *h-traces*$(\mathsf{VarHide}(W, \mathcal{A}))$ can be expressed as functions of *h-traces*$(\mathcal{A})$. ■

6 Hybrid I/O Automata

In this section we specialize the hybrid automaton model of Section 3 by adding a distinction between input and output. We show that the results on simulation relations, composition and hiding presented in Section 4 and 5 carry over straightforwardly to this new setting.

6.1 Hybrid I/O Automata

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

- $\mathcal{H} = (W, X, E, H, \Theta, \mathcal{D}, \mathcal{T})$ is a hybrid automaton.
- U and Y partition W into *input* and *output* variables, respectively.
  Variables in $Z \triangleq X \cup Y$ are called *local*.
- I and O partition E into *input* and *output actions*, respectively, such that $e \in I$.
  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, for $s, s' \in val(V)$, $k \in val(U)$ and $a \in A$:[1]

  Init (Start states closed under change of input variables)
  $s \in \Theta \quad \Rightarrow \quad \exists r \in \Theta : r \lceil U = k \wedge r \lceil Y = s \lceil Y.$

  D1 (Input enabling)
  $a \in I \quad \Rightarrow \quad \exists r : s \xrightarrow{a} r.$

  D2 (An environment action that does not change inputs does not affect the state)
  $s \xrightarrow{e} s' \ \wedge \ s \lceil U = s' \lceil U \quad \Rightarrow \quad s = s'.$

  D3 (Discrete transitions do not depend on input variable changes)
  $s \xrightarrow{a} s' \quad \Rightarrow \quad \exists r : s \xrightarrow{a} r \ \wedge \ r \lceil U = k \ \wedge \ r \lceil Y = s' \lceil Y.$

The intuition captured by axioms **Init** and **D1-3** is that an HIOA is responsible for performing locally controlled actions and for modifying the values of its local variables, whereas its environment is responsible for performing input actions and modifying the values of the input variables. Axiom **D1**, which is just the input enabling axiom from the (untimed) I/O automaton model, says that an HIOA accepts each input action in each state. Axiom **D2** postulates that an environment action that does not affect the input variables cannot be 'detected' by the automaton and therefore leaves the state unchanged. Axiom **D3** states that there is no functional dependence between the input and the output variables of an HIOA during a transition. If there is an a-step from a state s to a state s', then, for any valuation k of the input variables, there also exists an a-step from s to a state r with an input part k and an output part equal to that of s'. We do not require the internal variables of s' and r to have the same values, because we want to allow an HIOA to record all the discrete changes to its input variables in its internal state. The technical use of axiom **D3** is to avoid cyclic constraints during the interaction of two systems. The axiom ensures that the composition of two HIOA's is still input enabled and that the environment can never block the output actions of a system. Axiom **Init** says that a system may not constrain the initial values of its input variables: if in a start state we change

[1] The axiom **Init** that we present here, and also the axioms **D1-3** below, are slightly different from the axioms that we introduced in the preliminary version of this paper [29]. The new axioms allow an HIOA to change the values of its internal variables if the environment modifies the input variables of the HIOA.

the input variables then there is a way to change the internal variables as well (while leaving the output variables unchanged) so that the result is again a start state.

To understand better the roles of axioms **D1**, **D2** and **D3** we show that these axioms imply axiom **D**, i.e., the existence of stuttering transitions. Later, in Example 6.8, we show more explicitly why axiom **D3** is necessary to guarantee that HIOAs are closed under composition.

Proposition 6.1 *Axioms* **D1**, **D2** *and* **D3** *imply axiom* **D**.

Proof: Let s be a state of an HIOA $\mathcal{A}$. Since the environment action e is an input action, axiom **D1** implies that there is a state r such that $s \xrightarrow{e} r$. From **D3** there is a state r' such that $r' \lceil U = s \lceil U$ and $s \xrightarrow{e} r'$. From **D2**, $r' = s$. Thus, $s \xrightarrow{e} s$, i.e., $\mathcal{A}$ admits stuttering environment transitions. ■

Notation In the sequel, the components of an HIOA $\mathcal{A}$ will often be denoted by $W_{\mathcal{A}}$, $U_{\mathcal{A}}$, $A_{\mathcal{A}}$, $\Theta_{\mathcal{A}}$, $\mathcal{H}_{\mathcal{A}}$, etc. We sometimes omit these subscripts, where no confusion is likely. Also, the components of an HIOA $\mathcal{A}_i$ will be denoted by W_i, U_i, A_i, Θ_i, $\mathcal{H}_i$, etc. We abuse notation by referring to an HIOA $\mathcal{A}$ as an HA whenever we intend to refer to $\mathcal{H}_A$.

Example 6.2 (Timed I/O automata) A timed I/O automaton $\mathcal{A}$ in the sense of [42] can be represented as a hybrid I/O automaton $\mathcal{A}'$ without external variables. The construction is essentially the same as for Example 3.1. Observe that in the absence of external variables axiom **D2** states that all transitions labeled with e are stuttering transitions, and axiom **D3** does not impose any restriction. Thus, the only nontrivial axiom is axiom **D1**, which imposes the input enabling constraint of timed I/O automata. ■

6.2 Hybrid Executions and Hybrid Traces

A hybrid execution of an HIOA $\mathcal{A}$ is a hybrid execution of $\mathcal{H}_{\mathcal{A}}$. Similarly, a hybrid trace of $\mathcal{A}$ is a hybrid trace of $\mathcal{H}_{\mathcal{A}}$.

We say that 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 hybrid traces of $\mathcal{A}_1$ are included in those of $\mathcal{A}_2$: $\mathcal{A}_1 \leq \mathcal{A}_2 \stackrel{\Delta}{=} h\text{-}traces(\mathcal{A}_1) \subseteq h\text{-}traces(\mathcal{A}_2)$.

Proposition 6.3 *Let* $\mathcal{A}_1$ *and* $\mathcal{A}_2$ *be two comparable HIOAs. Then* $\mathcal{H}_1$ *and* $\mathcal{H}_2$ *are comparable and* $\mathcal{A}_1 \leq \mathcal{A}_2$ *iff* $\mathcal{H}_1 \leq \mathcal{H}_2$.

Proof: Immediate from the definitions. ■

6.3 Simulation Relations

The definition of simulation for HIOAs is the same as for HAs. Formally, if $\mathcal{A}_1$ and $\mathcal{A}_2$ are two comparable HIOAs, then a simulation from $\mathcal{A}_1$ to $\mathcal{A}_2$ is a simulation from $\mathcal{H}_1$ to $\mathcal{H}_2$.

Theorem 6.4 *If $\mathcal{A}_1$ and $\mathcal{A}_2$ are comparable HIOAs and there is a simulation from $\mathcal{A}_1$ to $\mathcal{A}_2$, then $\mathcal{A}_1 \leq \mathcal{A}_2$.*

Proof: Immediate from the definition of simulation, Theorem 4.2, and Proposition 6.3. ■

6.4 Composition and Hiding

The definitions of composition and hiding for HIOAs build on the corresponding definitions for HAs, but also take 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 $\mathcal{A}_2$. However, unlike in the definition for HAs, there is no need to require that the initial conditions of $\mathcal{A}_1$ and $\mathcal{A}_2$ are consistent: we will see that this consistency is implied by axiom **Init** for HIOAs. Finally, in order that the composition operation might satisfy nice properties (such as those of Section 8) we establish a convention 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 and output variables, respectively, of $\mathcal{A}_1$ and $\mathcal{A}_2$ are disjoint.

Formally, we say that HIOAs $\mathcal{A}_1$ and $\mathcal{A}_2$ are *compatible* if, for $i \neq j$,

$$X_i \cap V_j = Y_i \cap Y_j = H_i \cap A_j = O_i \cap O_j = \emptyset.$$

Lemma 6.5 *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be compatible HIOAs. Then $\mathcal{H}_1$ and $\mathcal{H}_2$ are compatible HAs.*

Proof: The second condition of the definition of compatibility for HAs is satisfied trivially. We show that the first condition is satisfied as well. Let $s_i \in \Theta_i$, for $i = 1, 2$. Define

$$\begin{aligned} k_1 &= s_2 \lceil (U_1 \cap Y_2) \ \cup \ s_1 \lceil (U_1 - Y_2), \\ k_2 &= s_1 \lceil (U_2 \cap Y_1) \ \cup \ s_2 \lceil (U_2 - Y_1). \end{aligned}$$

From **Init** applied to $\mathcal{A}_i$, there exist states $r_i \in \Theta_i$ such that $r_i \lceil U_i = k_i$ and $r_i \lceil Y_i = s_i \lceil Y_i$. Since $\mathcal{A}_1$ and $\mathcal{A}_2$ are compatible, $V_1 \cap V_2$ can be partitioned into sets $U_1 \cap U_2$, $U_1 \cap Y_2$ and

$U_2 \cap Y_1$. Now we derive

$$\begin{aligned} r_1 \lceil (V_1 \cap V_2) &= r_1 \lceil (U_1 \cap U_2) \cup r_1 \lceil (U_1 \cap Y_2) \cup r_1 \lceil (U_2 \cap Y_1) \\ &= s_2 \lceil (U_1 \cap U_2) \cup s_2 \lceil (U_1 \cap Y_2) \cup s_1 \lceil (U_2 \cap Y_1) \\ &= r_2 \lceil (U_1 \cap U_2) \cup r_2 \lceil (U_1 \cap Y_2) \cup r_2 \lceil (U_2 \cap Y_1) \\ &= r_2 \lceil (V_1 \cap V_2) \end{aligned}$$

Hence r_1 and r_2 are compatible and we can define $s = r_1 \cup r_2$. From the definition of s, $s \lceil V_1 = r_1 \in \Theta_1$ and $s \lceil V_2 = r_2 \in \Theta_2$. ■

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 $\mathcal{A} = (\mathcal{H}, U, Y, I, O)$ given by

- $\mathcal{H} = \mathcal{H}_1 \| \mathcal{H}_2$,
- $U = (U_1 \cup U_2) - (Y_1 \cup Y_2)$, $Y = Y_1 \cup Y_2$,
- $I = (I_1 \cup I_2) - (O_1 \cup O_2)$, $O = O_1 \cup O_2$.

Proposition 6.6 *$\mathcal{A}_1 \| \mathcal{A}_2$ is an HIOA.*

Proof: Let $\mathcal{A}$ denote $\mathcal{A}_1 \| \mathcal{A}_2$. We need to show that $\mathcal{A}$ satisfies the properties of a hybrid I/O automaton (cf. Section 6.1). Proposition 5.1 implies that $\mathcal{H} = \mathcal{H}_1 \| \mathcal{H}_2$ is a hybrid automaton. By construction, U and Y form a partition of W, and I and O form a partition of E. We verify the axioms **Init** and **D1-3**. Suppose $s, s' \in val(V)$, $k \in val(U)$ and $a \in A$. For $i \in \{1, 2\}$, let $s_i = \pi_i(s)$, $s'_i = \pi_i(s')$, and $a_i = \pi_i(a)$.

Init Assume $s \in \Theta_{\mathcal{A}}$. For $i \in \{1, 2\}$, let $k_i = k \lceil (U_i - Y_{\neg i}) \cup s \lceil (U_i \cap Y_{\neg i})$, where $\neg 1 = 2$ and $\neg 2 = 1$. From **Init** applied to $\mathcal{A}_i$, there are states $r_i \in \Theta_i$ such that $r_i \lceil Y_i = s \lceil Y_i$ and $r_i \lceil U_i = k_i$. It is not hard to show that r_1 and r_2 are compatible, so we can define $r = r_1 \cup r_2$. From the definition of r, $r \lceil V_1 \in \Theta_1$, $r \lceil V_2 \in \Theta_2$, $r \lceil Y = s \lceil Y$, and $r \lceil U = k$. Furthermore, from the definition of composition, $r \in \Theta_{\mathcal{A}}$.

D1 Assume $a \in I_{\mathcal{A}}$. From **D1** applied to $\mathcal{A}_i$, for $i = 1, 2$, there are states r_i such that $s_i \xrightarrow{a_i}_{A_i} r_i$. Define input states $k_i = r_i \lceil (U_i - Y_{\neg i}) \cup r_{\neg i} \lceil (U_i \cap Y_{\neg i})$. From **D3** applied to $\mathcal{A}_i$, there are states r'_i such that $s_i \xrightarrow{a_i}_{A_i} r'_i$, $r'_i \lceil U_i = k_i$, and $r'_i \lceil Y_i = r_i \lceil Y_i$. From the definitions of the k_i's, r'_1 and r'_2 are compatible. Thus, let r be $r'_1 \cup r'_2$. From the definition of composition, $s \xrightarrow{a}_A r$.

D2 Assume $s \xrightarrow{e}_A s'$ and $s \lceil U = s' \lceil U$. We show first that, for $i \in \{1, 2\}$, $s_i \lceil Y_i = s'_i \lceil Y_i$. In fact, from **D3**, there are states r_i such that $r_i \lceil U_i = s_i \lceil U_i$ and $r_i \lceil Y_i = s'_i \lceil Y_i$. From **D2**, $r_i = s_i$. Thus, $s'_i \lceil Y_i = s_i \lceil Y_i$.

From this fact $s' \lceil Y_{\mathcal{A}} = s \lceil Y_{\mathcal{A}}$. Thus, for $i \in \{1, 2\}$, $s_i \lceil U_i = s'_i \lceil U_i$. From **D2** applied to $\mathcal{A}_i$, $s_i = s'_i$. Thus, $s = s'$.

D3 Assume $s \xrightarrow{a}_A s'$. For $i \in \{1,2\}$, let $k_i = k \lceil (U_i - Y_{\neg i}) \cup s \lceil (U_i \cap Y_{\neg i})$. From **D3** applied to $\mathcal{A}_i$, there is a state r_i such that $s_i \xrightarrow{a_i}_{A_i} r_i$, $r_i \lceil U_i = k_i$, and $r_i \lceil Y_i = s_i' \lceil Y_i$. From the definition of the k_i's, r_1 and r_2 are compatible. Thus, let r be $r_1 \cup r_2$. Then, $r \lceil U_{\mathcal{A}} = k$ and $r \lceil Y_{\mathcal{A}} = s' \lceil Y_{\mathcal{A}}$. Furthermore, from the definition of composition, $s \xrightarrow{a}_A r$. ■

Theorem 6.7 *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 compatible with $\mathcal{B}$. Then $\mathcal{A}_1 \| \mathcal{B} \leq \mathcal{A}_2 \| \mathcal{B}$.*

Proof: Since $\mathcal{A}_1 \leq \mathcal{A}_2$, $\mathcal{A}_1$ and $\mathcal{A}_2$ are comparable. Using this and the definition of composition, we infer that $\mathcal{A}_1 \| \mathcal{B}$ and $\mathcal{A}_2 \| \mathcal{B}$ are comparable. Now the result follows by combining Proposition 6.3, Lemma 6.5, Theorem 5.5, and the definition of composition. ■

Example 6.8 (Need for axiom D3) In this example we show that axiom **D3** is necessary to guarantee that HIOAs are closed under composition. Let u and y be variables of type integer. Let $\mathcal{A}_1$ be an HIOA with input variable u, output variable y, and an input action a, and let $\mathcal{A}_2$ be an HIOA with input variable y, output variable u, and an input action a. Let each state of $\mathcal{A}_1$ and $\mathcal{A}_2$ enable a self-loop transition with action e, and transitions with action a such that in the post-state of the transition the value of the output variable is the value of the input variable plus 1. Let the start states of $\mathcal{A}_1$ and $\mathcal{A}_2$ be the valuations that assign 0 to each variable. Observe that $\mathcal{A}_1$ and $\mathcal{A}_2$ satisfy all the properties of an HIOA except for **D3**. Now consider $\mathcal{A}_1 \| \mathcal{A}_2$. It is simple to observe that no state of $\mathcal{A}_1 \| \mathcal{A}_2$ enables a transition labeled with a, and thus that axiom **D1** is violated, since the post-state of any transition labeled with a should satisfy $y = u + 1$ and $u = y + 1$. That is, $y = u + 1 = (y + 1) + 1 = y + 2$. Axiom **D3** avoids exactly this type of cyclic dependency among variables of a composed HIOA in the post-state of a transition. ■

The definition of variable and action hiding extend trivially to any HIOA $\mathcal{A}$.

(1) If $O \subseteq O_{\mathcal{A}}$ then $\mathsf{ActHide}(O, \mathcal{A})$ is the HIOA $\mathcal{B}$ that is equal to $\mathcal{A}$ except that $O_{\mathcal{B}} = O_{\mathcal{A}} - O$ and $H_{\mathcal{B}} = H_{\mathcal{A}} \cup O$.

(2) If $Y \subseteq Y_{\mathcal{A}}$ then $\mathsf{VarHide}(Y, \mathcal{A})$ is the HIOA $\mathcal{B}$ that is equal to $\mathcal{A}$ except that $Y_{\mathcal{B}} = Y_{\mathcal{A}} - Y$ and $X_{\mathcal{B}} = X_{\mathcal{A}} \cup Y$.

Proposition 6.9 *If $O \subseteq O_{\mathcal{A}}$ and $Y \subseteq Y_{\mathcal{A}}$, then $\mathsf{ActHide}(O, \mathcal{A})$ and $\mathsf{VarHide}(Y, \mathcal{A})$ are HIOAs.*

Theorem 6.10 *Suppose $\mathcal{A}$ and $\mathcal{B}$ are HIOAs with $\mathcal{A} \leq \mathcal{B}$, and suppose $O \subseteq O_{\mathcal{A}}$ and $Y \subseteq Y_{\mathcal{A}}$. Then $\mathsf{ActHide}(O, \mathcal{A}) \leq \mathsf{ActHide}(O, \mathcal{B})$ and $\mathsf{VarHide}(Y, \mathcal{A}) \leq \mathsf{VarHide}(Y, \mathcal{B})$.*

7 Reduced HIOAs

Axiom **D3** requires that an HIOA include transitions for all discrete actions, for all possible values of input variables. In practical applications of the HIOA model, however [34, 27, 47, 45, 46, 28, 23, 24, 15, 26, 25], it is often inconvenient to describe all these transitions. In fact, most of these applications do *not* specify them all. Technically, the automata defined in these papers are not HIOAs, but rather *reduced HIOAs*, defined as follows.

A *reduced HIOAs* is a structure that is like an HIOA, except that axiom **D3** is required only for *input* actions, and input variables do not change during input transitions. Specifically, axiom **D3** for HIOAs is replaced by

D3' (Discrete input transitions do not depend on input variable changes)
$s \xrightarrow{a} s' \wedge a \in I \quad \Rightarrow \quad \exists r : s \xrightarrow{a} r \wedge r \lceil U = k \wedge r \lceil Y = s' \lceil Y.$

and a new axiom is imposed

D4 (Locally controlled actions do not affect input variables)
$s \xrightarrow{a} s' \wedge a \in L \quad \Rightarrow \quad s \lceil U = s' \lceil U.$

Moreover, we require reduced HIOAs also to satisfy another axiom, saying that input actions do not affect output variables.

D5 (Input actions do not affect output variables)
$s \xrightarrow{a} s' \wedge a \in I \quad \Rightarrow \quad s \lceil Y = s' \lceil Y.$

Unfortunately reduced HIOAs are not closed under parallel composition as shown in the following example.

Example 7.1 (Non-closure of reduced HIOAs under parallel composition) Let $\mathcal{A}$ and $\mathcal{B}$ be two reduced HIOAs. Suppose $\mathcal{A}$ has an input variable u, and $\mathcal{B}$ has an output variable y. Suppose a is an output action of $\mathcal{B}$, but not an action of $\mathcal{A}$. Then, in $\mathcal{A}\|\mathcal{B}$ axiom **D5** is not satisfied, since, due to Axiom **D3** applied to $\mathcal{A}$ with action e, variable u is allowed to change arbitrarily whenever action a occurs. Thus, $\mathcal{A}\|\mathcal{B}$ is not a reduced HIOA. ■

In the rest of this section we show that, despite the non-closure under parallel composition of reduced HIOAs, it is possible to use them safely if they are used in contexts that do not have any input variables, i.e., if we compose them in parallel with other reduced HIOAs so that the resulting structure does not have any input variable. The result is easily extendable to contexts that involve hiding as well.

Before going further into details we need a new definition.

Definition 7.2 *Given an HIOA $\mathcal{A}$, define reduce$(\mathcal{A})$ to be the structure obtained from $\mathcal{A}$ by removing every discrete transition labeled with a locally controlled action in which some input variable changes.*

Observe that if $\mathcal{A}$ satisfies axiom **D5**, then *reduce*$(\mathcal{A})$ is a reduced HIOA. Furthermore, observe that any reduced HIOA $\mathcal{A}$ can be obtained from some HIOA $\mathcal{A}'$ through a *reduce* operation: in $\mathcal{A}'$ we simply add the extra transitions that are required for the satisfaction of axiom **D3**.

The formal result that we prove is the following.

Proposition 7.3 *Let $\mathcal{A}$ and $\mathcal{B}$ be two HIOAs where $\mathcal{B}$ satisfies axiom* **D5**. *If $\mathcal{A}\|\mathcal{B}$ does not have any input variables, then $\mathcal{A}\|\mathcal{B} = reduce(\mathcal{A})\|\mathcal{B}$.*

Proof: From the definitions of parallel composition and *reduce*, the structures $\mathcal{A}\|\mathcal{B}$ and *reduce*$(\mathcal{A})\|\mathcal{B}$ may differ only in their discrete transition relations. Thus, we need to show that $\mathcal{A}\|\mathcal{B}$ and *reduce*$(\mathcal{A})\|\mathcal{B}$ have the same discrete transition relation. Observe that, since $\mathcal{A}\|\mathcal{B}$ does not have any input variables, the input variables of $\mathcal{A}$ are included among the output variables of $\mathcal{B}$, and the input variables of $\mathcal{B}$ are included among the output variables of $\mathcal{A}$; that is, $U_{\mathcal{A}} \subseteq Y_{\mathcal{B}}$ and $U_{\mathcal{B}} \subseteq Y_{\mathcal{A}}$. Suppose that (s, a, s') is a transition of $\mathcal{A}\|\mathcal{B}$. If a is not a locally controlled action of $\mathcal{A}$, then, from the definition of *reduce*, $(\pi_{\mathcal{A}}(s), \pi_{\mathcal{A}}(a), \pi_{\mathcal{A}}(s'))$ is a transition of *reduce*$(\mathcal{A})$, and therefore (s, a, s') is a transition of *reduce*$(\mathcal{A})\|\mathcal{B}$. On the other hand, if a is a locally controlled action of $\mathcal{A}$, then $\pi_{\mathcal{B}}(a)$ is an input action of $\mathcal{B}$, and since $\mathcal{B}$ satisfies **D5**, $s'\lceil Y_{\mathcal{B}} = s\lceil Y_{\mathcal{B}}$. Since $U_{\mathcal{A}} \subseteq Y_{\mathcal{B}}$, $s'\lceil U_{\mathcal{A}} = s\lceil U_{\mathcal{A}}$. Thus, from the definition of *reduce*, $(\pi_{\mathcal{A}}(s), \pi_{\mathcal{A}}(a), \pi_{\mathcal{A}}(s'))$ is a transition of *reduce*$(\mathcal{A})$, and therefore (s, a, s') is a transition of *reduce*$(\mathcal{A})\|\mathcal{B}$.

Suppose that (s, a, s') is a transition of *reduce*$(\mathcal{A})\|\mathcal{B}$. Then (s, a, s') is trivially a transition of $\mathcal{A}\|\mathcal{B}$ since, from the definition of *reduce*, the set of discrete transitions of *reduce*$(\mathcal{A})$ is a subset of the set of discrete transitions of $\mathcal{A}$. ■

From Proposition 7.3 we deduce that reduced HIOAs can be used safely in contexts without any input variables since all the transitions that we rule out are never taken.

8 Receptiveness

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

8.1 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 local nondeterminism. An I/O behavior can be viewed as a restricted type of HIOA with only "continuous" behavior and no discrete transitions.

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

- U is a set of *input* variables.
- Y is a set of *output* variables with $U \cap Y = \emptyset$. We write $V \triangleq U \cup Y$.
- $\mathcal{T} \subseteq \mathit{trajs}(V)$ is a nonempty, prefix closed set of trajectories satisfying:

  B1 (Functional dependence of outputs from previous inputs)
  $\forall t\ \forall \tau, \tau' \in \mathcal{T}$ with domain $[0, t]$:

  $$(\tau \lhd t) \downarrow U = (\tau' \lhd t) \downarrow U \quad \Rightarrow \quad \tau(t)\lceil Y = \tau'(t)\lceil Y.$$

  B2 (Freedom of inputs)
  $\forall \tau \in \mathit{max}(\mathit{trajs}(U))\ \exists \tau' \in \mathit{max}(\mathcal{T}) : \tau' \downarrow U \leq \tau.$

  B3 (NonZenoness)
  $\mathit{max}(\mathcal{T}) \subseteq \mathit{closed}(\mathcal{T}) \cup \mathit{full}(\mathcal{T}).$

Axiom **B1** states that the output at time t is fully determined by the inputs at times up to, but not including, t. In particular, there is an output state $l \in \mathit{val}(Y)$ such that all trajectories $\tau \in \mathcal{T}$ begin with l, i.e., $\tau(0)\lceil Y = l$. We refer to this l as the *initial output* of $\mathcal{P}$. Axiom **B2** expresses the idea that the input is a signal that is imposed by the environment and over which the system has no control. 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 ∞. Observe that the full trajectories of an I/O behavior are precisely the limits of the finite trajectories, as the following completeness property holds due to **B2** and **B1**:

$$\forall \tau \in \mathit{trajs}(V) : (\forall t \in \mathsf{T}^{\geq 0} : \tau \trianglelefteq t \in \mathcal{T}) \quad \Rightarrow \quad \tau \in \mathcal{T}.$$

Also observe that the set of closed trajectories in $\mathcal{T}$ can be derived from the set of finite, right-open trajectories in $\mathcal{T}$. Thus the set $\mathcal{T}$ is fully determined by the finite, right-open trajectories contained in it.

As a consequence, the I/O behaviors presented here can be viewed as a special case of the I/O behaviors of Sontag [43]. Sontag defines I/O behaviors in terms of a *response map* from input signals up to, but not including, time t to the output at time t, but this presentation is

equivalent to a definition in terms of trajectories over both inputs and outputs. Technically, we found it a bit easier to use trajectories in this paper. Trajectories are also taken as primitive in the related notion of *I/O dynamical systems* of Willems [48]. In [43], no assumptions are made about possible input signals and the length of maximal trajectories (our axioms **B2** and **B3**). However, [43] distinguishes the so-called $\mathcal{V}$-*complete* I/O behaviors, which are I/O behaviors that accept any input in a family of controls (i.e., input trajectories) $\mathcal{V}$. In [43], for continuous time systems, $\mathcal{V}$ is always taken to be the class of all essentially bounded measurable controls. This is a special case of what we allow in our I/O behaviors, since we also include the possibility that certain input signals are only accepted up to and including some finite time.

Notation In the sequel, the components of an I/O behavior $\mathcal{P}$ will be denoted by $U_{\mathcal{P}}$, $Y_{\mathcal{P}}$ and $\mathcal{T}_{\mathcal{P}}$, etc. Also, if no confusion can arise, the components of an I/O behavior $\mathcal{P}_i$ will be denoted by U_i, Y_i and $\mathcal{T}_i$, etc.

The notions of compatibility and composition are defined for I/O behaviors just as for HIOAs. Formally, two I/O behaviors $\mathcal{P}_1$ and $\mathcal{P}_2$ are called *compatible* if $Y_1 \cap Y_2 = \emptyset$. In this case, we define the *composition* $\mathcal{P}_1 \| \mathcal{P}_2$ to be the structure $\mathcal{P} = (U, Y, \mathcal{T})$, where

- $U = (U_1 \cup U_2) - (Y_1 \cup Y_2)$;
- $Y = Y_1 \cup Y_2$;
- $\mathcal{T} \subseteq \mathit{trajs}(U \cup Y)$ is given by $\tau \in \mathcal{T} \Leftrightarrow \tau \downarrow V_1 \in \mathcal{T}_1 \wedge \tau \downarrow V_2 \in \mathcal{T}_2$.

In general, the composition of two compatible I/O behaviors need not be an I/O behavior since there may be "too many solutions" in the sense that there might be several distinct trajectories that projected onto the input variables give the same trajectory. This is illustrated by the following example.

Example 8.1 Suppose $\mathsf{T} = \mathsf{R}$ and u, y are 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. That is, $\mathsf{Copy}(u, y)$ is the I/O behavior $(\{u\}, \{y\}, \mathcal{T})$ where $\mathcal{T}$ is the set of trajectories τ such that $\tau(0)(y) = 0$ and, for each $t \in \mathit{dom}(\tau) \setminus \{0\}$, $\tau(t)(u) = \tau(t)(y)$. It is easy to verify that $\mathsf{Copy}(u, y)$ is an I/O behavior; in particular **B1** is guaranteed by the fact that the functions in the dynamic types of u and y have left-hand limits. Then the composition of $\mathsf{Copy}(u, y)$ and $\mathsf{Copy}(y, u)$ has no input variables and therefore there is just one full input trajectory. 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 outlined by the following example.

Example 8.2 Assume again that $\mathsf{T} = \mathsf{R}$ and u, y are 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 and their composition is an I/O behavior with just a single point trajectory. The composition does not allow any prolonged behavior to take place since there is no non-trivial trajectory that agrees with both $\mathsf{Add1}(u, y)$ and $\mathsf{Add1}(y, u)$. ■

If the composition of two I/O behaviors is an I/O behavior, then a trajectory of the composition can be maximal because the projection onto one of the components is maximal, or because no extension of the trajectory exists that is compatible with both components. The latter situation is excluded by following definition.

Two compatible I/O behaviors $\mathcal{P}_1$ and $\mathcal{P}_2$ are *strongly compatible* if $\mathcal{P} = \mathcal{P}_1 \| \mathcal{P}_2$ is an I/O behavior and, for each trajectory τ of $\mathcal{P}$,

C1 $\tau \in max(\mathcal{T}_{\mathcal{P}}) \quad \Rightarrow \quad \tau \downarrow V_1 \in max(\mathcal{T}_1) \vee \tau \downarrow V_2 \in max(\mathcal{T}_2)$.

The next definition associates I/O behaviors with HIOAs.

Let $\mathcal{A}$ be an HIOA and let $l \in val(Z_{\mathcal{A}})$ be a valuation of the local variables of $\mathcal{A}$. A set T of trajectories of $\mathcal{A}$ is called an *l-process* (or *process*) of $\mathcal{A}$ if $(U_{\mathcal{A}}, Z_{\mathcal{A}}, T)$ is an I/O behavior with initial output l.

Two compatible HIOAs $\mathcal{A}_1$ and $\mathcal{A}_2$ are *strongly compatible* if for each reachable state s of $\mathcal{A}_1 \| \mathcal{A}_2$, for each $(s \lceil Z_1)$-process T_1 of $\mathcal{A}_1$, and for each $(s \lceil Z_2)$-process T_2 of $\mathcal{A}_2$, the I/O behaviors (U_1, Z_1, T_1) and (U_2, Z_2, T_2) are strongly compatible.

8.2 Games and Strategies

Intuitively, a system is *receptive* if it can accept all the input provided by its environment and if time can advance to infinity irrespective of the input provided by its environment. Equivalently, a system is receptive if it accepts all input and does not rely on any specific behavior of its environment to let time advance forever. This informal explanation is not entirely correct though since, for example, there is no way to accept all input and let time advance to infinity if the environment provides input in a Zeno manner. In [14, 1, 42], various notions of receptivity have been defined formally in terms of games. Below, we extend these ideas to the setting of hybrid systems. The interaction between a system and its environment is represented as a two-player game in which the goal of the system is to construct an admissible execution, and the goal of the environment is to prevent such a construction. 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 an HIOA $\mathcal{A}$ is a function that specifies, for each sentence α of $\mathcal{A}$,

1. An l-process T^{α} of $\mathcal{A}$, where $l = \alpha.lstate \lceil Z_{\mathcal{A}}$.

2. A function $g^\alpha : \mathit{closed}(T^\alpha) \times I \times \mathit{val}(U) \to \mathit{val}(V)$ satisfying:

   S1 $g^\alpha(\tau, a, k) = s \quad \Rightarrow \quad \tau.\mathit{lstate} \xrightarrow{a} s \ \wedge \ s\lceil U = k.$

   S2 $g^\alpha(\tau, a, k)\lceil Y = g^\alpha(\tau, a, k')\lceil Y.$

3. A function $f^\alpha : \mathit{closed}(\mathit{max}(T^\alpha)) \times \mathit{val}(U) \to (L \times \mathit{val}(V))$ satisfying:

   S3 $f^\alpha(\tau, k) = (a, s) \quad \Rightarrow \quad \tau.\mathit{lstate} \xrightarrow{a} s \ \wedge \ s\lceil U = k.$

   S4 $f^\alpha(\tau, k) = (a, s) \ \wedge \ f^\alpha(\tau, k') = (a', s') \quad \Rightarrow \quad a = a' \ \wedge \ s\lceil Y = s'\lceil Y.$

At the beginning and immediately after each discrete transition, a strategy produces a process 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 and change discretely some input variables, a strategy also specifies, through the function g, what will be the next local state after such an event. Through the function f, a strategy specifies, for each closed maximal trajectory of the selected process, which transition will be performed at the end of this trajectory. Functions f and g are based also on the input that the environment may provide during the discrete transition performed by $\mathcal{A}$, so that the value of the input variables in the post-state of the returned transitions are those chosen by the environment. Axioms **S1** and **S3** ensure that a strategy always returns legal transitions that are consistent with the input provided by the environment; axioms **S2** and **S4** ensure that a strategy does not introduce functional dependencies between input and output variables during a transition: the system cannot react immediately to the changes in the values of the input variables. These axioms are closely connected to axiom **D3** for HIOAs.

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

$$\mathcal{I} \quad = \quad \tau_1 \ a_1 \ b_1 \ \tau_2 \ a_2 \ b_2 \ \cdots$$

of closed or maximal trajectories $\tau_i \in \mathit{trajs}(U)$, actions $a_i \in I$, and booleans $b_i \in \{\mathsf{true}, \mathsf{false}\}$.

In the i-th move of the game, the environment produces input signal τ_i. If τ_i is closed then the environment produces input action a_i right after signal τ_i. The boolean b_i serves to break ties in case the environment and the system both intend to perform a discrete action at the same time: if $b_i = \mathsf{true}$ then the environment is allowed to make a move and otherwise the system may perform an action. As in [42], our game starts after a finite execution α. The outcome of the game is described formally in the following definition.

Let $\mathcal{A}$ be an HIOA, ρ a strategy for $\mathcal{A}$, $\mathcal{I}$ an environment sequence for $\mathcal{A}$ (with ρ and $\mathcal{I}$ as defined above), and let α be a closed hybrid execution of $\mathcal{A}$. 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.

The hybrid execution α_0 is obtained by extending α in two steps to a sentence to which strategy ρ can be applied in combination with environment sequence $\mathcal{I}$. The first step adds a stuttering transition to α, thus generating a sentence; the second transition adds an environment transition using function g to account for the initial values of the input variables that the environment provides in $\mathcal{I}$. Formally, let $s = \alpha.lstate$ and $\alpha' = \alpha\ e\ \wp(s)$. Then

$$\alpha_0 \triangleq \alpha'\ e\ \wp(g^{\alpha'}(\wp(s), e, \tau_1(0))).$$

Observe that, from **D2**, if the input $\tau_1(0)$ is the same as $s\lceil U$, then α_0 is stuttering equivalent with α. For each $i > 0$, 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. Let τ_i' be the longest trajectory in $T^{\alpha_{i-1}}$ with $\tau_i' \downarrow U \leq \tau_i$. The existence of τ_i' is guaranteed by axiom **B2** after extending τ_i to any maximal trajectory, and the uniqueness by axiom **B1**; furthermore axiom **B3** tells us that τ_i' is either closed or full. Let $t = \tau_i.ltime$ and $t' = \tau_i'.ltime$. Then $t' \leq t$ since $\tau_i' \downarrow U \leq \tau_i$. We distinguish between three cases:

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

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

   This is the case where neither the system nor the environment performs a discrete action.

2. If $t = t' < \infty \wedge b_i = \mathsf{true}$, then

   $$\alpha_i \triangleq \alpha_{i-1} \frown (\tau_i'\ a_i\ \wp(g^{\alpha_{i-1}}(\tau_i', a_i, \tau_{i+1}(0)))).$$

   This is the case where after τ_i' the environment produces input action a_i. The resulting state after this action is obtained by applying the g-part of the strategy.

3. If $t' < t$ or $t = t' < \infty \wedge b_i = \mathsf{false}$ and if we let $f^{\alpha_{i-1}}(\tau_i', \tau_{i+1}(0)) = (a_i', s_i')$, then

   $$\alpha_i \triangleq \alpha_{i-1} \frown (\tau_i'\ a_i'\ \wp(s_i')).$$

   This is the case where, after τ_i' has been completed, the system performs a locally controlled transition as specified by the f-part of the strategy.

Proposition 8.3 *$\mathcal{O}_{\rho,\mathcal{I}}(\alpha)$ is a Zeno or admissible hybrid execution of $\mathcal{A}$.*

Proof: By construction the outcome $\mathcal{O}_{\rho,\mathcal{I}}(\alpha)$ is a hybrid execution of $\mathcal{A}$. According to the definition of outcome, either case 1 occurs at some point, or cases 2 and 3 occur infinitely often. If case 1 occurs, then $\mathcal{O}_{\rho,\mathcal{I}}(\alpha)$ is admissible; otherwise, $\mathcal{O}_{\rho,\mathcal{I}}(\alpha)$ contains infinitely many discrete actions, i.e., it is not a closed execution. ■

It is interesting to observe that in our definition of strategy functions f and g are based on sentences, while the definition of the outcome of a strategy is based on arbitrary closed hybrid executions. The reason behind this apparent inconsistency is that there is no need to consider general closed hybrid executions in the strategy definition, since, as we can see from the definition of α_0, there is an easy way to extend any closed hybrid execution to a sentence. On the other hand, defining strategies based on general closed hybrid executions would require to assert additional consistency conditions relating the strategies for hybrid executions obtained from other hybrid executions by extension with a single trajectory.

8.3 Composition of Strategies

Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be strongly compatible HIOAs and let ρ_1 and ρ_2 be strategies for $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively. The purpose of this section is to define a new strategy $\rho_1\|\rho_2$, the *composition* of ρ_1 and ρ_2. The composition uses ρ_1 to perform those parts of a move that are pertinent to $\mathcal{A}_1$, and uses ρ_2 for those parts of a move that are pertinent to $\mathcal{A}_2$. The main result that we prove (cf. Lemma 8.5) states that whenever an hybrid execution α is the outcome of $\rho_1\|\rho_2$ given a starting point α' and an environment sequence $\mathcal{I}$, it is possible, for $i = 1, 2$, to find an environment sequence $\mathcal{I}_i$ such that $\pi_i(\alpha)$ is the outcome of ρ_i given $\pi_i(\alpha')$ and $\mathcal{I}_i$. That is, it is like each component $\mathcal{A}_i$ has played according its strategy ρ_i during the game played according to $\rho_1\|\rho_2$. The composition of strategies that we define in this section will be used to show that receptiveness is compositional.

Before giving the formal definition of $\rho_1\|\rho_2$, we give some intuitions about its structure. Function $\rho_1\|\rho_2$ applied to a sentence α returns a process T^α and two functions g^α and f^α. The process T^α is obtained by composing processes $T_1^{\pi_1(\alpha)}$ and $T_2^{\pi_2(\alpha)}$. We are guaranteed that T^α is a process since $\mathcal{A}_1$ and $\mathcal{A}_2$ are strongly compatible.

Function g^α uses functions $g_1^{\pi_1(\alpha)}$ and $g_2^{\pi_2(\alpha)}$ to compute how $\mathcal{A}$ reacts to an input action. The definition of g^α is complicated by the fact that g_1 and g_2 have to agree on the common variables of $\mathcal{A}_1$ and $\mathcal{A}_2$. Informally, the computation of g is done as follows. First, g_1 and g_2 are used to figure out what should be the value of the output variables in the post-state of the transition returned by g (these are the y_i's in the following definition); then, based on the outputs just computed, g_1 and g_2 are used to determine the values of the internal variables in the post-state of the transition returned by g.

The definition of f^α is slightly more complicated. First of all f_1 and f_2 are used to determine which process among $\mathcal{A}_1$ and $\mathcal{A}_2$ moves first. In choosing among $\mathcal{A}_1$ and $\mathcal{A}_2$ we always give priority to $\mathcal{A}_1$; due to the symmetry of the problem our arbitrary decision does not compromise generality. Suppose that $\mathcal{A}_1$ moves first; the other case is symmetric. Then, f_1 and g_2 are

used to figure out what should be the value of the output variables in the post-state of the transition returned by f (these are the y_i's in the following definition); finally, based on the outputs just computed, f_1 and g_2 are used to determine the values of the internal variables in the post-state of the transition returned by f.

We are now ready for the formal definition. Let $\neg 1 = 2$ and $\neg 2 = 1$. We define the *composition* $\rho_1 \| \rho_2$ to be the function that associates to each sentence α of $\mathcal{A} = A_1 \| A_2$ the following objects:

1. $T^\alpha = \{\tau \in \mathcal{T}_\mathcal{A} \mid \pi_1(\tau) \in T_1^{\pi_1(\alpha)} \wedge \pi_2(\tau) \in T_2^{\pi_2(\alpha)}\}$.

2. A function $g^\alpha : \mathit{closed}(T^\alpha) \times I_\mathcal{A} \times \mathit{val}(U_\mathcal{A}) \to \mathit{val}(V_\mathcal{A})$ defined as follows.

   Let $(\tau, a, k) \in \mathit{dom}(g^\alpha)$. For $i \in \{1, 2\}$, let k_i be an arbitrary valuation for U_i, and let $y_i = g_i^{\pi_i(\alpha)}(\pi_i(\tau), \pi_i(a), k_i)\lceil Y_i$. Observe that, by **S2**, y_i is independent of the choice of k_i. This is the reason for our arbitrary choice of the k_i's. This same observation applies to the other places where we use arbitrary k_i's. Let $k_i' = (k \cup y_{\neg i})\lceil U_i$ and let $s_i' = g_i^{\pi_i(\alpha)}(\pi_i(\tau), \pi_i(a), k_i')$. Then

   $$g^\alpha(\tau, a, k) \triangleq s_1' \cup s_2'.$$

   Note that s_1' and s_2' are compatible:

   $$\begin{aligned} s_1'\lceil((Y_1 \cap U_2) \cup (Y_2 \cap U_1)) &= (s_1'\lceil Y_1)\lceil U_2 \cup (s_1'\lceil U_1)\lceil Y_2 \\ \text{(by \textbf{S1} and \textbf{S2})} &= y_1\lceil U_2 \cup k_1'\lceil Y_2 \\ &= y_1\lceil U_2 \cup ((k \cup y_2)\lceil U_1)\lceil Y_2 \\ &= y_1\lceil U_2 \cup y_2\lceil U_1 \\ \text{(by symmetric reasoning)} &= s_2'\lceil((Y_1 \cap U_2) \cup (Y_2 \cap U_1)) \end{aligned}$$

3. A function $f^\alpha : \mathit{closed}(\mathit{max}(T^\alpha)) \times \mathit{val}(U_\mathcal{A}) \to (L_\mathcal{A} \times \mathit{val}(V_\mathcal{A}))$ defined as follows.

   Let $(\tau, k) \in \mathit{dom}(f^\alpha)$. We distinguish between two cases:

   - $\pi_1(\tau) \in \mathit{max}(T_1^{\pi_1(\alpha)})$.

     Let k_1 be any valuation for U_1, let $f_1^{\pi_1(\alpha)}(\pi_1(\tau), k_1) = (a, s_1)$ and let $y_1 = s_1\lceil Y_1$. Let $k_2 = (k \cup y_1)\lceil U_2$, let $s_2 = g_2^{\pi_2(\alpha)}(\pi_2(\tau), \pi_2(a), k_2)$ and let $y_2 = s_2\lceil Y_2$. Finally, let $k_1' = (k \cup y_2)\lceil U_1$ and let $f_1^{\pi_1(\alpha)}(\pi_1(\tau), k_1') = (a, s_1')$. Then

     $$f^\alpha(\tau, k) \triangleq (a, s_1' \cup s_2).$$

     Note that s_1' and s_2 are compatible:

     $$\begin{aligned} s_1'\lceil((Y_1 \cap U_2) \cup (Y_2 \cap U_1)) &= (s_1'\lceil Y_1)\lceil U_2 \cup (s_1'\lceil U_1)\lceil Y_2 \\ \text{(by \textbf{S3} and \textbf{S4})} &= y_1\lceil U_2 \cup k_1'\lceil Y_2 \end{aligned}$$

$$
\begin{aligned}
&= ((k \cup y_1)\lceil U_2)\lceil Y_1 \cup ((k \cup y_2)\lceil U_1)\lceil Y_2 \\
&= k_2\lceil Y_1 \cup y_2\lceil U_1 \\
\text{(by } \mathbf{S1}) \quad &= (s_2\lceil U_2)\lceil Y_1 \cup (s_2\lceil Y_2)\lceil U_1 \\
&= s_2\lceil((Y_1 \cap U_2) \cup (Y_2 \cap U_1))
\end{aligned}
$$

- $\pi_2(\tau) \in max(T_2^{\pi_2(\alpha)})$ and $\pi_1(\tau) \notin max(T_1^{\pi_1(\alpha)})$.
  This case is symmetric to the previous case by exchanging 1 and 2.

Note that the above case distinction is complete: Let $l = \alpha.lstate\lceil Z_{\mathcal{A}}$. Then, for $i \in \{1,2\}$, $T^{\pi_i(\alpha)}$ is an $(l\lceil Z_i)$-process of $\mathcal{A}_i$. Since $\mathcal{A}_1$ and $\mathcal{A}_2$ are strongly compatible, T^α is an l-process of $\mathcal{A}$ and all trajectories in $\tau \in T^\alpha$ satisfy axiom **C1**.

The next proposition asserts that $\rho_1\|\rho_2$ is indeed a strategy for $\mathcal{A}$. In addition, the proposition states some technical properties of $\rho_1\|\rho_2$ that will be useful in proving the main result of this section. The technical properties essentially explain how the strategies ρ_1 and ρ_2 are used in $\mathcal{A}_1$ and $\mathcal{A}_2$ to obtain the transitions returned by $\rho_1\|\rho_2$.

Proposition 8.4 *$\rho_1\|\rho_2$ is a strategy for $\mathcal{A}$. Furthermore, for each sentence α of $\mathcal{A}$, each trajectory $\tau \in closed(T^\alpha)$, each action a of $\mathcal{A}$, each state s of $\mathcal{A}$, each valuation k for $U_{\mathcal{A}}$, and each $i \in \{1,2\}$,*

a. If $a \in I_{\mathcal{A}}$ and $g^\alpha(\tau,a,k) = s$, then $\pi_i(s) = g_i^{\pi_i(\alpha)}(\pi_i(\tau),\pi_i(a),s\lceil U_i)$.

b. If $\tau \in max(T^\alpha)$, $\pi_i(a) \in I_i$, and $f^\alpha(\tau,k) = (a,s)$, then $\pi_i(s) = g_i^{\pi_i(\alpha)}(\pi_i(\tau),\pi_i(a),s\lceil U_i)$.

c. If $\tau \in max(T^\alpha)$, $a \in L_i$, and $f^\alpha(\tau,k) = (a,s)$, then $(a,\pi_i(s)) = f_i^{\pi_i(\alpha)}(\pi_i(\tau),s\lceil U_i)$.

Proof: We show that the three objects returned by $\rho_1\|\rho_2$ satisfy the conditions of the definition of a strategy, and the implications (a)-(c). Let α, τ, a, s and k be as above, and let $l = \alpha.lstate\lceil Z_{\mathcal{A}}$.

1. As observed already above, T^α is an l-process of $\mathcal{A}$.

2. Assume $a \in I_{\mathcal{A}}$ and $g^\alpha(\tau,a,k) = s$. Let, for $i \in \{1,2\}$, k_i be arbitrary valuations for U_i, and let $y_i = g_i^{\pi_i(\alpha)}(\pi_i(\tau),\pi_i(a),k_i)\lceil Y_i$. Let $k_i' = (k \cup y_{\neg i})\lceil U_i$ and let $s_i' = g_i^{\pi_i(\alpha)}(\pi_i(\tau),\pi_i(a),k_i')$. Then, by definition of g^α, $s = s_1' \cup s_2'$. Since axiom **S1** holds for ρ_i,

   $$s\lceil U_i = s_i'\lceil U_i = g_i^{\pi_i(\alpha)}(\pi_i(\tau),\pi_i(a),k_i')\lceil U_i = k_i'$$

   and thus

   $$\pi_i(s) = s_i' = g_i^{\pi_i(\alpha)}(\pi_i(\tau),\pi_i(a),k_i') = g_i^{\pi_i(\alpha)}(\pi_i(\tau),\pi_i(a),s\lceil U_i) \tag{1}$$

Since $U \subseteq U_1 \cup U_2$, and since for $i \in \{1,2\}$, $k'_i = s\lceil U_i$, $s\lceil U = k$. Furthermore, from Equation (1) and from **S1** applied to ρ_i, $\pi_i(\tau.lstate) \stackrel{\pi_i(a)}{\rightarrow}_{A_i} \pi_i(s)$. From the definition of composition, $\tau.lstate \stackrel{a}{\rightarrow}_A s$. This shows that ρ satisfies **S1**.

To show **S2**, suppose by contradiction that there are two valuations k, k' such that $g^\alpha(\tau, a, k)\lceil Y \neq g^\alpha(\tau, a, k')\lceil Y$. Then, there exists $i \in \{1,2\}$ such that $g^\alpha(\tau, a, k)\lceil Y_i \neq g^\alpha(\tau, a, k')\lceil Y_i$, i.e., $\pi_i(g^\alpha(\tau, a, k))\lceil Y_i \neq \pi_i(g^\alpha(\tau, a, k'))\lceil Y_i$. From Equation (1) we derive that $g_i^{\pi_i(\alpha)}(\pi_i(\tau), \pi_i(a), g^\alpha(\tau, a, k)\lceil U_i)\lceil Y_i \neq g_i^{\pi_i(\alpha)}(\pi_i(\tau), \pi_i(a), g^\alpha(\tau, a, k')\lceil U_i)\lceil Y_i$, which contradicts **S2** for ρ_i.

3. Assume $\tau \in max(T^\alpha)$ and $f^\alpha(\tau, k) = (a, s)$. By the axiom **C1** for strong compatibility, either $\pi_1(\tau) \in max(T_1^{\pi_1(\alpha)})$ or $\pi_2(\tau) \in max(T_2^{\pi_2(\alpha)})$. We only consider the first case, the other case being symmetric. From the definition of f^α we infer $a \in L_1$ and $\pi_2(a) \in I_2$. Let k_1 be any valuation for U_1, let $f_1^{\pi_1(\alpha)}(\pi_1(\tau), k_1) = (a, s_1)$, and let $y_1 = s_1\lceil Y_1$. Let $k_2 = (k \cup y_1)\lceil U_2$, let $s_2 = g_2^{\pi_2(\alpha)}(\pi_2(\tau), \pi_2(a), k_2)$, and let $y_2 = s_2\lceil Y_2$. Finally, let $k'_1 = (k \cup y_2)\lceil U_1$ and let $f_1^{\pi_1(\alpha)}(\pi_1(\tau), k'_1) = (a, s'_1)$. Then $s = s'_1 \cup s_2$. From **S3** applied to ρ_1 we infer $k'_1 = s'_1\lceil U_1$, and from **S1** applied to ρ_2 we infer $k_2 = s_2\lceil U_2$. Thus $k'_1 = s\lceil U_1$, $k_2 = s\lceil U_2$,

$$(a, \pi_1(s)) = f_1^{\pi_1(\alpha)}(\pi_1(\tau), s\lceil U_1) \tag{2}$$

and

$$\pi_2(s) = g_2^{\pi_2(\alpha)}(\pi_2(\tau), \pi_2(a), s\lceil U_2). \tag{3}$$

Since $U \subseteq U_1 \cup U_2$, $s\lceil U = k$. Furthermore, from **S3** applied to ρ_1 and from **S1** applied to ρ_2, $\pi_i(\tau.lstate) \stackrel{\pi_i(a)}{\rightarrow}_{A_i} \pi_i(s)$, and from the definition of composition, $\tau.lstate \stackrel{a}{\rightarrow}_A s$. This shows that ρ satisfies **S3**.

To show **S4**, let k, k' be two valuations for the input variables of $\mathcal{A}$, and let $(a, s) = f^\alpha(\tau, k)$ and $(a', s') = f^\alpha(\tau, k')$. We want to show that $a = a'$ and $s\lceil Y = s'\lceil Y$. From Equation (2), $(a, \pi_1(s)) = f_1^{\pi_1(\alpha)}(\pi_1(\tau), s\lceil U_1)$ and $(a', \pi_1(s')) = f_1^{\pi_1(\alpha)}(\pi_1(\tau), s'\lceil U_1)$. From **S4** applied to ρ_1, $a = a'$ and $\pi_1(s)\lceil Y_1 = \pi_1(s')\lceil Y_1$. From Equation (3), $\pi_2(s) = g_2^{\pi_2(\alpha)}(\pi_2(\tau), \pi_2(a), s\lceil U_2)$ and $\pi_2(s') = g_2^{\pi_2(\alpha)}(\pi_2(\tau), \pi_2(a), s'\lceil U_2)$. From **S1** applied to ρ_2, $\pi_2(s)\lceil Y_2 = \pi_2(s')\lceil Y_2$. Thus, $s\lceil Y = s'\lceil Y$. ■

We are now ready to prove the main result of this section. Roughly speaking, playing a game on $\mathcal{A}_1 \| \mathcal{A}_2$ using $\rho_1 \| \rho_2$ is like playing a game on $\mathcal{A}_1$ using ρ_1 and on $\mathcal{A}_2$ using ρ_2 under some appropriate environment sequences.

Lemma 8.5 *Suppose $\mathcal{A}_1$ and $\mathcal{A}_2$ are strongly compatible HIOAs, ρ_1 and ρ_2 are strategies for $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively, $\mathcal{I}$ is an environment sequence of $\mathcal{A} = \mathcal{A}_1 \| \mathcal{A}_2$, and α is a closed execution of $\mathcal{A}$. Let $\rho = \rho_1 \| \rho_2$ and $i \in \{1,2\}$. Then there exists an environment sequence $\mathcal{I}_i$ for $\mathcal{A}_i$ such that $\mathcal{O}_{\rho_i, \mathcal{I}_i}(\pi_i(\alpha)) = \pi_i(\mathcal{O}_{\rho, \mathcal{I}}(\alpha))$.*

Proof: Let $s = \alpha.lstate$, $\alpha' = \alpha\, e\, \wp(s)$, and let $\tau'_0\, a'_1\, \tau'_1\, a'_2 \cdots$ be the unique hybrid execution fragment of $\mathcal{A}$ with $\mathcal{O}_{\rho,\mathcal{I}}(\alpha) = \alpha'\, e\, \tau'_0\, a'_1\, \tau'_1\, a'_2 \cdots$. Let n be the number of τ'_i's if the τ'_i's are finite, and ∞ otherwise. Let $\mathcal{I}_i$ be any environment sequence $\tau_1\, a_1\, b_1\, \tau_2\, a_2\, b_2 \cdots$ for $\mathcal{A}_i$ such that for each $j > 0$, if $j <= n$ then

$$\begin{aligned}
\tau_j &\triangleq \tau'_{j-1} \downarrow U_i \\
a_j &\triangleq \begin{cases} a'_j & \text{if } a'_j \in I_i \\ e & \text{otherwise} \end{cases} \\
b_j &\triangleq \begin{cases} \mathsf{false} & \text{if } a'_j \in L_i \\ \mathsf{true} & \text{otherwise} \end{cases}
\end{aligned}$$

We claim that $\mathcal{I}_i$ satisfies the required property. The fact that the elements with index greater that n are defined arbitrarily causes no problem since those elements will never be used in the proof. Let $\alpha'_0, \alpha'_1, \ldots$ denote the intermediate hybrid execution fragments in the definition of $\mathcal{O}_{\rho,\mathcal{I}}(\alpha)$, and let $\alpha_0, \alpha_1, \ldots$ denote the corresponding fragments in the definition of $\mathcal{O}_{\rho_i,\mathcal{I}_i}(\pi_i(\alpha))$. We first show by induction on j that for each $j \leq n$, $\alpha_j = \pi_i(\alpha'_j)$, and then we use this result to complete the proof of the lemma.

For the base case, applying the definitions of $\mathcal{O}$ and of the τ'_i's gives

$$\begin{aligned}
\alpha'_0 &= \alpha'\, e\, \wp(s') \\
\alpha_0 &= \pi_i(\alpha')\, e\, \wp(r')
\end{aligned}$$

where $s' = \tau'_0(0)$, $r' = g_i^{\pi_i(\alpha')}(\wp(r), e, \tau_1(0))$ and $r = \pi_i(s)$. Since by definition of $\mathcal{O}$ and **S1**, $s' = g^{\alpha'}(\wp(s), e, s'\lceil U)$, Proposition 8.4.a gives

$$\pi_i(s') = g_i^{\pi_i(\alpha')}(\pi_i(\wp(s)), \pi_i(e), s'\lceil U_i) = g_i^{\pi_i(\alpha')}(\wp(r), e, \tau_1(0)) = r'.$$

This implies that $\alpha_0 = \pi_i(\alpha'_0)$.

For the inductive step, choose $j > 0$ and assume $\alpha_{j-1} = \pi_i(\alpha'_{j-1})$. We distinguish cases based on the definition of $\mathcal{O}_{\rho_i,\mathcal{I}_i}(\pi_i(\alpha))$ applied to α_{j-1}. If α_{j-1} is admissible, then the conclusion is trivial since α'_{j-1} is admissible as well, and thus $\alpha'_j = \alpha'_{j-1}$ and $\alpha_j = \alpha_{j-1}$.

Otherwise, α'_{j-1} and α_{j-1} are sentences. Let $\tau^*_j = \pi_i(\tau'_{j-1})$. From the definition of ρ and the fact that τ'_{j-1} is a trajectory of $T^{\alpha'_{j-1}}$, it follows that τ^*_j is a trajectory of $T_i^{\alpha_{j-1}}$. Since

$$\tau^*_j \downarrow U_i = \pi_i(\tau'_{j-1}) \downarrow U_i = \tau'_{j-1} \downarrow U_i = \tau_j,$$

τ^*_j is in fact the longest trajectory of $T_i^{\alpha_{j-1}}$ with $\tau^*_j \downarrow U_i \leq \tau_j$. Let $t = \tau_j.ltime$. There are three cases.

1. $t = \infty$.

   In this case τ_j^* and τ'_{j-1} are admissible, $\alpha'_j = \alpha'_{j-1} \frown \tau'_{j-1}$, and $\alpha_j = \alpha_{j-1} \frown \tau_j^*$. Thus,

   $$\alpha_j = \alpha_{j-1} \frown \tau_j^* = \pi_i(\alpha'_{j-1}) \frown \pi_i(\tau'_{j-1}) = \pi_i(\alpha'_{j-1} \frown \tau'_{j-1}) = \pi_i(\alpha'_j).$$

2. $t < \infty \wedge b_j = \mathsf{true}$.

   In this case

   $$\begin{array}{rcl} \alpha'_j & = & \alpha'_{j-1} \frown \tau'_{j-1}\, a'_j\, \wp(s'_j) \\ \alpha_j & = & \alpha_{j-1} \frown \tau_j^*\, a_j\, \wp(s_j) \end{array}$$

   where $s'_j = \tau'_j(0)$ and $s_j = g_i^{\alpha_{j-1}}(\tau_j^*, a_j, s'_j\lceil U_i)$. By induction hypothesis, $\alpha_{j-1} = \pi_i(\alpha'_{j-1})$. By definition $\tau_j^* = \pi_i(\tau'_{j-1})$. Since $b_j = \mathsf{true}$, $a'_j \notin L_i$ and therefore $a_j = \pi_i(a'_j)$. Thus, in order to prove that $\alpha_j = \pi_i(\alpha'_j)$, we are left to show that $\pi_i(s'_j) = s_j$. We distinguish between two cases.

   (a) $a'_j \in I_{\mathcal{A}}$. Then, by **S1**, $s'_j = g^{\alpha'_{j-1}}(\tau'_{j-1}, a'_j, s'_j\lceil U)$. Application of Proposition 8.4.a gives

   $$\pi_i(s'_j) \;=\; g_i^{\pi_i(\alpha'_{j-1})}(\pi_i(\tau'_{j-1}), \pi_i(a'_j), s'_j\lceil U_i) = g_i^{\alpha_{j-1}}(\tau_j^*, a_j, s'_j\lceil U_i) = s_j.$$

   (b) $a'_j \in L_{\mathcal{A}}$. Then, by **S3**, $(a'_j, s'_j) = f^{\alpha'_{j-1}}(\tau'_{j-1}, s'_j\lceil U)$. Now Proposition 8.4.b gives

   $$\pi_i(s'_j) \;=\; g_i^{\pi_i(\alpha'_{j-1})}(\pi_i(\tau'_{j-1}), \pi_i(a'_j), s'_j\lceil U_i) = g_i^{\alpha_{j-1}}(\tau_j^*, a_j, s'_j\lceil U_i) = s_j.$$

3. $t < \infty \wedge b_j = \mathsf{false}$.

   In this case

   $$\begin{array}{rcl} \alpha'_j & = & \alpha'_{j-1} \frown \tau'_{j-1}\, a'_j\, \wp(s'_j) \\ \alpha_j & = & \alpha_{j-1} \frown \tau_j^*\, a''_j\, \wp(s_j) \end{array}$$

   where $s'_j = \tau'_j(0)$ and $(a''_j, s_j) = f_i^{\alpha_{j-1}}(\tau_j^*, s'_j\lceil U_i)$. Furthermore, $b_j = \mathsf{false}$ implies $a'_j \in L_i$ and thus $a'_j \in L_{\mathcal{A}}$. This means that $(a'_j, s'_j) = f^{\alpha'_{j-1}}(\tau'_{j-1}, s'_j\lceil U)$.

   From Proposition 8.4.c,

   $$(a'_j, \pi_i(s'_j)) \;=\; f_i^{\pi_i(\alpha'_{j-1})}(\pi_i(\tau'_{j-1}), s'_j\lceil U_i) = f_i^{\alpha_{j-1}}(\tau_j^*, s'_j\lceil U_i) = (a''_j, s_j).$$

   This implies $\pi_i(a'_j) = a'_j = a''_j$ and $\pi_i(s'_j) = s_j$, and thereby $\pi_i(\alpha'_j) = \alpha_j$.

We are now left to show that $\mathcal{O}_{\rho_i,\mathcal{I}_i}(\pi_i(\alpha)) = \pi_i(\mathcal{O}_{\rho,\mathcal{I}}(\alpha))$. If n is finite, then $\alpha'_n = \mathcal{O}_{\rho,\mathcal{I}}(\alpha)$, and by definition of $\mathcal{O}$ α'_n is admissible. Since $\alpha_n = \pi_i(\alpha'_n)$, then also α_n is admissible. Thus, by definition of $\mathcal{O}$, $\alpha_n = \mathcal{O}_{\rho_i,\mathcal{I}_i}(\pi_i(\alpha))$, which suffices. If n is infinite, then it suffices to show that $\lim_{j\to\infty} \alpha_j = \pi_i(\lim_{j\to\infty} \alpha'_j)$. Since for each $j > 0$ $\alpha'_j = \pi_i(\alpha_j)$, $\lim_{j\to\infty} \alpha_j = \lim_{j\to\infty} \pi_i(\alpha'_j)$. The conclusion then follows directly from the continuity of the projection operator, which is easy to show. ■

8.4 Receptiveness and Compositionality

To define the notion of receptiveness we need to introduce one last concept of *Zeno-tolerant* hybrid execution. Informally, a hybrid execution is Zeno-tolerant if it is Zeno and its Zenoness is caused only by the fact that the environment provided input in a Zeno manner. The need for Zeno-tolerance will be clarified in Example 8.8.

We define a hybrid execution α of an HIOA $\mathcal{A}$ to be *Zeno-tolerant* iff it is Zeno, contains infinitely many input actions and only finitely many locally controlled actions.

Lemma 8.6 *Suppose $\mathcal{A}_1$ and $\mathcal{A}_2$ are compatible HIOAs and α is a hybrid execution of $\mathcal{A}_1 \| \mathcal{A}_2$ such that $\pi_1(\alpha)$ and $\pi_2(\alpha)$ are Zeno-tolerant. Then α is Zeno-tolerant.*

Proof: Suppose by contradiction that α is not Zeno-tolerant. By Lemma 5.2, α is Zeno, and thus, by definition of Zeno-tolerance, either α contains infinitely many locally controlled actions, or α contains finitely many input actions. In the first case, either $\pi_i(\alpha)$ or $\pi_2(\alpha)$ would contain infinitely many locally controlled actions, contradicting the fact that $\pi_1(\alpha)$ and $\pi_2(\alpha)$ are Zeno-tolerant. Thus, α contains finitely many actions. However, in this case both $\pi_1(\alpha)$ and $\pi_2(\alpha)$ would contain finitely many actions, thus contradicting again the fact that $\pi_1(\alpha)$ and $\pi_2(\alpha)$ are Zeno-tolerant. ■

We define a strategy ρ for $\mathcal{A}$ to be *Zeno-tolerant* if for each environment sequence $\mathcal{I}$ and for each closed execution α, $\mathcal{O}_{\rho,\mathcal{I}}(\alpha)$ is either admissible or Zeno-tolerant. We call $\mathcal{A}$ *receptive* if there exists a Zeno-tolerant strategy for $\mathcal{A}$.

Proposition 8.7 *Suppose $\mathcal{A}$ is a receptive HIOA. Then $\mathcal{A}$ is feasible.*

Proof: Let τ be any full trajectory over the input variables of $\mathcal{A}$, b any boolean value, and let $\mathcal{I}$ be $\tau\, e\, b\, \tau\, e\, b\, \tau\, e\, b \cdots$. Let ρ be any Zeno-tolerant strategy for $\mathcal{A}$, and let α be any closed hybrid execution of $\mathcal{A}$. Then $\mathcal{O}_{\rho,\mathcal{I}}(\alpha)$ is an admissible execution of $\mathcal{A}$ that extends α. ■

Zeno-tolerance is needed to deal with environments that provide inputs in a Zeno manner. Intuitively it should be sufficient to assume that the environment is never Zeno. However, this assumption is not sufficient to preserve receptiveness under composition. The following example, taken from [42], clarifies this point.

Example 8.8 Consider hybrid I/O automata $\mathcal{A}, \mathcal{B}$ with $I_{\mathcal{A}} = O_{\mathcal{B}} = \{b\}$ and $O_{\mathcal{A}} = I_{\mathcal{B}} = \{a\}$. Assume $\mathcal{A}$ starts by performing its output action a and $\mathcal{B}$ starts by waiting for some input. Furthermore, assume that both $\mathcal{A}$ and $\mathcal{B}$ respond to their n^{th} input with an output action exactly $1/2^n$ time units after the input has occurred.

Consider the following alternative definition of receptiveness, which assumes that the environment does not behave in a Zeno manner. Call an environment sequence $\tau_1 a_1 b_1 \tau_2 a_2 b_2 \cdots$

admissible if, for each $i > 0$, $\sum_{j \geq i} \tau_j.ltime = \infty$. Suppose we redefine an HIOA to be receptive iff there exists a strategy ρ such that, for each closed hybrid execution α and for each admissible environment sequence $\mathcal{I}$, $\mathcal{O}_{\rho,\mathcal{I}}(\alpha)$ is admissible. Using the new definition of "receptive" it is easy to see that $\mathcal{A}$ and $\mathcal{B}$ are receptive. However, the composition of $\mathcal{A}$ and $\mathcal{B}$ yields no admissible hybrid executions. In fact, it yields only a Zeno hybrid execution that blocks time. Thus, the composition of $\mathcal{A}$ and $\mathcal{B}$ is not receptive according to neither definitions of receptiveness. One might say that $\mathcal{A}$ and $\mathcal{B}$ "unintentionally collude" to generate a Zeno behavior: each of the HIOAs looks like a Zeno environment to the other. The main idea behind Zeno-tolerance is to exclude this type of collusions between a system and its environment. ■

We now come to the main result of this paper, stating that receptiveness is preserved by composition.

Theorem 8.9 *Suppose $\mathcal{A}_1$ and $\mathcal{A}_2$ are strongly compatible, receptive HIOAs. Then $\mathcal{A}_1 \| \mathcal{A}_2$ is receptive.*

Proof: Let ρ_1 and ρ_2 be Zeno-tolerant strategies for $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively. We prove that $\rho = \rho_1 \| \rho_2$ is a Zeno-tolerant strategy for $\mathcal{A} = \mathcal{A}_1 \| \mathcal{A}_2$.

By Proposition 8.4, it follows that ρ is a strategy for $\mathcal{A}$. Thus it suffices to show that, for each environment sequence $\mathcal{I}$ of $\mathcal{A}$ and for each closed execution α of $\mathcal{A}$, $\alpha' = \mathcal{O}_{\rho,\mathcal{I}}(\alpha)$ is either admissible or Zeno-tolerant.

Suppose that α' is not admissible. Then, by Proposition 8.3, it is Zeno. Therefore, by Lemma 5.2, $\pi_1(\alpha')$ and $\pi_2(\alpha')$ are also Zeno. Let $i \in \{1, 2\}$. By Lemma 8.5, there exists an environment sequence $\mathcal{I}_i$ such that $\mathcal{O}_{\rho_i,\mathcal{I}_i}(\pi_i(\alpha)) = \pi_i(\alpha')$. Since ρ_i is Zeno-tolerant, this implies that $\pi_i(\alpha')$ is Zeno-tolerant. Now apply Lemma 8.6 to conclude that α' is Zeno-tolerant. ■

With Theorem 8.9 we have decomposed the compositionality property of feasible HIOAs into two parts: strong compatibility and receptiveness. Strong compatibility deals with the compositionality of the continuous behaviors of an HIOA, while receptiveness, assuming strong compatibility, deals with the compositionality of the discrete part of an HIOA. In control theory strong compatibility corresponds to checking the consistency of different sets of differential equations. This is hard to check in general, and constitutes an important research topic within control theory. Receptiveness is a well-known issue within the theory of concurrency where it has been thorougly investigated [14, 1, 40, 42]. Thus, Theorem 8.9 decomposes the compositionality problem of HIOAs into two problems that are best suited to be analyzed within control theory and concurrency theory, respectively. This decomposition result is an important feature of our hybrid model, as it allows one to analyze each problem within its natural framework.

The compositionality results for the hiding operations are much easier to prove:

Theorem 8.10 *Suppose $\mathcal{A}$ is a receptive HIOA, $O \subseteq O_{\mathcal{A}}$ and $Y \subseteq Y_{\mathcal{A}}$. Then* $\mathsf{ActHide}(O, \mathcal{A})$ *and* $\mathsf{VarHide}(Y, \mathcal{A})$ *are receptive.*

Proof: It is sufficient to show that if ρ is a Zeno-tolerant strategy for $\mathcal{A}$, then ρ is a Zeno-tolerant strategy for $\mathsf{ActHide}(O, \mathcal{A})$ and $\mathsf{VarHide}(Y, \mathcal{A})$ as well. This proof is simply long and tedious, but it does not present any particular difficulty. ■

8.5 Strong Compatibility versus Compatibility

In order to apply Theorem 8.9, one has to establish that two HIOAs $\mathcal{A}_1$ and $\mathcal{A}_2$ are strongly compatible. As mentioned above, this is a difficult problem in general. Nevertheless, it is possible to identify certain classes of I/O behaviors for which strong compatibility reduces to compatibility. This means that if all processes of $\mathcal{A}_1$ and $\mathcal{A}_2$ are within such a class, the condition of strong compatibility in Theorem 8.9 reduces to the 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 following axiom **B4**, which strengthens axiom **B1**,

B4 $\forall \tau, \tau' \in \mathcal{T} : dom(\tau) = dom(\tau') \;\Rightarrow\; \tau \downarrow Y = \tau' \downarrow Y.$

It is routine to verify that two autistic I/O behaviors are strongly compatible iff they are compatible. From the perspective of classical control theory autistic I/O behaviors are definitely of no interest: why have an input if it is not used at all? In a hybrid setting, however, a system 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. *Linear hybrid automata* [3, 2], for instance, have no input variables and therefore, by direct application of **B1**, all their 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 [43]. 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

$$D \;\triangleq\; \left\{ \begin{array}{lcl} \dot{x} & = & f(x,u) \\ y & = & g(x) \end{array} \right.$$

where u, y, and x are the vectors of input, output, and internal 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 D, defined on a maximal neighborhood of 0, such that $x(0) = x_0$. Suppose that the dynamic type of each input variable in u is the set of all $\mathcal{C}^1$ functions that are defined in a left closed (possibly empty) interval. Consider the set T of all the solutions to D for each possible choice of x_0 and of $u(t)$. Observe that T is prefix closed (just define $u(t)$ on closed intervals). Let $(U, X \cup Y, T')$ be any I/O behavior such that $T' \subseteq T$. We say that $(U, X \cup Y, T')$ is an I/O behavior of D.

Consider now two systems, described by equations D_1 and D_2 with the same form as D, and suppose there are no common locally controlled variables in D_1 and D_2. The interaction between D_1 and D_2 can be described by a new set of equations D_3 obtained by considering together the equations of D_1 and D_2. If also the g functions of D_1 and D_2 are globally Lipschitz, then it is easy to show that D_3 can be represented in the same form as D where f and g are globally Lipschitz. Furthermore, the following proposition holds.

Proposition 8.11 *Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two I/O behaviors of D_1 and D_2, respectively. Then $\mathcal{P}_1$ and $\mathcal{P}_2$ are strongly compatible and $\mathcal{P}_1 \| \mathcal{P}_2$ is an I/O behavior of D_3.*

Proof: Denote $\mathcal{P}_1 \| \mathcal{P}_2$ by $\mathcal{P}_3$. Let l_1 and l_2 be the initial outputs of $\mathcal{P}_1$ and $\mathcal{P}_2$, respectively. We need to prove the following facts.

1. $\mathcal{T}_3$ is prefix closed.

   This follows immediately from the prefix closure of $\mathcal{T}_1$ and $\mathcal{T}_2$.

2. Each trajectory of $\mathcal{P}_3$ is a solution to D_3.

   Fix $i \in \{1, 2\}$. From the definition of $\mathcal{P}_i$, $\pi_i(\tau)$ is a solution to D_i. This means that τ satisfies all the equations of D_1 and of D_2, i.e., τ is a solution to D_3.

3. $\mathcal{P}_3$ satisfies **B3**.

   Suppose for the sake of contradiction that there is a maximal Zeno trajectory τ in $\mathcal{P}_3$. Then, $\pi_1(\tau)$ and $\pi_2(\tau)$ are Zeno trajectories in $\mathcal{P}_1$ and $\mathcal{P}_2$, respectively. By **B3** applied to $\mathcal{P}_1$ and $\mathcal{P}_2$, $\pi_1(\tau)$ and $\pi_2(\tau)$ are not maximal. Thus, there are two finite trajectories τ_1 and τ_2 in $\mathcal{P}_1$ and $\mathcal{P}_2$, respectively, defined in the interval $[0, \tau.ltime]$, such that $\pi_1(\tau) \leq \tau_1$ and $\pi_2(\tau) \leq \tau_2$. From the continuity of the solutions to D_3, the valuations $\tau_1(\tau.ltime)$ and $\tau_2(\tau.ltime)$ are determined by τ and are compatible. Thus the $[0, \tau.ltime]$-trajectory τ' defined by $\tau'(t) = \tau_1(t) \cup \tau_2(t)$ is a trajectory of $\mathcal{P}_3$ that extends τ. This contradicts the hypothesis that τ is maximal in $\mathcal{P}_3$.

4. $\mathcal{P}_3$ satisfies **B1**.

   Consider two trajectories τ_1 and τ_2 of $\mathcal{T}_3$ and a time $t \in dom(\tau_1) \cap dom(\tau_2)$ such that $(\tau_1 \lhd t) \downarrow U_3 = (\tau_2 \lhd t) \downarrow U_3$. We distinguish two cases.

   (a) $t = 0$. Let $i \in \{1, 2\}$. From Axiom **B1** applied to $\mathcal{P}_i$, $\tau_1(0)\lceil Y_i = \tau_2(0)\lceil Y_i$. This means that $\tau_1(0)\lceil Y_3 = \tau_2(0)\lceil Y_3$.

(b) $t > 0$. Since τ_1 and τ_2 are solutions to D_3, from the local existence theorem (ordinary calculus) $\tau_1 \lhd t = \tau_2 \lhd t$. From the continuity of τ_1 and τ_2 in their domain, $\tau_1(t) = \tau_2(t)$, i.e., $\tau_1(t)\lceil Y_3 = \tau_2(t)\lceil Y_3$.

5. $\mathcal{P}_3$ satisfies **B2**.

   Let τ_{U_3} be a maximal trajectory for the input variables of $\mathcal{P}_3$. Then τ_{U_3} is either full or Zeno with some unbounded variable. Let x_0 be $l_1 \cup l_2 \cup \tau_{U_3}(0)$. For $i \in \{1, 2\}$, from **B2** and the prefix closure of T_i, $\wp(\pi_i(x_0))$ is a trajectory of $\mathcal{P}_i$. Thus, $\wp(x_0)$ is a trajectory of $\mathcal{P}_3$. From the local existence theorem (ordinary calculus), there is a unique solution τ to D_3, defined in a maximal neighborhood of 0, with input τ_{U_3} and that starts from x_0. Also, again from the local existence theorem, for $i \in \{1, 2\}$, $\pi_i(\tau)$ is the unique maximal solution to D_i with input $\tau \downarrow U_i$. If τ is full or $\tau \downarrow U_1$ and $\tau \downarrow U_2$ are Zeno with some unbounded variable, then, from **B2** applied to $\mathcal{P}_1$ and $\mathcal{P}_2$, and from the uniqueness property of $\pi_1(\tau)$ and $\pi_2(\tau)$, there is a maximal prefix τ_1 of $\pi_1(\tau)$ in $\mathcal{P}_1$ and a maximal prefix τ_2 of $\pi_2(\tau)$ in $\mathcal{P}_2$. From the definition of composition, $\tau[0, min(\tau_1.ltime, \tau_2.ltime)]$ is a maximal prefix of τ in $\mathcal{P}_3$.

   If $\tau \downarrow U_1$ is Zeno with some unbounded variables and $\tau \downarrow U_2$ is Zeno without unbounded variables, then the input of $\pi_2(\tau)$ can be prolonged to a full input. From the local existence theorem (ordinary calculus), the unique maximal solution to the prolonged input is either $\pi_2(\tau)$ or a prolongation of $\pi_2(\tau)$, and thus the maximal prefix of τ can be defined as for the first case.

   If $\tau \downarrow U_1$ and $\tau \downarrow U_2$ are Zeno without unbounded variables, then, since τ is a maximal solution to D_3, and since all the variables of τ are continuous, there is at least one unbounded local variable in τ. Assume without loss of generality that one of such variables is in Y_1. From continuity, $\tau \downarrow U_1$ and $\tau \downarrow U_2$ can be prolonged to full inputs. Furthermore, since $\tau \downarrow Y_1$ contains an unbounded variable, the maximal solution to D_1 with the prolonged inputs is $\pi_1(\tau)$, and the maximal solution to D_2 with the prolonged inputs is either $\pi_2(\tau)$ or a prolongation of $\pi_2(\tau)$. In particular $\tau_1.ltime < \tau.ltime$. Then, the maximal prefix of τ can be defined as for the first case.

6. $\mathcal{P}_1$ and $\mathcal{P}_2$ satisfy axiom **C1**.

   Consider a maximal trajectory τ of $\mathcal{P}_3$. By **B3**, either τ is full or τ is closed. If τ is full, then its projections are maximal as well. If τ is closed, then suppose for the sake of contradiction that $\tau_1 = \pi_1(\tau)$ and $\tau_2 = \pi_2(\tau)$ are not maximal in $\mathcal{P}_1$ and $\mathcal{P}_2$, respectively. Let τ_U be any full extension of $\tau \downarrow U_3$. From the local existence theorem (ordinary calculus) there is a unique solution τ' to D_3 with input τ_U that has value $\tau(0)$ at time 0. Furthermore, $\tau \leq \tau'$. Let τ_1' denote $\pi_1(\tau')$ and τ_2' denote $\pi_2(\tau')$. Let $i \in \{1, 2\}$. Then $\tau_i \leq \tau_i'$. Since τ_i is not maximal, and since $\mathcal{P}_i$ satisfies **B2**, there is a trajectory τ_i'' in $\mathcal{P}_i$ such that $\tau_i < \tau_i'' \leq \tau_i'$. Thus, there is an extension of τ in $\mathcal{P}_3$, e.g., $\tau' \trianglelefteq min(\tau_1''.ltime, \tau_2''.ltime)$, a contradiction. ■

The conclusion that we derive from Proposition 8.11 is that strong compatibility reduces to 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 as a basis to define a class of processes for which strong compatibility reduces to compatibility.

Example 8.12 We reconsider the case of Lipschitz functions to show a naive choice of conditions on f, g, and u that does not work. If in addition to the global Lipschitz requirement for f and g we require u to be globally Lipschitz, then from ordinary calculus we know that all the solutions to D are global, i.e., defined on the full time domain. However, we cannot impose such a restriction on the dynamic types of input variables since otherwise we could not let systems interact: the output variables of a Lipschitz system, which can also be used as inputs in an interaction, are not necessarily Lipschitz. ■

Acknowledgment We thank Jan van Schuppen for constructive criticism on an early version of this paper.

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. [17], pages 209–229.

[4] R. Alur and T.A. Henzinger. Reactive modules. In *Proceedings of the 11th IEEE Syposium on Logic in Computer Science*, pages 207–218, 1996.

[5] R. Alur and T.A. Henzinger. Modularity for timed and hybrid systems. In *Proceedings of the Ninth International Conference on Concurrency Theory*, volume 1243 of *Lecture Notes in Computer Science*, pages 74–88. Springer-Verlag, 1997.

[6] R. Alur, T.A. Henzinger, and E.D. Sontag, editors. *Hybrid Systems III*, volume 1066 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.

[7] J.C.M. Baeten and W.P. Weijland. *Process Algebra*. Cambridge Tracts in Theoretical Computer Science 18. Cambridge University Press, 1990.

[8] D.J.B. Bosscher, I. Polak, and F.W. Vaandrager. Verification of an audio control protocol. In H. Langmaack, W.-P. de Roever, and J. Vytopil, editors, *Proceedings of the Third International School and Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'94),* Lübeck, Germany, September 1994, volume 863 of *Lecture Notes in Computer Science*, pages 170–192. Springer-Verlag, 1994.

[9] A. Bouajjani and O. Maler, editors. *Proceedings Second European Workshop on Real-Time and Hybrid Systems*, Grenoble, France, June 1995.

[10] M.S. Branicky. *Studies in Hybrid Systems: Modeling, Analysis, and Control.* PhD thesis, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA, USA, June 1995.

[11] M.S. Branicky. Analyzing and synthesizing hybrid control systems. In Rozenberg and Vaandrager [41], pages 74–113.

[12] J.W. de Bakker, C. Huizing, W.P. de Roever, and G. Rozenberg, editors. *Proceedings REX Workshop on Real-Time: Theory in Practice,* Mook, The Netherlands, June 1991, volume 600 of *Lecture Notes in Computer Science*. Springer-Verlag, 1992.

[13] R. De Nicola and F.W. Vaandrager. Action versus state based logics for transition systems. In I. Guessarian, editor, *Semantics of Systems of Concurrent Processes, Proceedings LITP Spring School on Theoretical Computer Science,* La Roche Posay, France, volume 469 of *Lecture Notes in Computer Science*, pages 407–419. Springer-Verlag, 1990.

[14] D. Dill. *Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits.* ACM Distinguished Dissertations. MIT Press, 1988.

[15] E. Dolginova and N.A. Lynch. Safety verification for automated platoon maneuvers: A case study. In Maler [35], pages 154–170.

[16] A. Fehnker. Automotive control revisited — linear inequalities as approximation of reachable sets. In Henzinger and Sastry [18], pages 110–125.

[17] R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, editors. *Hybrid Systems*, volume 736 of *Lecture Notes in Computer Science*. Springer-Verlag, 1993.

[18] T.A. Henzinger and S. Sastry, editors. *Proceedings First International Workshop on Hybrid Systems: Computation and Control (HSCC'98),* Berkeley, California, volume 1386 of *Lecture Notes in Computer Science*. Springer-Verlag, April 1998.

[19] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall International, Englewood Cliffs, 1985.

[20] Y. Kesten, Z. Manna, and A. Pnueli. Verification of clocked and hybrid systems. In Rozenberg and Vaandrager [41], pages 4–73.

[21] L. Lamport. What good is temporal logic? In R.E. Mason, editor, *Information Processing 83*, pages 657–668. North-Holland, 1983.

[22] L. Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, May 1994.

[23] C. Livadas. *Formal verification of safety-critical hybrid systems.* Master's thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, September 1997. Also, MIT/LCS/TR-730.

[24] C. Livadas and N.A. Lynch. Formal verification of safety-critical hybrid systems. In Henzinger and Sastry [18], pages 253–272.

[25] J. Lygeros and N.A. Lynch. On the formal verification of the TCAS conflict resolution algorithms. In *Proceedings 36th IEEE Conference on Decision and Control,* San Diego, CA, pages 1829–1834, December 1997. Extended abstract.

[26] J. Lygeros and N.A. Lynch. Strings of vehicles: Modeling and safety conditions. In Henzinger and Sastry [18], pages 273–288.

[27] N.A. Lynch. Modelling and verification of automated transit systems, using timed automata, invariants and simulations. In Alur et al. [6], pages 449–463.

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

[29] N.A. Lynch, R. Segala, F.W. Vaandrager, and H.B. Weinberg. Hybrid I/O automata. In Alur et al. [6], pages 496–510.

[30] N.A. Lynch and M.R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proceedings of the* 6^{th} *Annual ACM Symposium on Principles of Distributed Computing*, pages 137–151, August 1987. A full version is available as MIT Technical Report MIT/LCS/TR-387.

[31] N.A. Lynch and F.W. Vaandrager. Forward and backward simulations, I: Untimed systems. *Information and Computation*, 121(2):214–233, September 1995.

[32] N.A. Lynch and F.W. Vaandrager. Action transducers and timed automata. *Formal Aspects of Computing*, 8(5):499–538, 1996.

[33] N.A. Lynch and F.W. Vaandrager. Forward and backward simulations, II: Timing-based systems. *Information and Computation*, 128(1):1–25, July 1996.

[34] N.A. Lynch and H.B. Weinberg. Proving correctness of a vehicle maneuver: Deceleration. In Bouajjani and Maler [9].

[35] O. Maler, editor. *Proceedings International Workshop on Hybrid and Real-Time Systems (HART'97),* Grenoble, France, volume 1201 of *Lecture Notes in Computer Science*. Springer-Verlag, March 1997.

[36] O. Maler, Z. Manna, and A. Pnueli. From timed to hybrid systems. In de Bakker et al. [12], pages 447–484.

[37] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, 1992.

[38] A. Nerode and A. Yakhnis. Concurrent programs as strategies in games. In Y. Moschovakis, editor, *Logic from Computer Science*. Springer-Verlag, 1992.

[39] A. Pnueli and J. Sifakis, editors. *Special Issue on Hybrid Systems of Theoretical Computer Science,* 138(1). Elsevier Science Publishers, February 1995.

[40] J.M.T. Romijn and F.W. Vaandrager. A note on fairness in I/O automata. *Information Processing Letters*, 59(5):245–250, 1996.

[41] G. Rozenberg and F.W. Vaandrager, editors. *Lectures on Embedded Systems*, volume 1494 of *Lecture Notes in Computer Science*. Springer-Verlag, October 1998.

[42] 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.

[43] E.D. Sontag. *Mathematical Control Theory — Deterministic Finite Dimensional Systems*, volume 6 of *Texts in Applied Mathematics*. Springer-Verlag, 1990.

[44] F.W. Vaandrager and J.H. van Schuppen, editors. *Proceedings Second International Workshop on Hybrid Systems: Computation and Control (HSCC'99),* Berg en Dal, The Netherlands, volume 1569 of *Lecture Notes in Computer Science*. Springer-Verlag, March 1999.

[45] H.B. Weinberg. *Correctness of vehicle control systems: A case study.* Master's thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, February 1996. Also, MIT/LCS/TR-685.

[46] H.B. Weinberg and N.A. Lynch. Correctness of vehicle control systems: A case study. In *Proceedings of the 17th IEEE Real-Time Systems Symposium (RTSS'96),* Washington, D.C., pages 62–72. IEEE Computer Society Press, December 1996.

[47] H.B. Weinberg, N.A. Lynch, and N. Delisle. Verification of automated vehicle protection systems. In Alur et al. [6], pages 101–113.

[48] J.C. Willems. Models for dynamics. In U. Kirchgraber and H.O. Walther, editors, *Dynamics Reported, Volume 2*, pages 171–269. John Wiley & Sons Ltd and B.G. Teubner, 1989.

A Notational Conventions

a	action
b	boolean
e	the environment action
f	function, in particular local step function in strategy
g	function, in particular input step function in strategy
h	function
i, j	index
k	input state
l	local or output state
r, s	state
t	time point
u	input variable
v	variable
w	external variable
x	internal variable
y	output variable
z	local variable
A	set of actions
C	context
D	set of differential equations
E	set of external actions
F	set of functions
H	set of internal (hidden) actions
I	set of input actions
J	interval
L	set of locally controlled actions
O	set of output actions
R	(simulation) relation
S	set
T	set of trajectories
U	set of input variables
V	set of variables
W	set of external (Dutch: waarneembare) variables
X	set of internal variables
Y	set of output variables
Z	set of local variables

$\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$	hybrid (I/O) automaton
$\mathcal{D}$	set of discrete transitions
$\mathcal{H}$	hybrid automaton
$\mathcal{I}$	environment sequence
$\mathcal{P}$, $\mathcal{Q}$	I/O behavior
$\mathcal{T}$	set of trajectories
N	the natural numbers
R	the real numbers
T	the time axis
Z	the integers
V	the universe of variables
α	hybrid execution fragment
β	hybrid trace
γ	sequence
ι	the 'generic' internal action
λ	the empty sequence
π	projection function
ρ	strategy
σ	sequence
τ	trajectory
Θ	set of start states